Supervised Boundary Formation

Carol Orange

Supervised Boundary Formation

Supervised Boundary Formation

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rector Magnificus Prof. ir. K.F. Wakker,
in het openbaar te verdedigen ten overstaan van een commissie,
door het College van Dekanen aangewezen,
op maandag 31 oktober 1994 te 13.30 uur

door

Carol Marie ORANGE

Bachelor of Arts,
Reed College, Portland, Oregon

geboren te Salem, Verenigde Staten

Dit proefschrift is goedgekeurd door de promotoren:
Prof. dr. ir. F.C.A. Groen
Prof. dr. I.T. Young

This research was partially supported by The Netherlands Project Team for Computer Science Research (SPIN Project: Three Dimensional Image Analysis).

Portions of Chapters 4 and 5 are reprinted with permission from [OG93, OG94] ©1993, 1994 IEEE.

Published and distributed by:

Delft University Press
Stevinweg 1
2628 CN Delft
The Netherlands

Telephone +31 15 783254
Fax +31 15 781661

CIP–DATA KONINGLIJKE BIBLIOTHEEK, DEN HAAG

Orange, C.M.

Supervised Boundary Formation C.M. Orange – Delft:
Delft University Press. – Ill.
Thesis Delft University of Technology. With ref. – With summary in Dutch.
ISBN 90-407-1045-7
NUGI 841
Subject headings: image analysis, processing, segmentation.

Copyright ©1994 by C.M. Orange
Cover: Carol Orange and Hans van Herwijnen

All rights reserved.
No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without permission from the publisher: Delft University Press, Stevinweg 1, 2628 CN Delft, The Netherlands.

Printed in the Netherlands.

The camera is an instrument of detection. We photograph not only what we know, but also what we don't know.

Lisette Model

Contents

Chapter 1

Introduction

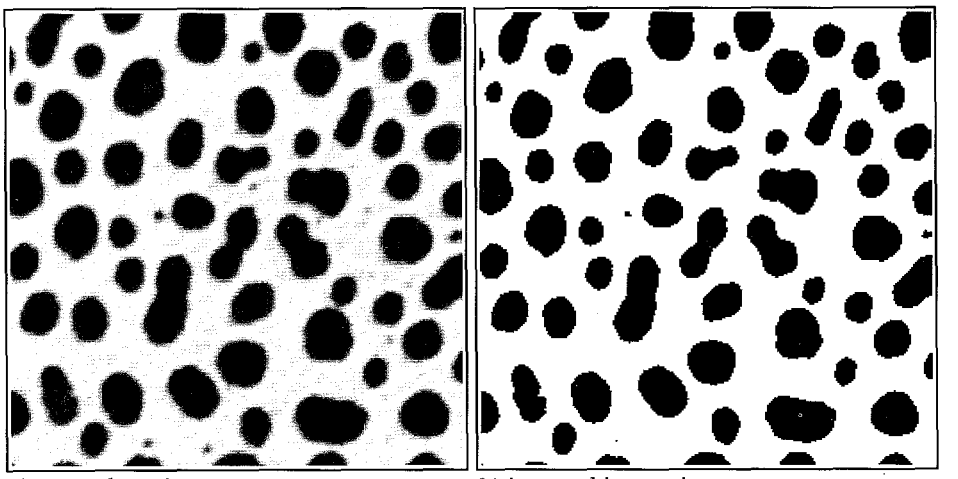
1.1 Image Segmentation

The segmentation of an image into meaningful parts is a key step in nearly every image analysis problem. It is crucial to the successful identification of image objects, and to the accuracy of object analysis such as shape and area. In general, a successful partitioning results in either a description of one or more regions associated with each object of interest, or a description of the boundary between each image object region and the remainder of the image. In most cases, either representation can be directly computed from the other. It is the manner in which the partitioning is extracted that distinguishes the two.

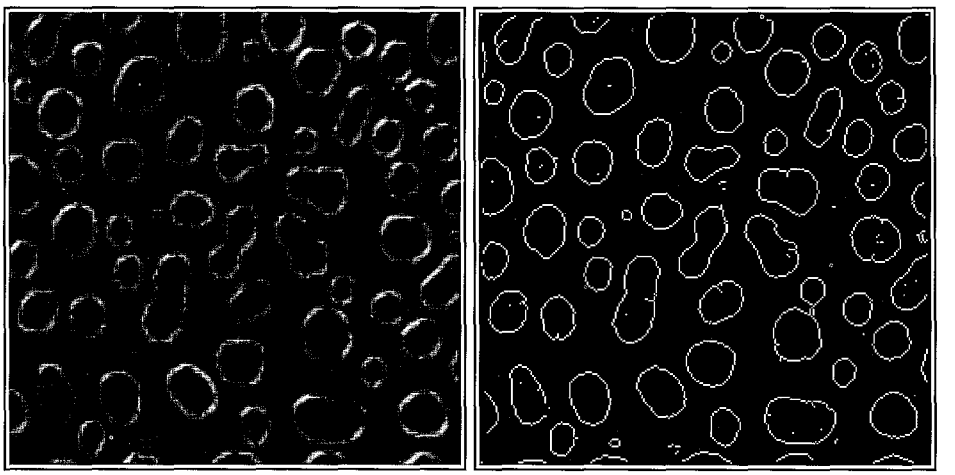
In region formation, a similarity measure is used to establish which parts of the image should be associated. Based on that measure, the image is separated into connected regions determined to be similar. One or more such regions may be identified as a segment of the image corresponding to a specific object. A simple example is when each image object is associated with a single connected region, and all objects are either darker or lighter than the background region which separates the objects. Such an image is shown in Figure 1.1a. In this case, the grey value intensity can be used as a similarity measure, and the object and background regions can be distinguished by selecting an appropriate intensity threshold, producing an image such as that shown in Figure 1.1b.

Alternatively, one might seek paths along which the image function changes significantly, thus indicating a boundary between an image object and its background. In boundary formation, one generally makes use of a difference measure which should have a strong response between the object and background regions and a low value in uniform regions. For the example in Figure 1.1a, this results in a figure like that in Figure 1.1c, which was obtained with the Prewitt difference operator [Pre70]. The boundary paths are then extracted based on the strength of the change in grey value intensity. Figure 1.1d, for example, shows the results obtained with the Hilditch skeleton [Hil69] applied after thresholding Figure 1.1c.

In spite of the apparent ease with which segmentation is accomplished with the human eye, it remains a central problem in computer vision [BB82, Pra91, GW92]. For



a) a grey level image b) image object regions



c) a grey level difference operator d) image object boundaries

Figure 1.1: Segmentation of a grey value image (a). In (b), image object regions are determined based on grey value intensity as a similarity measure. In (c), the response to an intensity difference measure is shown, and in (d) we show the paths of the image object boundaries as determined by thresholding (c) and thinning the resulting paths with a skeleton operator.

a limited set of applications in medical and industrial image analysis, a model-based approach to segmentation has proven effective. In [SD89] and [GAW93], for example, appropriate models for the expected objects are incorporated in the segmentation process.

Implicitly or explicitly, all region formation methods are based on some model of similarity which determines what parts of an image should be associated, and all boundary formation techniques are based on some difference model which determines where the boundary is. The effectiveness of the segmentation technique depends on how well the underlying segmentation model fits the image and problem at hand. For example, to obtain the results both in Figure 1.1b and in Figure 1.1d, we used the knowledge that the objects in the image in Figure 1.1a have a lower grey value intensity than the background.

A model used for image segmentation may also contain knowledge of the geometrical attributes of the object boundary, such as shape (e.g. circular) and size (e.g. radius r). This information may be used to direct and verify the partitioning of the image, and to adjust parameters (such as the threshold value), in the course of segmentation. It can also be used to correct the initial segmentation results. For example, the expected size of image objects may be used to eliminate small objects in Figure 1.1b, and short boundary paths in Figure 1.1d. Further, if image object boundaries are known to be smooth (curvature $\kappa > 1/r$) and connected, most of the object boundaries in Figure 1.1d can be corrected.

For the majority of images, sufficient models of the objects contained in the image have not been developed. Segmentation then requires some form of input from an expert. Traditionally, the role of the human expert in the segmentation process is limited to the selection of algorithms for detecting object boundaries or for defining object regions. In general, these operations do not result in the object boundaries or regions perceived by the expert. Much effort is thus spent trying to find the best combination of operations, and to correct unsatisfactory results.

In this thesis, we investigate techniques with which an expert can facilitate the segmentation process in a more direct and less frustrating manner. Because the human vision system is particularly sensitive to discontinuities in an image which may be due to object boundaries [Wat87], we want to develop interactive techniques for two dimensional path specification. *Direct manipulation* drawing tools have proven to be effective for path specification in packages such as MacDraw [Cla92], and Xfig [Sut85]. Similar techniques for object boundary specification, would allow a user to draw the path of an object boundary, rather than requiring a user to understand how the object differs from its background, and how this information can be translated to determine an effective segmentation technique.

Because the motor control capabilities of users vary and because hardware devices (such as mice) for screen location specification [HHN86] are indirect, a sketch acquired from a user cannot be considered more than a rough approximation to the image object boundary. Thus, simple tools for graphical interaction will not produce results from which we can reliably measure object features, such as area, average grey level, and

shape. Furthermore, interactive specification of boundaries is tedious at best, and the effort required of the user should be minimized.

Rather than expecting a human expert to provide an accurate description of an object boundary, we therefore aim to use the information obtained from the expert to develop a model of the boundary, based on the geometry of the sketch and an associated evaluation of the image function in the immediate region. This model is used in turn to produce the correct path of the object boundary. We thus introduce a collaborative approach to the segmentation problem, which we call *supervised boundary formation.*

1.2 Related Work

Assisting users in precision drawing has been a topic of interest to human computer interface (HCI) specialists since the origins of interactive drawing environments[1] [Sut63]. The grid technique available in most interactive drawing packages gives intersection points in a rectangular grid gravity, thereby moving any point specified by the user to the nearest grid point (see for example [Cla92]). This allows users to specify and line up rectangular objects in a precise way, but not to define geometrical shapes, such as equilateral triangles with arbitrary orientations, because these geometrical constraints cannot be specified in terms of a rectangular grid. Constraint based methods [Bor81, Bor86, Sut63], allow users to specify geometrical and other relationships among graphical objects in a scene with a large degree of freedom. Specification of such constraints is, however, awkward and time consuming. Snap-dragging, introduced by Bier in [BS86], allows users to control drawing with a compass and ruler. These techniques are easier to use because of the direct manipulation of the compass and ruler in the display of the drawing. Snap-dragging provides a compromise between grid-based and constraint-based techniques, by providing more geometric flexibility than grid based techniques, without the complexity of constraint based techniques. This technique has also proven useful in three dimensional drawing editors [Bie90].

Hudson introduced the concept of *semantic snapping* to make particular screen locations attractive based on semantics of an application which are unrelated to geometry [Hud90]. For example, within the context of visual programming, an icon representing a function may have connection points for input parameters and return values. If a connection is initiated from some function A, which returns a valid type for input to function B, the input connection on B's icon will be made attractive when the connection is initiated. If, on the other hand, B requires input of another type, the input connection will be made repellent when a connection is initiated at A.

If the concept of semantic snapping is applied to the problem of image segmentation, and in particular to the problem of image object boundary formation, then locations between image objects should be made more attractive as boundary points than those in the object and background regions. An example is found in [KWT88], where Kass, Witkin and Terzopoulos introduce an approach to boundary formation

[1] In fact, long before computers were used for drawing, rulers and compasses were used to solve the same problems for draftsmen using paper and pencil.

which they called *active contours*. They define an energy functional in terms of a set of constraints evaluated over the length of the object boundary. The optimal boundary is that for which the functional

$$E^*_{snake} = \int_0^1 E_{int}(\mathbf{v}(s)) + E_{image}(\mathbf{v}(s)) + E_{con}(\mathbf{v}(s))ds$$

is optimized. Contributing to the functional are the gradient magnitude (E_{image}), which has a high value along object boundaries in images such as Figure 1.1a, and measures of continuity and smoothness (E_{int}) over the length of the path. To support interactive tools for pulling a contour toward the user pointer, and pushing it away, they incorporate a set of external constraints (E_{con}) in the functional. This allows the user to influence the energy or cost of a path, and in so doing to easily select a path which satisfies the semantics of an object boundary, and which agrees with the path the user perceives as the object boundary.

1.3 Automatic Boundary Formation

The detection of discontinuities in a grey level image has been the traditional focus of research in boundary formation. If one views an image as a continuous intensity landscape, those areas corresponding to hills in the landscape may correspond to edges between light and dark regions, and local extrema may correspond to lines or points in the image. Traditionally, edges have been sought by inspecting the gradient

$$\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y} \right)$$

of the image function $f(x,y)$. In the continuous domain, the magnitude of the gradient is high in hilly regions, and the direction of the gradient is perpendicular to the path of the edge and pointing towards the hill top. Likewise, the Laplacian

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial^2 x} + \frac{\partial^2 f(x,y)}{\partial^2 y},$$

has a strong positive or negative response at extrema of the image function $f(x,y)$. Viewed in this fashion, it is logical that initial attempts at edge detection centered on the design of convolution kernels to approximate the gradient [FC77, Kir71, Pre70, Rob65, Rob76]. Likewise, detection of lines and dots was attempted using kernels which approximate the Laplacian (cf. [GW92, Pra91, RK76]).

The ideal result of a convolution with one of these kernels is a new image function which has high values at positions of edges, lines, or spots, and in smooth regions has the value zero. The determination of a path corresponding to an image object boundary based on the images resulting from these operations, however, remains problematic for a number of reasons.

1. **Localization error** Edges and lines are somewhat blurred in digital images due to the image formation process. Depending on the degree of blur, there may be multiple high responses perpendicular to a single edge or line.

2. **Noise sensitivity** Both the gradient and the Laplacian, being differentiation operators are very sensitive to noise. This may result in some boundary points having a weak response, while some nonboundary points have a strong response.

3. **Smoothness constraints** The detection of boundary points based on differentiation requires the path of the edge to be locally linear [MH80]. Key points such as corners and junctions, at which the path of the boundary is not smooth, will not be detected. If the image is blurred sufficiently, these points may be detected as edge points, but be poorly localized.

4. **Homogeneity assumptions** The measure with which an object may be distinguished from its background is restricted to differences in intensity. Not only will the boundaries between textured objects be missed because of this assumption, but many nonboundary points within textured regions will have a significant response to derivative based image operations. Detecting the discontinuities of the image function may be neither a useful nor a sufficient approach to extracting object boundaries in an unknown image.

5. **Data selection** In the vast majority of image segmentation problems, only a subset of the detected boundary points will be of interest. Given an image of a house and garden, one may be interested in deriving the dimensions of the house, or in identifying the plants in the garden. If the former, the boundaries of the plants are superfluous, and if the latter, the boundaries of the house are irrelevant.

6. **Interference** Depending on the spatial extent of the convolution filter, other image objects, shadows and reflections may interfere with the detection of sections of an object boundary.

In the field of edge detection, significant progress has been made towards solving the first two problems [GW92, Pra91]. In particular, the work of Canny [Can86] was central in its formalization of edge and line models upon which detection can be based, and in formalizing the goal of edge detection to be that of minimizing the localization error and sensitivity to noise simultaneously. For a limited set of homogeneity models, progress has also been made towards the detection of key boundary points such as corners and junctions at which the boundary does not satisfy the smoothness requirements assumed by the gradient model [MNR90, tHRFKV92].

The final three problems listed above, have only been solved for specific applications in which models have been developed containing the required information about the geometrical and homogeneity properties of the image objects of interest. For unknown images and new applications, these problems remain significant obstacles in the segmentation process.

Without a priori information, there is no basis for deciding which image features a detection operation should be modeled upon. Inherent to every boundary and region formation method is an assumption of the image feature measure which can be used to distinguish an object from its background. A given method is only applicable when the assumptions are well suited to the application and image at hand.

Even when appropriate boundary extraction techniques have been applied, given the set of resulting contour segments,[2] there is no way to automatically evaluate the data as to whether it is necessary and sufficient to describe the boundary of an image object. Many of the contour segments may be due to shadows and reflections caused by lighting conditions and object surface properties. In two dimensional images of three dimensional scenes, some objects may be partially occluded by others due to the position of the camera. In both two and three dimensional images of biological specimens, the grey value behavior associated with object boundaries may be extremely subtle. Mist in a natural scene, inconsistent lighting conditions, or significant noise in the imaging process may be impediments to identification of the proper set of contour segments to describe an object boundary. Fully automatic segmentation techniques which perform consistently and satisfactorily under the above mentioned conditions when no formal knowledge of image content is available have yet to be developed.

1.4 Human Boundary Formation

In general, a human observing an image will immediately segment it into meaningful parts. Reflections and shadows not only do not prevent the human eye from identifying an object, but play an important role in the recognition process. Object boundaries are immediately identified [Cor70, Wat87], often in spite of poor lighting conditions and significant noise. Further, the human vision system is robust to boundary sections being occluded by other image objects. In most cases, this will not result in an increased difficulty in recognition of the object in the scene. A classic example is the ability to instantly recognize a house when it is partially occluded by trees. Although the exact position of an image object boundary may be difficult to identify for some homogeneity criteria, a rough outline can generally be specified. For example, most people will correctly identify a bush in a field of grass, even if the grass and the bush are the same shade of green.

It is difficult, however, for a human observer to describe what change in an image lead her to perceive the presence of a boundary. Even when a person is able to describe the change, a question arises as to how this information can be communicated to the computer, and subsequently transformed to a set of homogeneity criteria which may be used to locate the boundary. Consider the problems involved in translating a human description of a texture pattern such as "small leaves in the wind" to homogeneity criteria which may be used to distinguish the bush from the grass in a digital image.

In summary, we may assume a user is able to provide information regarding the

[2] We use the term *contour segment* to refer to a geometrical representation of a section of an image object boundary.

approximate form and location of object contours in an image. However, one should not expect a user to specify the contour of an object to a satisfactory degree of precision for object analysis [HHN86]. A human expert can often indicate significant rather than negligible transitions between homogeneous regions,[3] and is able to associate a set of contour segments, whether connected or not with the boundary of a particular object. A human expert should not, however, be expected to specify the properties of the image function which can be used to separate an object of interest from the remainder of the image.

1.5 Supervised Boundary Formation

Claim: *While neither humans nor machines possess capabilities sufficient for the accurate segmentation of unknown images, their combined capabilities are sufficient for the accurate segmentation of a significant set of images which cannot be segmented by either independently.*

In terms of the problems listed in Section 1.3, the progress in automated and model-based boundary formation shows that the first two problems, localization error and sensitivity to noise, can be solved automatically, for images with dark objects on a bright background (and vice versa). If a good model for the boundary geometry is also available, the third problem, namely the detection of points at which a boundary is not smooth, can be solved automatically. This geometrical model can easily be furnished by a user if it is unavailable. Further, humans are able to identify the object boundaries for a wider set of homogeneity criteria, and can provide information on significant versus superfluous changes in an image, even if interference occurs along parts of the image object boundary.

The work in [KWT88] provides an example of interactive boundary editing tools in which the user provides this information when an object of interest is partially occluded by another image object. The interactive model presented there, while elegant, is limited to techniques for correcting previously extracted image object boundaries.

In this thesis, we investigate interactive models for boundary specification, assuming no estimation of the boundary has been computed. Because a human expert can be expected to indicate the approximate form and location of a transition between an object and its background, using tools similar to those provided in interactive drawing packages, we must address the following question.

Problem: *Based on an approximate location and geometry of an image object boundary, can the difference model which may be used to localize it be deduced?*

In practice, this would mean data gathered from the user would be used to construct a model for a section of the object contour, which would be used to develop and/or

[3] Any image phenomena which is not of interest for a particular segmentation task, be it due to a shadow, or a boundary of an object which is not of interest, is considered negligible.

select an appropriate detector to locate the object boundary. In the segmentation process, such a scheme would simultaneously exploit the image understanding inherent to the human vision system and the precision which can be obtained with a machine.

Input/Correction Models A common input model for path specification which has proven successful in interactive drawing packages is *connect-the-dots* model. Using a pointer tool (usually a mouse) the user specifies a series of (x, y) positions in a two dimensional plane. In most packages, a polygonal or smooth (spline) path can be generated based on the user input positions.

For image object boundary formation, a series of (x, y) locations can be used to generate a variety of paths. If the object boundary is smooth, the user may intend to specify a spline like path which follows the boundary, and interpolates some point in the neighborhood of each position specified. For polygonal objects, one may seek a corner near each point specified by the user similar to that defined by the user point set. Some correction of the boundary between the corner points may also be desired. Thus if the input is an ordered sequence of points, a variety of corrections may be applied to acquire the required image object boundary.

Another generally supported input model is *free-hand drawing.* In this case a path is traced as the user draws, which results in a sense of drawing with a pen on paper. The actual path generated is usually different than the set of points the user traces. Historically, computers have been too slow to store all points through which the user pointer passes, so some form of interpolation is used to produce a connected path. Even when this is no longer the case, however, some form of smoothing may be desirable.

In forming object boundaries, one wants the resulting path to be well connected, and fall between an object and its background. This might be approached by first deriving a path using traditional interpolation techniques, and subsequently correcting the result. Alternatively, one may want to produce a correct boundary as it is drawn. Other alternatives, such as a best polygonal approximation to the boundary based on some previously defined criteria, might be among the set of required boundary corrections.

Input tools which allow one to specify circles, ellipses and rectangles are available in most packages. In general, however, these geometric models are too strict to specify the boundary of an arbitrary object in an unknown image. Model-based approaches which incorporate an approximate geometric model in addition to an appropriate difference model have proven effective for extracting the boundary of objects with a simple geometrical form in [BD92].

User Interpretation Perhaps the biggest obstacle in the development of tools for supervised boundary formation is the question of user interpretation. Suppose a toolbox of input/correction techniques are at the user's disposal, and the user has selected one of them, how close, geometrically, can we assume the user sketch is to the object boundary? In a direct manner, this influences the error which the user should be permitted. More importantly, however, it determines whether we can model the properties of the image function near the boundary. Suppose we investigate a neigh-

borhood about the user sketch to model the image function in the object, boundary and background regions, and suppose the size of the neighborhood is determined by the permitted user error. If the permitted error is too small, the neighborhood about the user sketch defined may be too small to extract a model of all three regions, as one or more may not be present. If the neighborhood is too large, our image region evaluation may be disturbed by other objects in the region.

1.6 Scope of this Thesis

In Chapter 2, we perform experiments in which the user error near corner points on polygonally shaped objects, specified with a connect-the-dots input tool, is evaluated. This allows us to construct a user error model for the specification of points on similar objects. The evaluation of errors made at each corner of a user input polygon, however, requires the user polygon to be matched with the model polygon used in the user experiments. Because various users make a variety of errors in the specification of such polygons, the matching problem turns out to be nontrivial, and is addressed in Chapter 3. The user error model for corner specification which we derive in Chapter 2 is employed in Chapter 4 to derive corner models from user specified polygonal paths. Each corner model is used to localize the corner on the image object which corresponds to that specified by the user.

In Chapter 5, we turn our attention to a technique called *magnetic contour tracing*. In this case, a user traces a contour with a free-hand drawing tool, and the correct path of the boundary is produced as the user draws. To produce a path which follows the image object boundary, we develop a dynamic programming algorithm to attract the ink of the pen, as it were, to locations near the user with a high gradient magnitude. Using dynamic programming guarantees we produce a well behaved path in terms of connectivity and smoothness. By allowing the user to influence the boundary path definition while relying on gradient based techniques, we obtain a well localized boundary path, and overcome the problems of data selection and interference described in Section 1.3.

The methods described in this thesis were implemented using the Scil_Image package for image analysis [vBtKK+93, oA91]. Because software which supports the development of highly interactive methods for image analysis is not publicly available, we designed the GRIP library for image processing, which is described in Appendix A. All interactive methods described in this thesis were implemented with GRIP.

Chapter 2

Errors in Corner Specification

2.1 Introduction

For deriving the correct path of an object boundary from a user sketch, we first need to estimate the type and degree of errors a user makes in task of polygonal boundary specification. To sketch boundaries in our system, users are provided with a *connect-the-dots* tool, which is similar to those available in general purpose drawing packages like MacDraw [Cla92].

In light of Attneave's work [Att54], which showed that human perception is particularly sensitive to corners and local peaks in curvature, the geometry of the user sketch at each *dot* specified is likely to provide a good model for the geometry of a corner near the point. Likewise the boundary is likely to be relatively smooth between the dots. In this chapter, we thus aim to extract a model for errors made by users at corner points in the specification of polygonal object boundaries.

The error model is derived based on a set of experiments in which users are asked to sketch the (known) boundaries of objects in test images. In the terminology of user modeling (see [Cou92]), the user error model we extract is called an *explanatory user model* because it predicts the type and degree of user error based on actual user performance, rather than on a theoretical hypothesis (which corresponds to a *predictive* user model). Because the error model we derive turns out to differ for individual users, it should be viewed as a user dependent model which may be incorporated in an *adaptive* user interface (see [KDMSH92]), for boundary specification. More specifically, based on the user error model derived in this chapter, the method to extract corner points described in Chapter 4 can be adapted for individual users, increasing the likelihood of a correct corner model being extracted from the image.

In the following section, we state the questions which must be addressed to develop a user error model for polygonal boundary corner specification. In Section 2.3, we describe the experiments performed to measure user errors. The results are presented in Section 2.4, and in Section 2.5, we derive a user error model for corner specification.

2.2 Problem Statement

Suppose a user specifies an image object contour with a polygonal shape defined by the point set $P = \{p_i\}_{i=1}^{n}$. In practice, we treat the points p_1 and p_n as a single point if they are sufficiently close (thus a closed polygon) and as two distinct end points otherwise. Because we are interested in corner points, we simplify the discussion by assuming $p_n = p_1$, which means P describes a closed polygonal path. Now, for $1 \leq i < n$, each point p_i in the user sketch may be seen as an approximation of a corner on the object boundary. The geometric characteristics of the corner on P at the point p_i can be expressed in terms of the corner triplet $P_i = \{p_{i-1}, p_i, p_{i+1}\}$.[1]

We assume in the ideal world that the user has an interactive toolkit at hand for object contour specification which allows the user input to be interpreted and corrected in a variety of ways, just as with interactive drawing package one may sketch using a variety of tools. In this case, however, the user can choose both the input tool and the correction model. In this chapter, we assume the nature of the correction requested is an adjustment of the location of each point specified in the sketch. We therefore expect there to be an object in the image, the boundary of which can be described by a polygonal shape $Q = \{q_j\}_{j=1}^{m}$, such that for each corner $P_i \in P$, there is a corner $Q_j \in Q$ which is *close* to P_i. By close, we mean the differences between the key geometric characteristics of P_i and Q_j should be small.

2.2.1 Geometrical Characteristics of a Corner

Given a corner $P_i = \{p_{i-1}, p_i, p_{i+1}\}$, geometric characteristics which are relevant to the corner model are the position p_i, the corner angle $\alpha(P_i)$, and its orientation $\beta(P_i)$. The latter can be expressed in terms of the two vectors which define geometry of the corner on the sketch at p_i. Let

$$a_i = p_{i-1} - p_i \qquad \text{and} \qquad b_i = p_{i+1} - p_i, \tag{2.1}$$

and consider Figure 2.1. The angle of the corner on the path defined by P at the point p_i is given by

$$\alpha(P_i) = \cos^{-1}\left(\frac{a_i \cdot b_i}{\|a_i\|\|b_i\|}\right). \tag{2.2}$$

We use corner orientation to measure the direction of the cone section of the corner, and so we define it as the average direction of the vectors a_i and b_i, which determine the cone geometry. As can be seen in Figure 2.1, this is simply the direction of the vector which bisects the corner defined by a_i and b_i, defined by

$$B(P_i) = \frac{a_i}{\|a_i\|} + \frac{b_i}{\|b_i\|}. \tag{2.3}$$

[1] Due to the cyclic nature of polygonal point sets, if $i \geq n$, p_i should be interpreted as $p_{i \bmod (n-1)}$, and if $i \leq 0$, p_i should be read $p_{i+n-1 \bmod (n)}$.

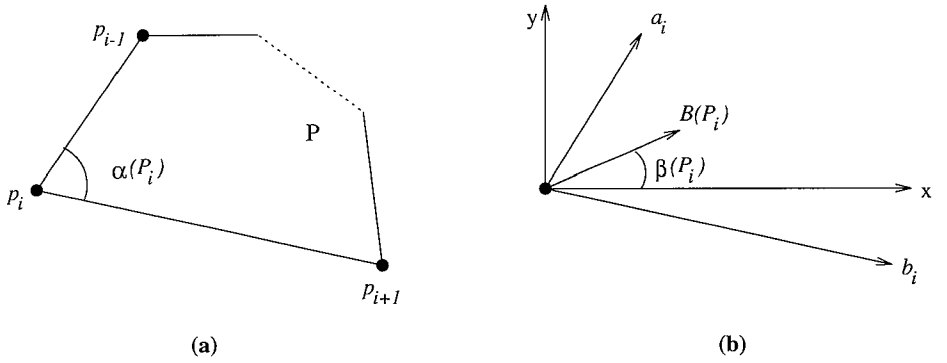


Figure 2.1: The geometric characteristics of a corner on a polygonal shape defined by (a) the triplet $P_i = \{p_{i-1}, p_i, p_{i-1}\}$ which determines the corner geometry of the path P at the point p_i, and (b) the vectors a_i and b_i derived from the triplet P_i.

Finally, the size or *scale* of the corner is significant for proper construction of a corner model. The following definition for corner scale provides a measure of the boundary detail near the corner.

Definition 2.1 *The* **scale** *of a corner P_i in a polygonal shape P is given by the length of the shorter of the two line segments which meet at the corner point p_i. If a_i and b_i are defined as in Equation 2.1, the scale is given by*

$$S(P_i) = \min\{\|a_i\|, \|b_i\|\}. \tag{2.4}$$

If Q_j is a corner triplet on an object boundary, and there are no disturbances in the boundary model due to nearby or occluding sources of interference, then the scale $S(Q_j)$ tells us at what distance from the corner point q_j the image function should look like a corner transition. This assumes other sections of the path of Q do not intersect the immediate neighborhood of q_j.

2.2.2 Corner Specification Errors

When a user produces a point set $P = \{p_i\}_{i=1}^n$ in specifying an object boundary defined by a polygonal shape $Q = \{q_j\}_{j=1}^m$, we are concerned with the errors made at each corner P_i. In particular, if the user point $p_i \in P$ corresponds to an object boundary point $q_j \in Q$, we are interested in user performance with respect to the following errors.

1. Positional Error - the Euclidean distance between the user specified point p_i, and the position q_j of the corner on the boundary, given by

$$\delta_{ij} = \| p_i - q_j \| . \tag{2.5}$$

2. Corner Angle Error - the difference between $\alpha(P_i)$, the angle of the corner in the user sketch at p_i, and $\alpha(Q_j)$, the angle of the corner in the image object at q_j. This is defined by
$$\delta\alpha_{ij} = \alpha(P_i) - \alpha(Q_j). \tag{2.6}$$

3. Orientation Error - the difference in corner orientation between P_i and Q_j. This can be measured with the corner angle between the bisecting vectors $B(P_i)$ and $B(Q_j)$ defined in Equation 2.3, which is given by
$$\delta\beta_{ij} = \cos^{-1}\left(\frac{B(P_i)\cdot B(Q_j)}{\|B(P_i)\|\|B(Q_j)\|}\right). \tag{2.7}$$

4. Scale Error - the difference between the scale $S(P_i)$ of the user defined corner, and the scale $S(Q_j)$ of the corresponding corner on the object boundary. The error is defined by
$$\delta S_{ij} = S(P_i) - S(Q_j). \tag{2.8}$$

The second and third errors depend on the error a user makes in the angle of each of the lines which meet at the corner. Thus, we might have examined errors in line angle rather than in corner angle and orientation. The geometry of the corner, however, depends on both lines which meet there. The corner angle and orientation capture the full corner geometry, and are therefore the relevant measures for study of errors at corner points.

2.2.3 Questions

Suppose a user specifies a corner triplet $P_i = \{p_{i-1}, p_i, p_{i+1}\}$ as part of an object boundary, the path of which is correctly described by $Q = \{q_j\}_{j=1}^m$. If $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$ is the corner triplet which corresponds to that specified by the user, we want to know if the user behavior with respect to the four errors defined above depends on the geometry of the object corner defined by Q_j. We also want to know if the error level varies among users or if it is influenced by the characteristics of the image function near the boundary. Specifically, we are concerned with the following.

For each of the four corner measures defined in Section 2.2.2, does the difference between the user defined corner triplet P_i and the image object corner triplet Q_j depend on one or more of the following factors?

A) The scale $S(Q_j)$ of the corner on the image object boundary;

B) The characteristics of the image function in the object, background and transition regions in the neighborhood of the corner point q_j;

C) Whether there is disturbance to the image function, such as noise and shading in the neighborhood of the point q_j;

D) Which user specified the sketch P of the image object boundary; and

E) The object shape.

Both δ_{ij} and $\delta\beta_{ij}$ are nonnegative entities, which by definition, produce an error magnitude. The definitions of $\delta\alpha_{ij}$ and δS_{ij} allow inspection to the bias of the user error. This will be used to form a model for user errors in Section 2.5.

2.3 The User Experiments

The experiments described in this section were designed to collect data on user errors made in corner specification. They are used to address the questions posed above.

2.3.1 The Session

The six subjects who produced the object boundary sketches were computer science graduate students and programmers, each of whom had substantial experience using a mouse. Prior to the session, all subjects were unaware of the questions to be addressed in the experiment. Each subject was presented a series of 20 images containing a polygonally shaped object, and asked to specify the object boundary.

A *connect-the-dots* tool was provided for the specification of polygonal boundaries. It should be familiar to those who have used interactive drawing packages to draw polygonal or curved paths by indicating a number of points, which are to be connected in the order specified. After the first point is specified, the movement of the cursor is tracked, and a line is drawn from the specified point to the current cursor location. The line is constantly updated as the user moves about. When a new point is specified, the line is made a permanent part of the polygonal shape. The process then continues with the new point as the starting point for the moving or *rubber-band* line. When the last point is specified, the object is complete.

Each subject was presented with the written instructions in Appendix 2.A. In addition to directions on the use of the tool for boundary specification, the instructions encourage the subjects not to specify the points with too much care. The session required approximately 15 minutes including time to read the instructions and practice with the tool.

2.3.2 The Stimuli

Each image in the series presented to the users contains an object with a boundary defined by one of the two polygonal shapes in Figure 2.2. Each shape was used to create a variety of ten 256×256 images. The ten images differ in the grey level functions $f_0(x, y)$, $f_1(x, y)$ and $f_b(x, y)$ in the object, background and boundary regions.

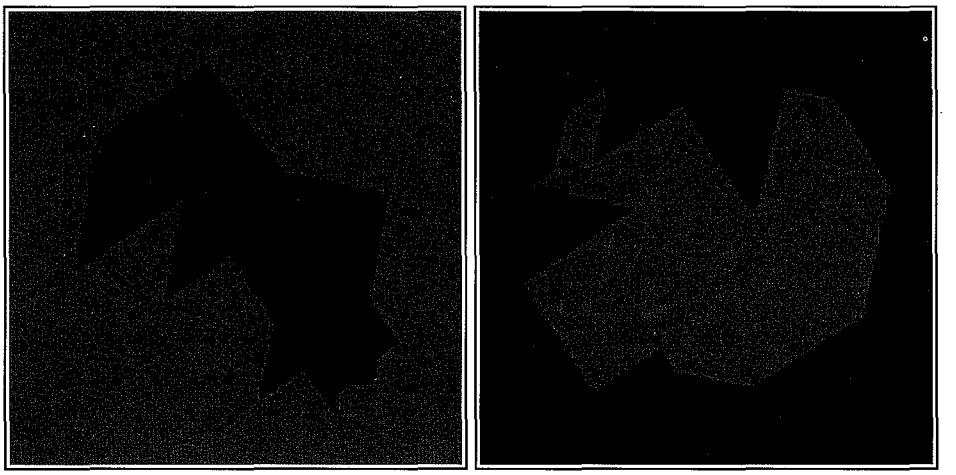


Figure 2.2: The shapes *pol1* (left) and *pol2* (right) used in the experiments.

Object Shapes

The two polygonal shapes used to define the images for this study share the following properties. Each contains corners of scale $S(Q_j) \in L$, where

$$L = \{15, 30, 45, 60, 75\},$$

with the length $\ell \in L$ expressed in image pixels. Further the angle of each corner satisfies $\alpha(Q_j) \in A$, where

$$A = \{45, 90, 135\},$$

with the angle $\alpha \in A$ expressed in degrees.

We defined each shape Q, so that for each corner angle $\alpha \in A$, and each corner scale $\ell \in L$, there would be at least one corner $Q_j \in Q$ with $\alpha(Q_j) = \alpha$ and $S(Q_j) = \ell$. Because the number of angles is $n(A) = 3$ and the number of corner scales is $n(L) = 5$, this requires the number of points on test shape $Q = \{q_j\}_{j=1}^{m}$ to satisfy $m \geq 15$. Meanwhile, the number of points on each object shape must be kept at a minimum to prevent intrasession fatigue [GS66]. To satisfy both criteria, we defined each of the shapes in Figure 2.2.

Test Images

We created a variety of images for each shape in Figure 2.2. Specifically, 20 images were created in which the object, background, and boundary were characterized according to Table 2.1. The images have been categorized in one of four groups, namely *ramp*, *roof*, *disturbed*, and *texture*, depending on the object and physical imaging models used

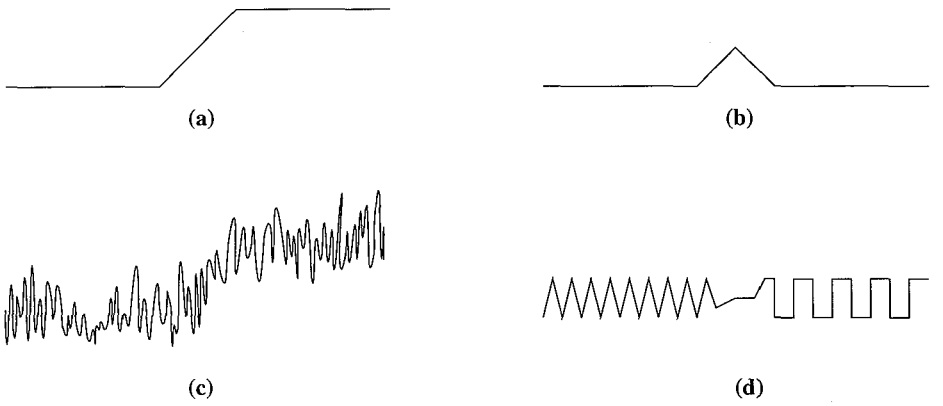


Figure 2.3: Image Models: a) ramp, b) roof, c) disturbed ramp, d) texture

to create them. Each of these groups are described in detail below, and models are shown in Figure 2.3. To prevent learning effects, the images were presented to the users in the order: a) texture, b) disturbed, and c) ramp and roof.

Let f_0 denote the value of the image function in the object region, f_1 the value in the background region, and f_b the value in the boundary region. In describing the image functions, we make use of the *signal* magnitude, as defined by

$$s(f) = |\max\{< f_0 >, < f_1 >, < f_b >\} - \min\{< f_0 >, < f_1 >, < f_b >\}|. \quad (2.9)$$

Thus, the signal is the difference in the average value of the image function in the object and background regions. For some images (namely roofs), it will be the difference in average value in the boundary and background regions.

Ramp Images An $X \times Y$ image, in which the image function is similar to that which would have occurred had the object been imaged with a camera, is generated as follows. We create the shape in an $8X \times 8Y$ image, the resulting image is smoothed with a two dimensional Gaussian filter, with standard deviation $\sigma = 8$. The test image is then obtained by subsampling the smoothed image to obtain an $X \times Y$ image with realistic edges. The resulting images are those displayed in Figure 2.2. For each shape, we created both a bright object ($f_0 = 150$) on a dark background ($f_1 = 50$) as well as a dark object on a bright background, resulting in the four images in the *ramp* category in Table 2.1. The signal level for these images is $s(f) = |f_1 - f_0| = 100$. Simple objects obtained with a perfect noise free camera would fall in this category, and would have boundary functions f_b similar to those depicted in Figure 2.3a.

Roof Images The roof images are created as the ramp images, but in this case, we have $f_0 = f_1$ and f_b, the boundary function is a roof like peak, with its maximum

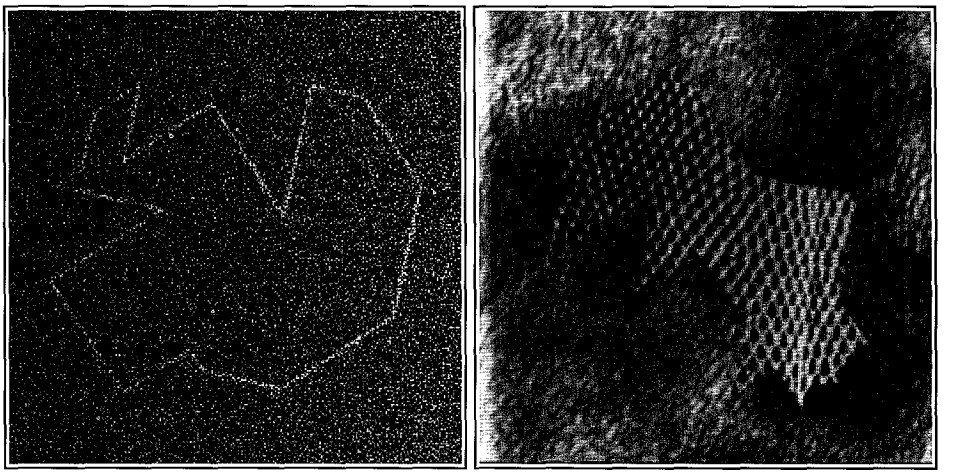


Figure 2.4: On the left, a bright roof boundary disturbed by noise and shading. On the right, an image with different Brodatz textures in the object and background regions.

(or minimum) between the object and background regions. The one dimensional roof boundary model is depicted in Figure 2.3b, and an example of a roof boundary (disturbed by noise and shading) is shown in Figure 2.4. For each shape the background function $f_1 \in \{50, 150\}$, was used to create the roof images. The average value along the peak of the boundary is $< f_b >= 90$ for a dark roof on a bright background, and $< f_b >= 107$ for a bright roof on a dark background. We thus compute the signal for the roof images as the difference between the average background value and the average roof value $s(f) = | < f_1 > - < f_b > | \approx 58.5$. The values are close to this for all four roof images which make up the *roof* image category in Table 2.1.

Disturbed Images The third group of images was generated by adding shading and noise to each of the images in the roof and ramp groups. Before sampling the image, we added a linear shading function f_s to each of the images. The function increased linearly as a function of x starting on the left with $f_s(0, y) = 0$, and ending on the right with the signal value $f_s(X, y) = s(f)$. Independent Gaussian noise resulting in a signal to noise ratio $SNR = 2$ was then added to each of the images, where

$$SNR = \frac{s(f)}{\sigma_{noise}}$$

and σ_{noise} is the standard deviation of the Gaussian noise. The eight images resulting from these modifications make up the *disturbed* category of images in Table 2.1, an example of which is shown in Figure 2.4.

Test Images					
Image Category	Shape	Object Function	Background Function	Boundary Function	Signal $s(f)$
Ramp	1,2	Bright: f_0=150	Dark: f_1=50	Blurred	100
(4 images)	1,2	Dark: f_0=50	Bright: f_1=150	Blurred	100
Roof	1,2	Dark: f_0=50	Dark: f_1=50	Bright: $f_b \approx 107$	57
(4 images)	1,2	Bright: f_0=150	Bright: f_1=150	Dark: $f_b \approx 90$	60
Disturbed	Ramp and roof images disturbed with additive linear shading in				
(8 images)	the range $[0, s(f)]$ and additive Gaussian noise giving $SNR = 2$.				
Texture	1	D28	D6	Blurred	0
(4 images)	1	D36	D93	Blurred	0
	2	D105	D57	Blurred	0
	2	D19	D38	Blurred	0

Table 2.1: The object, background and boundary functions which characterize the 20 test images. The Dn labels are the identifiers for the Brodatz textures in [Bro66].

Texture Images To create the texture images we made use of images[2] of the textures in the Brodatz album [Bro66]. The size of the images used was 512×512. To assure the difference between the foreground texture function f_0 and the background f_1 was strictly due to a difference in texture and not due to a difference in average grey value, we normalized the images used so that $< f_0 >=< f_1 >= 128$, where the average was taken over a large region encompassing the object[3]. In order to simulate the camera model along the boundary without averaging out the texture pattern, we created the texture images by filling the object region in a $2X \times 2Y$ image. Subsequently we blurred with a two dimensional Gaussian ($\sigma = 2$), and subsampled to obtain the $X \times Y$ image presented to the user. We made two images in this fashion for each of the two shapes resulting in the images in the *texture* category in Table 2.1, an example of which is shown in Figure 2.4. The textures selected for the experiment were required to have small texture elements in relation to the smallest corner so that all corners would in principle be visible.

2.4 Results

In this section, we examine the data collected from the user experiments with respect to the issues posed in Section 2.2.3. Our analysis is geared toward the development of a user error model for the task of corner specification. Before the errors can be measured, the points in each polygon obtained from a user must be matched with the points in the polygon used to generate the test image. Because users sometimes skip

[2]We used images from the set scanned at MIT

[3]There are some artifacts near the edges of the images we used which we did not want to include in our computations, as they did not effect the region in which the user was to draw.

corner points on the polygon, and insert points between two corners on the object, this turns out to be a nontrivial step in the user error evaluation. To address this problem, we developed the matching algorithm in Chapter 3. In the remainder of this chapter, we assume the user and polygon point pairs evaluated, are correctly matched.

The four errors $\{\delta_{ij}, \delta\alpha_{ij}, \delta\beta_{ij}, \delta S_{ij}\}$, defined in Section 2.2.2, were measured for each of the user/model corner point pairs. Before evaluating the degree and type of errors made in defining the corners, we must determine whether the data collected for different types of images should be evaluated as a set. By grouping the data appropriately we can address the influence of variation in B (image function), C (disturbance), and E (shape) on user behavior (see Section 2.2.3). In Section 2.4.1, the errors made on various groups of images will be compared using the Kolmogorov-Smirnov test [vM64].

In Section 2.4.2, for each of the four errors, we summarize each data set in terms of its mean value and standard deviation, for the four errors. The results are used to model the user error distributions in Section 2.4.3. Because user performance on the texture images differed dramatically from the performance on the remainder of the images, the results for these images are considered separately in Section 2.4.4.

Outlyers In specifying the polygonal boundary paths, each of the users B, D, and F twice inserted an extra point. These points were not matched with points on the true polygons, and do not contribute to the error measures. However, they introduce a severe error in corner scale δS_{ij}, as the length of the line to the neighboring corner is much shorter than it should be. Unless these points are removed prior to evaluation, the error scale data, and sometimes angle data, cannot be correctly evaluated, as these points have a strong influence both on the mean and standard deviation. The outlyers, seven in total, were removed prior to the evaluation presented in the following sections.

Note that in practice, the presence of such outlyers indicates the necessity of good correction facilities for this form of boundary specification. They not only introduce errors in the neighborhood of the extra point, but introduce errors in the modeling of the immediate neighboring points on the polygon.

2.4.1 Comparison of Error Distributions

Before examining the errors made in corner specification, we first need to establish which of the data can be grouped and viewed as a consistent set to be evaluated. Because the variation in the distance and scale error distribution parameters among users is significant, the error levels for each user must be examined separately. Moreover, angle errors show a clear dependence on corner scale, which means the errors made at each scale should be viewed separately. Both of these points will be established in Section 2.4.2, but first we must establish which errors can be viewed as a group for individual users at each scale.

Data Groups

For each user, we want to know if the distance errors made on ramp and roof images are drawn from the same distribution. In the experiments, there were eight ramp and

Number of corners						
	Corner Scale					
Shape	15	30	45	60	75	all
pol1	5	3	3	3	3	17
pol2	4	4	3	3	3	17
both	9	7	6	6	6	34

Table 2.2: The number of corners in each polygon at the given scale.

eight roof images used. Of the eight ramp images, four were of one shape shown in Figure 2.2, and four of the other. Four contained added disturbance, and four did not. Four were of a bright object on a dark object, and four contained the reverse. The same variations held for the roof images. To see if the error levels are effected by whether an image contains a ramp or roof boundary, we pool the eight distinct ramp images and compare each error set with that for the group of eight roof images to see if this aspect (ramp versus roof) of the image function is a significant factor in the user error level.

Likewise, to compare the influence of disturbance on user error levels, we pool the user errors at each scale for all eight images with added noise and shading, and compare them to the errors made for the group of eight images without disturbance. Each of these groups contain eight distinct images, namely, ramp and roof images with dark and bright objects of each form. The error sets for dark and bright images are similarly pooled and compared, and finally the error sets for all images containing one shape is compared with those for all images containing the other.

The left polygon in Figure 2.2 is referred to as "pol1" and the right polygon is referred to as "pol2". Table 2.2 shows the number of points of a given scale (see Definition 2.1) present in each polygon. Because there are four ramp images in the "pol1" shape and four in the "pol2" shape, and four of each shape in the eight roof images, we compare $4 \times 5 + 4 \times 4 = 36$ errors made at corner scale 15 on the ramp images with the 36 errors made on roof images at corners of the same scale. All error sets compared below are constructed similarly.

Kolmogorov-Smirnov Statistics

For a given user specifying points on a set of images, we can view the errors measured with each of the functions in Section 2.2.2, at each corner scale, as a set $E = \{e_1, e_2, \ldots, e_N\}$. If we define the subset

$$E(x) = \{y \in E : y \leq x\},$$

then the distribution function of the error set E is given by

$$S_E(x) = \frac{n(E(x))}{n(E)}, \tag{2.10}$$

where $n(X)$ denotes the number of elements in a set X. To compare the performance of a user on two nonintersecting error sets E_1 and E_2, we can compute the Kolmogorov-Smirnov statistic [Smi39], defined by

$$D = D(E_1, E_2) = \max_{-\infty<x<\infty} |S_{E_1}(x) - S_{E_2}(x)|. \tag{2.11}$$

For significantly large values of D, we can assume the two error sets are from distinct distributions. In comparing two data sets, the significance of D is given by

$$\lambda = \sqrt{\frac{N_1 N_2}{N_1 + N_2}} D, \tag{2.12}$$

where $N_i = n(E_i)$. As shown in [vM64], the probability that the null hypothesis "E_1 and E_2 are drawn from the same distribution" is not false, is given by

$$p(\lambda) = 2 \sum_{k=1}^{\infty} (-1)^{k-1} e^{2k^2\lambda^2}, \tag{2.13}$$

where λ is defined in Equation 2.12.

Each of the errors δ_{ij}, $\delta\alpha_{ij}$, $\delta\beta_{ij}$ and δS_{ij}, for a given user were grouped according to corner scale. For each of the error set groupings described above, we computed the Kolmogorov-Smirnov statistic D as well as the probability defined in Equation 2.13 for the pair of error distributions. For example, we compare errors in E_1 and E_2 where E_1 = { user A, scale = 15, δ_{ij}, ramp images } and E_2 = { user A, scale = 15, δ_{ij}, roof images }. The Kolmogorov-Smirnov D statistic for this comparison is shown in the δ_{ij} column in the row marked 15 in Table 2.3. Lower in the table, the probability of the significance of D is given. We present the statistic D along with its associated probability for user "A" for the four image groupings to be compared in Tables 2.3, 2.4, 2.5 and 2.6 below, to illustrate the set of statistics gathered for each user.

Note that for nearly all error measures, at all scales, the difference in error distributions on the groups compared is not significant. That is, in almost all cases, $p(\lambda) \gg 0.05$. This is, however, not always the case. We therefore summarize the results for all users, to see if there are patterns which should influence the data groups examined.

Consider the set of error set pairs evaluated for ramp versus roof images (Table 2.3). Suppose we call it Ξ. There are four error set pairs (1 for each error type), for each of the five corner scales, and there are six users. So there are $n(\Xi) = 120$ error set pairs compared in the ramp versus roof evaluation. Likewise there are 120 error set pairs compared in the disturbed versus clean, dark versus bright, and "pol1" versus "pol2" evaluations. Let Ξ_x denote the number of pairs in a given group Ξ of error set pairs

K-S statistics for E_1 = **ramp** versus E_2 = **roof**						
Corner Scale	Error Sets for user "A"				Size	
	δ_{ij}	$\delta\alpha_{ij}$	$\delta\beta_{ij}$	δS_{ij}	N_1	N_2
$D(E_1, E_2)$						
15	0.1389	0.3056	0.2778	0.2778	36	36
30	0.2857	0.2143	0.3214	0.2143	28	28
45	0.1667	0.2500	0.2917	0.3333	24	24
60	0.2917	0.3333	0.2083	0.2500	24	24
75	0.2083	0.2083	0.0833	0.1667	24	24
$p(\lambda)$						
15	0.8782	0.0694	0.1243	0.1243	36	36
30	0.2032	0.5412	0.1108	0.5412	28	28
45	0.8928	0.4413	0.2591	0.1389	24	24
60	0.2591	0.1389	0.6749	0.4413	24	24
75	0.6749	0.6749	1.0000	0.8928	24	24

Table 2.3: The Kolmogorov-Smirnov statistics for comparing error sets on *ramp* and *roof* images. There were eight *ramp* and eight *roof* images used in the experiments.

K-S statistics for E_1 = **disturbed** versus E_2 = **clean**						
Corner Scale	Error Sets for user "A"				Size	
	δ_{ij}	$\delta\alpha_{ij}$	$\delta\beta_{ij}$	δS_{ij}	N_1	N_2
$D(E_1, E_2)$						
15	0.1944	0.1389	0.1389	0.3056	36	36
30	0.1786	0.2143	0.1429	0.2500	28	28
45	0.1667	0.2083	0.1250	0.2917	24	24
60	0.0833	0.1250	0.2083	0.0833	24	24
75	0.2500	0.2500	0.2500	0.4167	24	24
$p(\lambda)$						
15	0.5041	0.8782	0.8782	0.0694	36	36
30	0.7634	0.5412	0.9375	0.3457	28	28
45	0.8928	0.6749	0.9920	0.2591	24	24
60	1.0000	0.9920	0.6749	1.0000	24	24
75	0.4413	0.4413	0.4413	0.0310	24	24

Table 2.4: The Kolmogorov-Smirnov statistics for comparing error sets on *disturbed* and *clean* images. The *disturbed* images are the eight with added noise and shading, and the *clean* images are those without added disturbance.

K-S statistics for E_1 = **dark** versus E_2 = **bright**						
Corner	Error Sets for user "A"				Size	
Scale	δ_{ij}	$\delta\alpha_{ij}$	$\delta\beta_{ij}$	δS_{ij}	N_1	N_2
$D(E_1, E_2)$						
15	0.1944	0.1667	0.2778	0.2500	36	36
30	0.1429	0.2143	0.1786	0.4286	28	28
45	0.1250	0.2500	0.2917	0.1667	24	24
60	0.2917	0.1250	0.2083	0.2500	24	24
75	0.2500	0.2083	0.1667	0.1667	24	24
$p(\lambda)$						
15	0.5041	0.6994	0.1243	0.2106	36	36
30	0.9375	0.5412	0.7634	0.0117	28	28
45	0.9920	0.4413	0.2591	0.8928	24	24
60	0.2591	0.9920	0.6749	0.4413	24	24
75	0.4413	0.6749	0.8928	0.8928	24	24

Table 2.5: The Kolmogorov-Smirnov statistics for sets from *dark* and *bright* images. The four ramp images with a dark object, and the four roof images with a dark border are the *dark* images. The *bright* images are the eight for which the reverse holds.

K-S statistics for E_1 = **pol1** versus E_2 = **pol2**						
Corner	Error Sets for user "A"				Size	
Scale	δ_{ij}	$\delta\alpha_{ij}$	$\delta\beta_{ij}$	δS_{ij}	N_1	N_2
$D(E_1, E_2)$						
15	0.3438	0.2188	0.1938	0.2125	40	32
30	0.2500	0.2188	0.2396	0.2083	24	32
45	0.3750	0.3333	0.2500	0.3750	24	24
60	0.1667	0.2500	0.2917	0.3333	24	24
75	0.2917	0.2500	0.2500	0.4167	24	24
$p(\lambda)$						
15	0.0299	0.3626	0.5169	0.3983	40	32
30	0.3581	0.5278	0.4106	0.5911	24	32
45	0.0684	0.1389	0.4413	0.0684	24	24
60	0.8928	0.4413	0.2591	0.1389	24	24
75	0.2591	0.4413	0.4413	0.0310	24	24

Table 2.6: The Kolmogorov-Smirnov statistics for error sets from *pol1* and *pol2* images. The eight ramp and roof images of the first shape (Figure 2.2 left) make up *pol1*, and the eight of the second shape make up *pol2* (Figure 2.2 right).

Summary of K-S statistics for all users					
Image Sets (Ξ)	$n(\Xi)$	$n(\Xi_{0.05})$	$\%(\Xi_{0.05})$	$n(\Xi_{0.01})$	$\%(\Xi_{0.01})$
ramp/roof	120	7	5.83	3	2.50
disturbed/clean	120	11	9.16	2	1.67
dark/bright	120	4	3.33	0	0.00
pol1/pol2	120	13	10.83	5	4.17
Measure (Ξ)	$n(\Xi)$	$n(\Xi_{0.05})$	$\%(\Xi_{0.05})$	$n(\Xi_{0.01})$	$\%(\Xi_{0.01})$
δ_{ij}	120	6	5.00	1	0.83
$\delta\alpha_{ij}$	120	5	4.17	1	0.83
$\delta\beta_{ij}$	120	8	6.67	2	1.67
δS_{ij}	120	16	13.33	6	5.00
all	480	35	7.29	10	2.08

Table 2.7: A summary of the significant Kolmogorov-Smirnov probabilities on various groupings (Ξ) of the error set pairs. For each set Ξ of error set pairs, the number and percentage of error set pairs in Ξ for which $p(\lambda) < x$ is given, for both $x = 0.05$ and $x = 0.01$. See Equations 2.14 and 2.15.

which are significant in the sense that $p(\lambda) \leq x$, so

$$\Xi_x = \{(E_1, E_2) \in \Xi : p(\lambda) \leq x\}, \tag{2.14}$$

where λ is defined in Equation 2.12. The number of error set pairs in Ξ for which $p(\lambda) \leq x$ is given by $n(\Xi_x)$, and the percentage of Ξ with $p(\lambda) \leq x$ is given by

$$\%(\Xi_x) = (100 \times n(\Xi_x))/n(\Xi). \tag{2.15}$$

In Table 2.7, we show the number and percentage of pairs for which $p(\lambda) < 0.05$, and for which $p(\lambda) < 0.01$ when the statistics are grouped in various manners.

Evaluation

Suppose we require $p(\lambda) < x$ before we reject the null hypothesis that two error sets come from the same distribution. Consider the first three error measure sets $\{\delta_{ij}\}$, $\{\delta\alpha_{ij}\}$, and $\{\delta\beta_{ij}\}$ in Table 2.7. We see the percentage of error set pairs in Ξ for which $p(\lambda) < 0.01$ is very low ($\%(\Xi_{0.01}) \ll 5\%$), and the percentage $\%(\Xi_{0.05}) \approx 5\%$, for which $p(\lambda) < 0.05$ is insufficient to reject the null hypothesis.

In contrast, we see that the set $\Xi = \{\delta S_{ij}\}$ of error set pairs, contains a significant number of pairs which, based on the Kolmogorov-Smirnov test, appear to come from different distributions. At the $x = 0.05$ significance level, we have $\%(\Xi_{0.0.5}) = 13.33\%$ and at the $x = 0.01$ significance level, we have $\%(\Xi_{0.01}) = 5\%$. We therefore inspect

Summary of K-S statistics for δS_{ij} - all users					
Image Sets (Ξ)	$n(\Xi)$	$n(\Xi_{0.05})$	$\%(\Xi_{0.05})$	$n(\Xi_{0.01})$	$\%(\Xi_{0.01})$
ramp/roof	30	2	6.67	1	3.33
disturbed/clean	30	7	23.33	1	3.33
dark/bright	30	1	3.33	0	0.00
pol1/pol2	30	6	20.00	4	13.33
all	120	16	13.33	6	5.00

Table 2.8: A summary of the Kolmogorov-Smirnov probabilities for comparing corner scale error sets $\{\delta S_{ij}\}$. For each set Ξ of error set pairs, the number and percentage of error set pairs in Ξ for which $p(\lambda) < x$ is given, for both $x = 0.05$ and $x = 0.01$.

the contributions to the set $\Xi_{0.01}$, and $\Xi_{0.05}$ for $\Xi = \delta S_{ij}$ in Table 2.8, to see which image function comparisons, contribute to the error set pairs in $\Xi_{0.05}$ and $\Xi_{0.01}$.

Inspection of Table 2.8 shows that the majority of the error set pairs which are significantly different at the $x = 0.05$ are found when comparing the sets with and without disturbance, and when comparing the error sets on the two polygons. Note that the former (disturbed/clean) does not contribute significantly to $\Xi_{0.01}$. When comparing the scale error set pairs on the two polygons (pol1/pol2), however, we find 13.33% of the comparisons fall in $\Xi_{0.01}$. We therefore reject the hypothesis that the corner scale error sets are drawn from the same distributions for the two polygons. In the following section, the corner scale errors on each polygon will be evaluated separately.

Because the error set pairs on corner scale $\Xi = \{\delta S_{ij}\}$, account for the majority of error sets which appear to be significantly different in Table 2.7, we evaluate the performance on all other error measures as a single set.

Information With respect to the questions posed in Section 2.2.3, from the above analysis, we can conclude the following.

B) User performance in corner specification is not influenced by whether the image function which determines the nature of the object boundary is best described in terms of a ramp or a roof model. For ramp model images, it does not matter if the object is bright or dark, in comparison with the background. Likewise, whether a roof is dark or bright as compared with the background does not influence the errors in corner specification.

C) With the possible exception of the corner scale error, there is no significant change in user error levels when noise and shading are added to an image.

E) The degree of corner scale error depends on object shape. Other errors are uninfluenced by shape.

2.4.2 User Error Summary

In this section, we present statistics on user errors as a function of the corner scale $S(Q_j)$, defined in Definition 2.1. The mean and standard deviation of each of the measures defined in Section 2.2.2 are presented for each user. In Figures 2.5, 2.6, and 2.7, the summary of the statistics for distance, angle, and orientation errors for the sixteen nontexture images in our experiments is shown. Due to the results in the previous section, the corner scale error statistics are evaluated separately for the two polygons in Figures 2.8 and 2.9.

Note that for the measures of distance error δ_{ij} and orientation error $\delta\beta_{ij}$, the mean corresponds with the average user error magnitude, whereas for corner angle $\delta\alpha_{ij}$ and scale δS_{ij}, the mean gives the bias.

Evaluation From Figure 2.5, we see that user errors in distance are fairly consistent for various scales, and the deviations are small. Further, with the exception of users E and F, the error magnitude for the various users is distinct.

In Figure 2.6, the average difference in corner angle ($\delta\alpha_{ij} = \alpha(P_i) - \alpha(Q_j)$), for each of the users is shown. With the exception of A and C, the users tend to slightly overestimate the corner angle. Both in Figure 2.6 and in Figure 2.7, we see the magnitude and variation of corner angle errors decrease as a function of corner scale for all users. This is logical, considering the error in distance does not depend on corner scale, and can be understood as follows.

Let $\mathbf{v}$ be a vector in $\mathbf{R}^2$, and let $\widehat{\mathbf{v}} = a\mathbf{v}$ for some $a \in (0,1)$. Let $\mathbf{e} \in \mathbf{R}^2$ be a constant vector. Consider the angles of $\mathbf{v}$ and $\mathbf{v}+\mathbf{e}$ given by

$$\theta = \cos^{-1}\left(\frac{\mathbf{v}\cdot(1,0)}{\|\mathbf{v}\|}\right) \quad \text{and} \quad \gamma = \cos^{-1}\left(\frac{(\mathbf{v}+\mathbf{e})\cdot(1,0)}{\|\mathbf{v}+\mathbf{e}\|}\right)$$

If we denote the angles of $\widehat{\mathbf{v}}$ and $\widehat{\mathbf{v}}+\mathbf{e}$ with $\widehat{\theta}$ and $\widehat{\gamma}$ respectively, then

$$|\theta - \gamma| \leq |\widehat{\theta} - \widehat{\gamma}|,$$

with equality holding only when $\mathbf{e} = c\mathbf{v}$ for some $c \in \mathbf{R}$. This is easy to prove using arguments in [MT81]. Because both corner angle and orientation depend on the angles of the lines which define the corners, the errors decrease for increasing corner scales, as shown in Figure 2.10.

In Figures 2.8 and 2.9, we show the bias and standard deviations of the corner scale error for the two polygons used in the experiments. As was expected from the conclusions of the evaluation in the previous section, the user patterns for scale error differ. For corners with scale $S(Q_j) > 15$, however, all users other than "A" share the tendency to underestimate the corner scale. Interestingly, the scale error patterns for all users are quite similar on the second polygon (Figure 2.2 right), and somewhat similar on the first polygon (Figure 2.2 left). Finally, the magnitude of the corner scale error is small for very small corners $S(Q_j) = 15$.

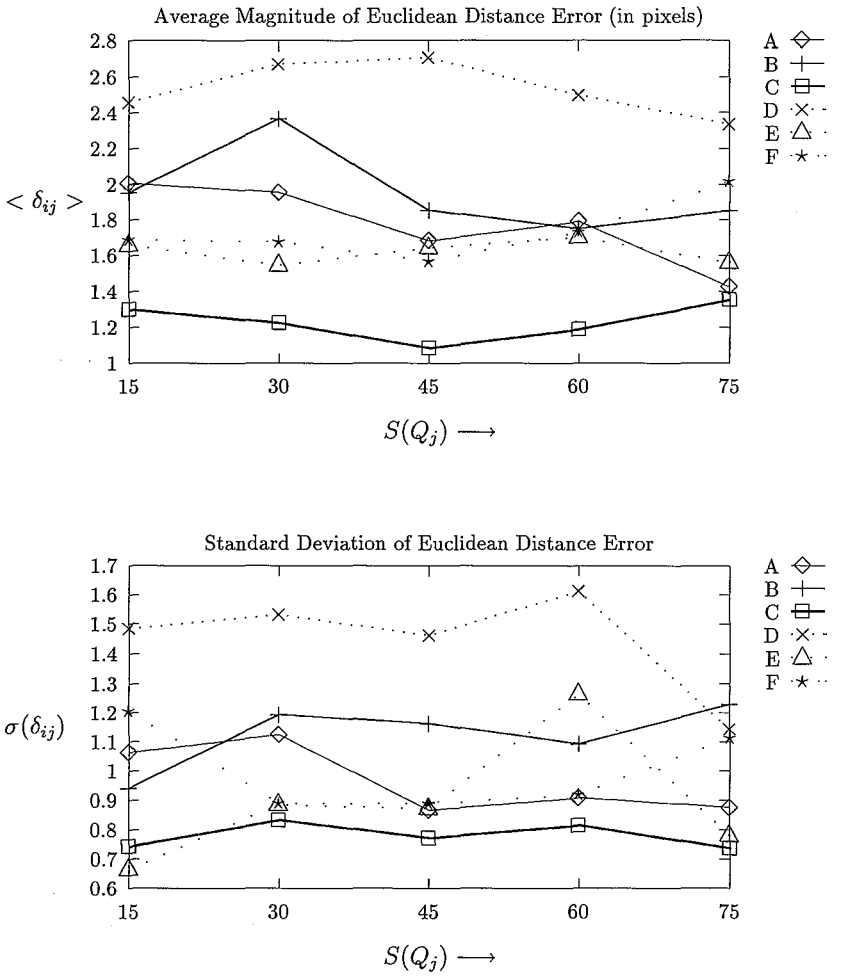


Figure 2.5: *Above*: the average magnitude of the error in Euclidean distance $\delta_{ij} = \|p_i - q_j\|$ made by each of six different users as a function of corner scale. *Below*: the standard deviation $\sigma(\delta_{ij})$ of the distance errors for each of the users.

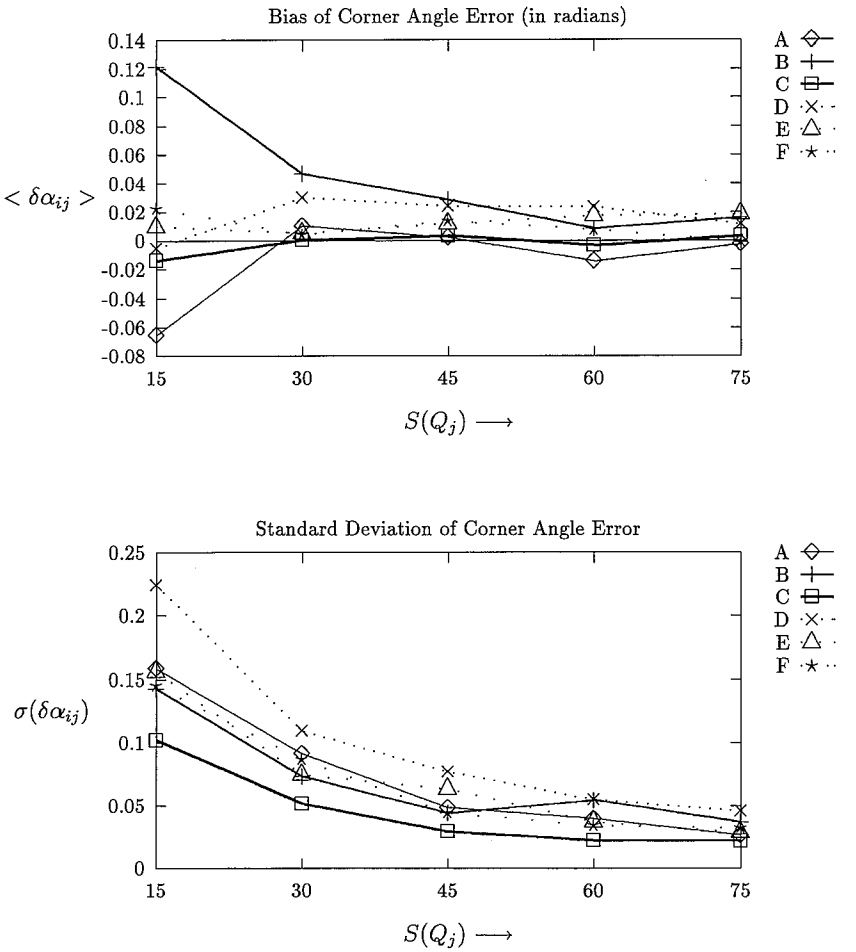


Figure 2.6: *Above*: for each of the six users, the average difference in corner angle size $\delta\alpha_{ij} = \alpha(P_i) - \alpha(Q_j)$ is plotted as a function of corner scale. *Below*: the standard deviation $\sigma(\delta\alpha_{ij})$ of the angle difference for each of the users.

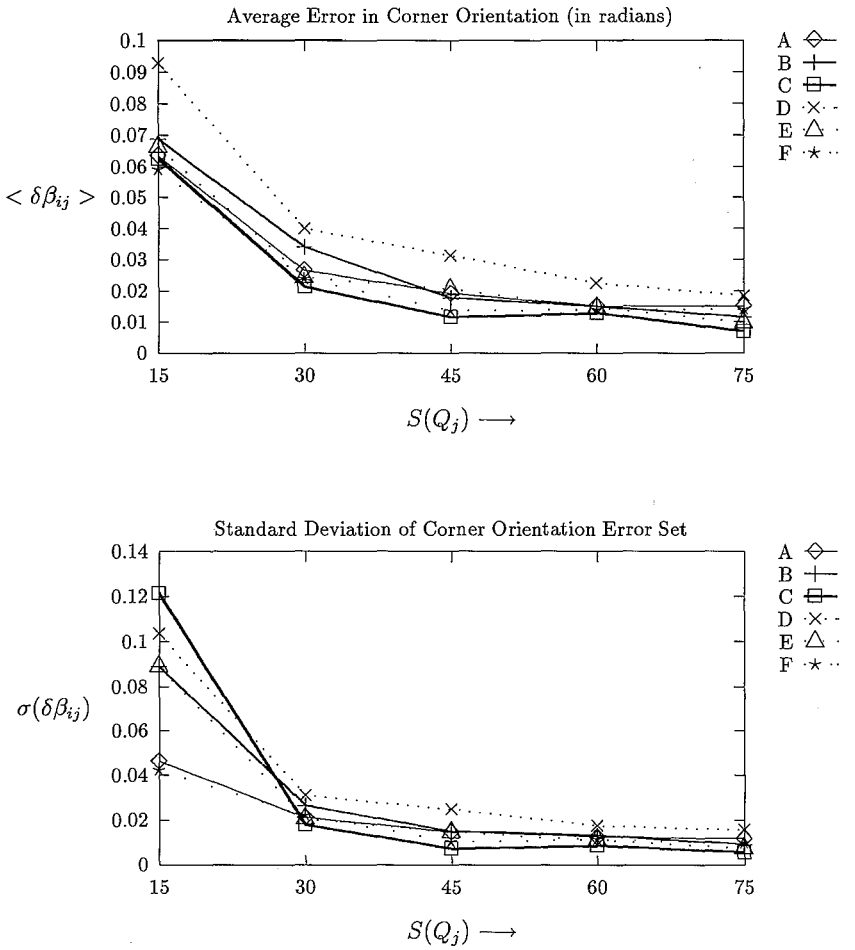


Figure 2.7: *Above*: the average error in corner orientation $\delta\beta_{ij}$ (defined in Equation 2.7) made by each of the six users, plotted as a function of corner scale. *Below*: the standard deviation $\sigma(\delta\beta_{ij})$ for each of the users.

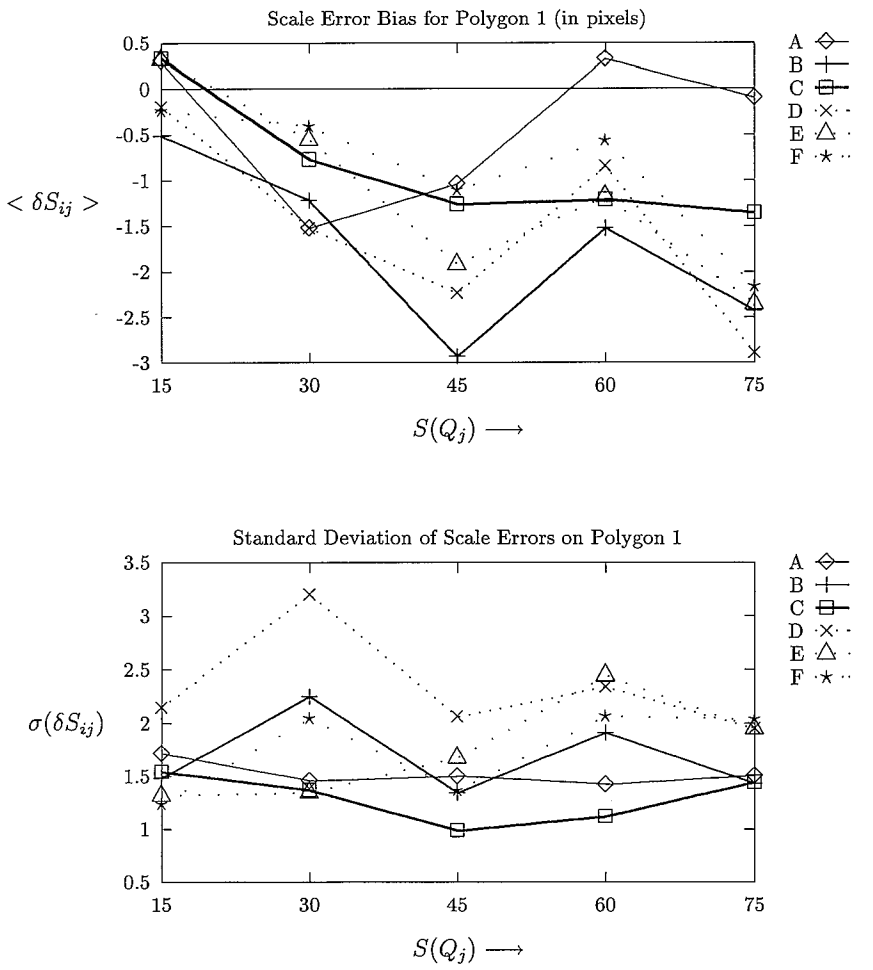


Figure 2.8: *Above:* for each of the users, the average difference in corner scale $\delta S_{ij} = S(P_i) - S(Q_j)$, plotted as a function of the true corner scale $S(Q_j)$ for the left shape (pol1) in Figure 2.2. *Below*: the standard deviation $\sigma(\delta S_{ij})$ of the scale difference on this polygon (pol1) for each of the users.

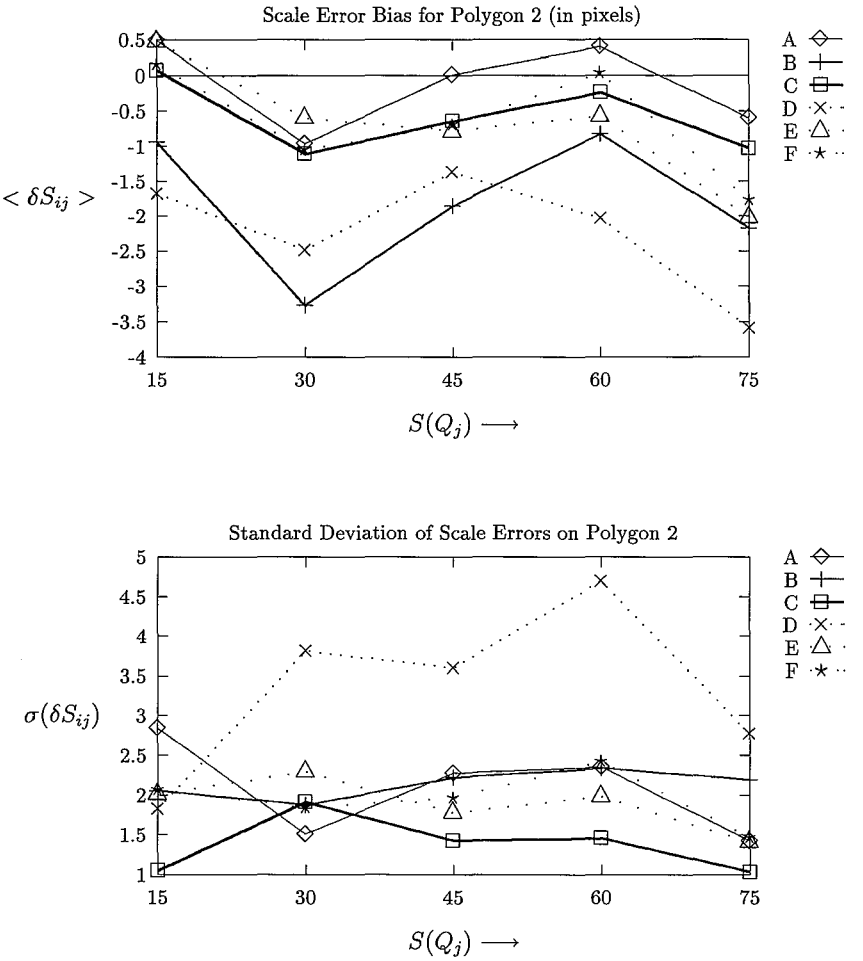


Figure 2.9: *Above:* for each of the users, the average difference in corner scale $\delta S_{ij} = S(P_i) - S(Q_j)$, plotted as a function of the true corner scale $S(Q_j)$ for the shape (pol2) on the right in Figure 2.2. *Below*: the standard deviation $\sigma(\delta S_{ij})$ of the scale difference on this polygon (pol2) for each of the users.

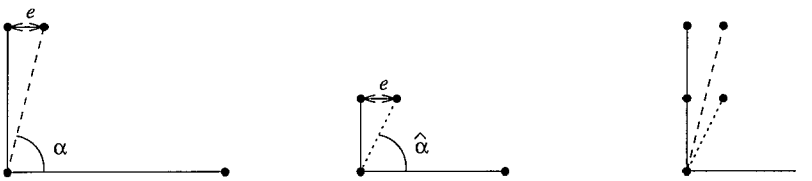


Figure 2.10: The same distance error results in larger angle errors for a corner of smaller scale.

Information With respect to the issues raised in Section 2.2.3, we can conclude the following based on the data presented in Figures 2.5, 2.6, 2.7, 2.8, and 2.9.

A) Errors in distance are unrelated to corner scale. The magnitude of the corner scale error, however, are smaller for corners of small scale ($S(Q_j) = 15$). Meanwhile, angle related errors increase in magnitude for corners of very small scale.

D) For all the errors measured, the performance of the users as measured by the magnitude of the means depends on the individual user. The error patterns for the users are, however, very similar.

2.4.3 Modeling User Error Sets

Suppose for each of the users, and each error measured, we investigate the error set at each scale. Then we can extract some set of parameters which describe the error set, and may help us to find a statistical model which it fits.

In considering the corner angle error ($\delta\alpha_{ij}$), at each scale, we may expect the user error to behave as a Gaussian distribution, centered about $\mu =< \delta\alpha_{ij} >$, with standard deviation $\sigma = \sigma(\delta\alpha_{ij})$. Given the parameter pair $\{\mu, \sigma\}$, the Gaussian distribution is defined by

$$F(x) = F(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \frac{1}{\sigma} e^{\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dt. \tag{2.16}$$

We might also suppose the corner scale error (δS_{ij}) sets to fit a Gaussian distribution, with $\mu =< \delta S_{ij} >$ and $\sigma = \sigma(\delta S_{ij})$.

We can evaluate the validity of these hypotheses, using the Kolmogorov statistic for comparing a single data set with a known distribution. Suppose $F(x)$ describes the distribution we expect the data to fit. And suppose $S_E(x)$ is the distribution function of the error set, as defined in Equation 2.10. Let

$$D = \max_{-\infty<x<\infty} |S_E(x) - F(x)|. \tag{2.17}$$

For comparing a single data set with a known distribution, the significance of D is given by

$$\lambda = \sqrt{N}D, \tag{2.18}$$

where $N = n(E)$. The probability that the null hypothesis "E is drawn from the distribution described by $F(x)$" is not false is given by $p(\lambda)$, as defined in Equation 2.13 [vM64].

Corner Angle $\delta\alpha_{ij}$: For each of the six users, there were five scales for which the corner angle error sets were generated. For each set, we computed D and $p(\lambda)$ for comparing the error set E with the Gaussian distribution, using the parameter pair $\{\mu, \sigma\}$ drawn from the error set. In every case, $p(\lambda) > 0.05$. We may therefore assume that user errors in corner angle are well modeled with a Gaussian distribution.

Corner Scale δS_{ij}: The corner scale error sets were handled in the same fashion, but in this case, there were three different sets of errors generated for each user, one for all images, and one for each of the two polygons. In each case, there were 30 error sets compared with the Gaussian distribution. In each case, there was exactly one set for which $p(\lambda) < 0.05$. Since this is insufficient to be considered statistically significant, we may assume the sets are well modeled by a Gaussian distribution for which the parameter pair $\{\mu, \sigma\}$ corresponds with the mean and standard deviation of the set at hand. In light of the results in Section 2.4.1, it is interesting (and useful), that this holds even when the data for both polygons is pooled.

Distance δ_{ij} and Orientation $\delta\beta_{ij}$ Although we were unable to find distributions for the errors in distance and orientation, both data sets were well behaved.

In particular, if μ and σ represent the mean and standard deviation for the distance error made by a given user at a given scale, then the actual error made by that user at that scale satisfies $\delta_{ij} \leq \mu + 2\sigma$ for at least 93% of the distance errors δ_{ij} made by the same user at that scale. This was true for every user at every scale. Moreover, in many cases, the distribution of the distance squared error fit that of a χ^2 distribution.

Likewise, if μ and σ represent the parameters of the orientation distribution for a given user at a given scale, then $\delta\beta_{ij} \leq \mu + 2\sigma$, for more than 91% of the errors $\delta\beta_{ij}$, made by the same user at that scale. This held for every user tested.

Although we did not find a mathematical model which could be used to describe the distributions of these errors in all cases, the above findings allow us to derive a model for the expected worst case for these two errors.

2.4.4 Performance on Texture

The polygons traced by users in texture images have been excluded from the analysis to this point. Users showed large variation in the specification of the object boundaries in these images. These were the first images presented to the users, so the object shapes were not yet familiar. In some cases, a user specified the boundary with far more points than in Q, while others skipped many points in Q. Even after the points in the user polygon P were matched with those in Q, the missing and extra points in P introduced spurious large errors for the angle and scale for other corners. Because each of the texture images was created from a different pair of Brodatz textures, the

performance on the different images varied widely for each user. Unlike the other images used in the experiments, the differences in the Brodatz texture images is not easily quantified [Har78]. Based on the experiments performed here, it is therefore not possible to quantify a user model for performance on texture images.

2.5 The User Error Model

Based on the results in the previous section, we are now able to extract a user error model for the specification of polygonal objects in grey level images. Given a point set $P = \{p_i\}_{i=1}^{n}$ specified by a user, the user error model is characterized by the following properties.

Maximum Euclidean Distance Let $\mu(\delta_{ij}, S, U)$ be the mean of the distance errors made by the user U for corners of scale S, and let $\sigma(\delta_{ij}, S, U)$ be the standard deviation of the same set. Inspection of the user error sets shows that for every user U and every scale S, the distance error δ_{ij} satisfies

$$\delta_{ij} < \mu(\delta_{ij}, S, U) + 2\sigma(\delta_{ij}, S, U),$$

for more than 93% of the measurements δ_{ij} made for the user U at scale S. Therefore, given a user point p_i specified by user U, there should be a corner point q_j on the image object boundary which satisfies $\|p_i - q_j\| \leq \varepsilon$, for

$$\varepsilon \geq \mu(\delta_{ij}, S, U) + 2\sigma(\delta_{ij}, S, U). \tag{2.19}$$

when the scale of the triplet P_i is close to S. Because the distance error did not show any dependence on scale, for each user U we can simply use the worst case value (for all scales S for user U) for the maximum user error ε.

Corner Angle Let $\mu(\delta\alpha_{ij}, S, U)$ be the mean bias of the corner angle for user U at scale S, and let $\sigma(\delta\alpha_{ij}, S, U)$ be the standard deviation of this angle error set. The corner angle errors for a given user a given scale were shown to fit a Gaussian distribution in Section 2.4.3, if the parameter pair $\{\mu = \mu(\delta\alpha_{ij}, S, U), \sigma = \sigma(\delta\alpha_{ij}, S, U)\}$, is used in Equation 2.16. Therefore, if we have a user defined triplet P_i which defines a corner at the point p_i, the probability is at least $p = 0.92$ that there is a corner in the image object defined by Q_j such that

$$\mu(\delta\alpha_{ij}, S, U) - 2\sigma(\delta\alpha_{ij}, S, U) \leq \delta\alpha_{ij} \leq \mu(\delta\alpha_{ij}, S, U) + 2\sigma(\delta\alpha_{ij}, S, U), \tag{2.20}$$

for the error $\delta\alpha_{ij} = \alpha(P_i) - \alpha(Q_j)$ generated by the user U of scale $S(P_i) \geq S$. The inequality is because the error level for all users decreases as a function of corner scale, and because users underestimate corner scale almost uniformly (see Figures 2.8 and 2.9). Therefore if the user U defines a corner P_i of scale $S(P_i)$ in a boundary sketch, there is a high probability ($p \geq 0.92$) of a corner Q_j on the object boundary

which satisfies

$$\alpha(P_i) - \mu(\delta\alpha_{ij}, S, U) - 2\sigma(\delta\alpha_{ij}, S, U) \leq \alpha(Q_j) \leq \alpha(P_i) - \mu(\delta\alpha_{ij}, S, U) + 2\sigma(\delta\alpha_{ij}, S, U). \tag{2.21}$$

Corner Orientation Let $\mu(\delta\beta_{ij}, S, U)$ be the mean error in orientation produced by the user U at scale S, and let $\sigma(\delta\beta_{ij}, S, U)$ be the standard deviation of the set. For the user error in orientation, the errors $\delta\beta_{ij}$ for each user at each scale satisfy

$$\delta\beta_{ij} < \mu(\delta\beta_{ij}, S, U) + 2\sigma(\delta\beta_{ij}, S, U),$$

for more than 91% of the errors $\delta\beta_{ij}$ generated for the user U at scale S. Thus, given a user defined corner P_i, with orientation $\beta(P_i)$, and scale $S(P_i)$, there should be a corner Q_j on the object boundary, the orientation $\beta(Q_j)$ of which satisfies

$$\beta(P_i) - \mu(\delta\beta_{ij}, S, U) - 2\sigma(\delta\beta_{ij}, S, U) \leq \beta(Q_j) \leq \beta(P_i) + \mu(\delta\beta_{ij}, S, U) + 2\sigma(\delta\beta_{ij}, S, U), \tag{2.22}$$

for $S \leq S(P_i)$. Again, the inequality holds because the orientation scale decreases as a function of scale, and because users underestimate corner scale.

Corner Scale Finally, let $\mu(\delta S_{ij}, S, U)$ be the mean bias of the corner scale error for the user U at the scale S, and let $\sigma(\delta S_{ij}, S, U)$ be the standard deviation of the set. Because δS_{ij} was shown to fit a Gaussian distribution in Section 2.4.3, if a user defines a corner P_i with scale $S(P_i)$, then $p = 0.92$ is the probability that the user error in scale satisfies

$$\mu(\delta S_{ij}, S, U) - 2\sigma(\delta S_{ij}, S, U) \leq \delta S_{ij} \leq \mu(\delta S_{ij}, S, U) + 2\sigma(\delta S_{ij}, S, U), \tag{2.23}$$

where $\delta S_{ij} = S(P_i) - S(Q_j)$, where Q_j is a corner on the image object at some point q_j near p_i. Given a corner P_i, there is likely to be a corner on the image object defined by Q_j such that

$$S(P_i) - \mu(\delta S_{ij}, S, U) - 2\sigma(\delta S_{ij}, S, U) \leq S(Q_j) \leq S(P_i) - \mu(\delta S_{ij}, S, U) + 2\sigma(\delta S_{ij}, S, U). \tag{2.24}$$

Additionally, in light of the variation in the error distributions for corner scale on the two objects tested, the user errors in scale can be predicted more closely if the user data is gathered on objects which are representative for a particular application. If this is not possible due to a lack of a priori object models, the user data should be gathered for a variety of objects, and if critical, the error should be based on the worst case performance.

2.6 Conclusions

Based on a simple set of user experiments, we have been able to extract a wealth of information about the degree and nature of the errors users make in the specification

of polygonal object boundaries. With respect to the questions posed in Section 2.2.3, we have shown

A) 1. User errors in distance and scale do not show any clear relationship to the scale of the corner on the object being specified.
2. Both the error in corner orientation and the absolute value of the corner angle bias decrease as a function of corner scale.

B) The user ability to specify the boundary of an unknown object is severely hindered if the object and background functions are arbitrary unknown textures with the same average grey value.

C) User errors in boundary specification are not significantly affected by the presence or absence of noise and shading in an image.

D) All errors depend on which user specifies the boundary. The differences among users are accentuated for the distance and scale error measures.

E) The user errors on corner scale depend on the shape of the object being sketched.

In addition to addressing these questions, we were able to develop the user error model described in Section 2.5, which predicts a bound for each of the corner errors defined in Section 2.2.2. For errors in corner angle and scale, the predictions were based on the result that the error sets for each user at each scale behave as a Gaussian distribution.

2.A Instructions

Please outline the object in each of the images presented with the tool for polygonal shape specification which works as follows:

Mouse:

Left Button Press
Specify start and corner points

Middle Button Press
Last point in polygon (connect to first)

Keyboard:

n - Request next image

r - Redo drawing for this image

q - Quit run

Practice Run

You may practice using the tool by outlining the objects in the practice images. Each of the objects should be outlined by specifying exactly one point for each of the corners on the image objects.

1. Press Practice in dialog box using Left mouse button.
2. Repeat until comfortable:
   - Specify object boundaries in image (Mouse - Left and Middle).
   - Request next image (**n**) or repeat drawing (**r**).
3. Quit practice run (**q**).

Experiment

In each of the 20 images used in the experiment, a single object is present. Please outline the object by specifying a point with the left mouse button for each corner in the image object, and close the polygon with the middle mouse button.

Important:

- You have 10 minutes to outline the objects in all 20 images.
- A point must be specified for each corner in the object, however the location of each point needn't be perfect. *Please refrain from using a lot of care and time in specifying each point.*
- Please refrain from using the redo facility (**r**) unless corners have not been specified, or a mouse button is accidentally pressed causing a seriously erroneous drawing.
- Please stop only when finished with all images.

If the instructions are unclear, please request clarification now. The practice run can also be repeated with these instructions in mind by choosing Practice in the dialog box.

1. Press Start in dialog box using Left mouse button.
2. Specify object boundaries in image (Mouse - Left and Middle).
3. Request next image (**n**).

Chapter 3

An A* Algorithm for Inexact Polygon Matching

3.1 Introduction [1]

In Chapter 2, we compared the geometric properties of two polygons near corresponding points. This allowed us to evaluate the type and extent of user error in the performance of polygonal sketching tasks. Prior to the comparison of corresponding points in the user and reference polygons, each point in the user polygon must be matched with some (at most one) point in the reference or model polygon. The matching task is problematic because users sometimes skip points in a reference polygon and insert some which are not.

Matching polygons, either of which may contain points missing in the other is a common problem in computer vision applications. In particular, in model based vision, matching a polygon extracted from a database with one extracted from an image can be difficult due to noise in an image, and due to occluding objects in an imaged scene. Even when a polygonal boundary is compared with a correct object model, an isomorphic match may not exist because the boundary obtained from an image may include points not present in the model and miss some that are. It is therefore often useful to seek a *best fit* or an *inexact match* of two polygons, such as that in Figure 3.1, rather than an isomorphism. If a polygon is viewed as a cyclic graph of nodes, this can be approached by introducing a similarity measure between nodes in two polygons.

As suggested by Ballard and Brown in [BB82], such a measure can be used to generate an association graph in which a maximal clique is sought using graph theoretical methods. A binary decision is made for each pair of nodes which can be generated from the polygonal graphs, to determine whether it may be part of a match and thus should be incorporated in the association graph. A match is then selected by seeking a maximal clique in the association graph.

To match 3D wire-frames in the stereo system described by Buurman and Duin

[1] A shortened version of this chapter will appear in [OGA94].

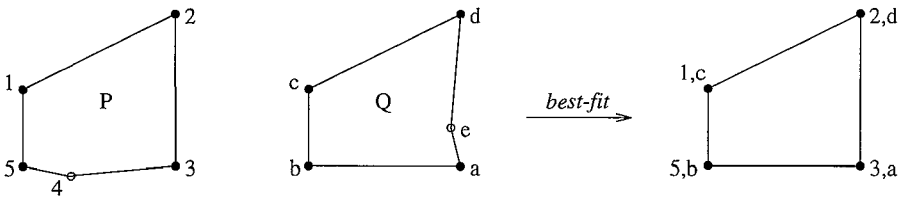


Figure 3.1: Polygons P and Q and the *best fit.*

in [BD90], for each node[2] extracted from a scene, a best match is sought in a model. If there is sufficient support, a vote is added for support of the model. If the model receives sufficient support as a whole, verification is sought. This is similar to the decision process in the association graph method, but stops when there is sufficient support for some hypothesis.

Geometric hashing, introduced by Lamdan and Wolfson in [LW88] and analized by Gavrila and Groen in [GG92], can be used to match polygons as follows. A polygon is viewed as a set of m interest points, which might correspond to the points $p = (x, y)$ which define it. For each of the the $m(m-1)$ ordered point pairs in a model polygon, a basis for a coordinate system is defined and the (model, basis) is stored at the coordinates of each of the remaining $m-2$ interest points in the hash table. When a polygon made up of n interest points is extracted from an image, a basis pair is selected, and for each of the $n-2$ remaining interest points, a vote is given to every (model, basis) pair stored at its coordinates in the hash table. If some (model, basis) pair has received sufficient support, the process halts, and verification is sought. Otherwise, a new basis is selected from the object interest points. Geometric hashing has been successfully applied to several 2D and 3D vision problems. It is particularly suitable when an object and its transformation (location and orientation) are to be identified among numerous alternatives.

For inexact matching, there are two primary drawbacks shared by these methods. First, because node pairs are eliminated prior to the matching step, global considerations cannot influence the selection of node pairs to be included in a solution match. Second, a best fit of the polygons is neither defined nor guaranteed. The maximal clique finding favors matching as many nodes as possible which can easily result in matching nodes which should have been skipped [BB82]. Both the 3D wire-frame matching and the geometric hashing methods generate a hypothesis based on a voting mechanism, and decide upon a match based on a verification step [BD90, LW88, GG92]. Whereas a maximal clique fully describes the match in terms of node pairs, some extra effort either in terms of storage or computation, is required to obtain the match with geometric hashing or 3D wire-frame matching. In the method proposed here, we separate the node pair evaluation step from the matching step.

[2]The nodes correspond to edges in the wire-frame.

This allows us to quantify the characteristics of a best fit, and to develop an algorithm guaranteed to produce one.

Because we are concerned with identifying which parts of two polygons should be paired up in addition to whether the polygons are well matched, the L_2 metric for comparing polygonal shapes, introduced by Arkin, et al in [ACH+91] is not applicable. There are two additional drawbacks in using the L_2 metric for matching problems. First, although it performs well in the presence of noise, points missing due to occluding objects may severely effect the shape, and therefore result in a very high value for the metric for an otherwise good match. Secondly, because it is invariant with respect to affine transformations (rotation, translation, and scale) it is inapplicable for some matching problems. For example, for model based vision systems in which the camera position is known a priori, it should be possible to distinguish a large object from a much smaller one of a similar shape. In matching a user specified polygon with one in an image, no form of affine transform should be tolerated, because the absolute position of the points is relevant.

In our method, each pair of nodes which can be generated from the two polygons is evaluated using a "templates and spring" cost function. The cost function is designed to be adaptable with respect to its sensitivity to affine transformations, and to decrease as a function of similarity between the nodes. As shown in Section 3.2, it incorporates aspects of the geometrical relationship of a node with its neighbors, and thus acts much like the voting mechanism used in geometric hashing. It is used to produce a cost matrix containing an entry for every node pair generated by the two polygons. In Section 3.3, an A* algorithm is described to find an optimal path through the matrix, which corresponds to a permissible match between cyclic graphs. Our algorithm is particularly efficient in its exploitation of the cyclic characteristics of the graphs.

Because the node evaluation/voting procedure is completely separate from the matching procedure, every possible match of the two polygons can be obtained. Methods which eliminate some pairs in the evaluation stage [BB82], or generate a hypothesis about the best fit without evaluating all node pairs [BD90, LW88], cannot guarantee an optimal fit of the polygons.

The algorithm is designed to allow paths which result in a match with skipped nodes such as that shown in Figure 3.1. A node may be skipped at an added *jump cost* J, if this will result in a reduction of the total match cost. The jump cost J thus determines whether matches with skipped nodes can be obtained. We describe an empirical method for finding an optimal value for the jump cost J in Section 3.5. Results are described in Section 3.6. In Section 3.7, we analyze the complexity of the algorithm, and in Section 3.8, minor modifications are discussed which make the method applicable for matching 3D polyhedra.

3.2 The Cost Function

Let $P = \{p_i\}_{i=1}^{n}$ be a polygon defined by n points $p_i \in \mathbf{R}^2$. The key properties of a polygon can be described either in terms of the points $p_i \in P$, or in terms of the line

segments L_i^P between them:

$$L_i^P = \{(1-v)p_i + vp_{i+1} : 0 \leq v \leq 1\} \text{ for } 1 \leq i < n, \text{ and} \quad (3.1)$$
$$L_n^P = \{(1-v)p_n + vp_1 : 0 \leq v \leq 1\}. \quad (3.2)$$

We can view a polygon as a directed cyclic graph of nodes, $G_P = \{\mu_i^P\}_{i=1}^n$, each with a predecessor μ_{i-1}^P and a successor μ_{i+1}^P. Whether a polygonal graph G_P should be defined with nodes represented by points $\mu_i^P = p_i$ or by line segments $\mu_i^P = L_i^P$ depends on the application. For example, the position of the points is significant in the matching problem in Chapter 2, and $\mu_i^P = p_i$ is a suitable node representation for comparing the polygons. In model based vision systems (cf [GAW93]), however, position is irrelevant, as the object may be transformed in relation to the model. In this case the representation $\mu_i^P = L_i^P$ is convenient, because a number of position independent properties can easily be derived with it.

3.2.1 General Form of the Cost Function

We apply the "template and spring" paradigm [FE73], to design the cost functions used to evaluate node pairs in polygonal graphs. If $G_P = \{\mu_i^P\}_{i=1}^n$ and $G_Q = \{\mu_j^Q\}_{j=1}^m$ are polygonal graphs, then we compare a node $\mu_i^P \in G_P$ with a node $\mu_j^Q \in G_Q$ based on the similarity of μ_i^P and μ_j^Q (templates). We can also evaluate the relationship of a node $\mu_i^P \in G_P$ with its neighbors in G_P, in comparison with the relationship of a node $\mu_j^Q \in G_Q$ to its neighbors in G_Q (spring).

Templates A *template* function $t(\mu_i^P, \mu_j^Q)$ measures the similarity of $\mu_i^P \in G_P$ and $\mu_j^Q \in G_Q$. For example, if $\mu_i^P = p_i$, then $t(\mu_i^P, \mu_j^Q)$ can be defined in terms of the distance $\| p_i - q_j \|$ between two points $p_i \in P$ and $q_j \in Q$.

Springs A *spring* function measures the difference between the relationship $r(\mu_i^P, \mu_k^P)$ of μ_i^P to $\mu_k^P \in G_P$, for some $k \neq i$ with the relationship $r(\mu_j^Q, \mu_\ell^Q)$ of μ_j^Q to $\mu_\ell^Q \in G_Q$, for some $\ell \neq j$. For example, if $k = i \pm 1$ and $\ell = j \pm 1$, the a spring function compares the relationship of the two nodes to their immediate neighbors.

If the node representation is $\mu_i^P = p_i$, then a spring function might compare the length and direction of the vectors $p_i - p_k$ and $q_j - q_\ell$, as in geometric hashing [LW88, GG92]. If there is an isomorphism between G_P and G_Q, and if μ_i^P and μ_j^Q should be matched, then $r(\mu_i^P, \mu_k^P)$ should be close to $r(\mu_j^Q, \mu_\ell^Q)$ when $k = i + \delta \bmod n$ and $\ell = j + \delta \bmod m$, for $1 \leq \delta \leq n$.

Voting Let $t(\mu_i^P, \mu_j^Q)$ be a template function which compares a node $\mu_i^P \in G_P$ with a node $\mu_j^Q \in G_Q$. Further let $s\{r(\mu_i^P, \mu_k^P), r(\mu_j^Q, \mu_\ell^Q)\}$ be a spring function which compares the relationship of μ_i^P to μ_k^P with the relationship of μ_j^Q to μ_ℓ^Q. Suppose that both $t(\cdot)$ and $s(\cdot)$ are bounded functions which increase as a function of similarity, and

that their sum

$$\Omega\{r(\mu_i^P, \mu_k^P), r(\mu_j^Q, \mu_\ell^Q)\} = t(\mu_i^P, \mu_j^Q) + s\{r(\mu_i^P, \mu_k^P), r(\mu_j^Q, \mu_\ell^Q)\} \tag{3.3}$$

satisfies

$$0 \leq \Omega\{r(\mu_i^P, \mu_k^P), r(\mu_j^Q, \mu_\ell^Q)\} \leq B.$$

For some range of δ we want to add a *vote* for the pair (μ_i^P, μ_j^Q) if the response to $\Omega\{r(\mu_i^P, \mu_k^P), r(\mu_j^Q, \mu_\ell^Q)\}$ measure is strong, where $k = i + \delta \bmod n$ and $\ell = j + \delta \bmod m$. To obtain a voting mechanism based on the response to $\Omega(\cdot)$, we can define a threshold value T, and add a vote for the pair (μ_i^P, μ_j^Q) for every δ resulting in $\Omega\{r(\mu_i^P, \mu_k^P), r(\mu_j^Q, \mu_\ell^Q)\} > T$. Alternatively, we can define a function which increases rapidly, so that small differences in the response to $\Omega(\cdot)$ are accentuated. Choosing the latter is advantageous because it makes the estimation of a threshold parameter unnecessary. In the voting function described below, we use the exponential function to translate the response to $\Omega(\cdot)$ to a vote for or against a given pair (μ_i^P, μ_j^Q). For the exponential to have the desired effect, we require the bound $B \gg 1$.

If G_P and G_Q were known to be isomorphic, we could define a voting function as

$$\sum_{\delta=1}^{n} e^{\Omega\{(\mu_i^P, \mu_k^P), (\mu_j^Q, \mu_\ell^Q)\}},$$

where $k = i + \delta \bmod n$ and $\ell = j + \delta \bmod n$. If $t(\cdot)$ and $s(\cdot)$ are appropriately defined, then the pairs with the most votes should be members of the correct match. If, however, a point μ_u^P has no match in G_Q, then the contribution to the above sum will be negligible for $\delta \geq u - i$. For $\delta > u - i$, the relationship $r(\mu_i^P, \mu_{i+\delta}^P)$ is comparable with that of $r(\mu_j^Q, \mu_{j+\delta+1}^Q)$. To be sure we get a vote for δ when μ_i^P and μ_j^Q are well matched and G_P and G_Q are not isomorphic, we thus want to summarize votes for neighboring elements of μ_k^P and μ_ℓ^Q.

If we let Δs denote the maximum number of nodes which may be missed in G_P or G_Q, then we can evaluate the support for pairing $\mu_i^P \in G_P$ with $\mu_j^Q \in G_Q$ with the voting function

$$V(\mu_i^P, \mu_j^Q) = \sum_{\delta=1}^{\max(n,m)} \left(\sum_{a=-\Delta s}^{\Delta s} \sum_{b=-\Delta s}^{\Delta s} e^{\Omega\{(\mu_i^P, \mu_k^P), (\mu_j^Q, \mu_\ell^Q)\}} \right) \tag{3.4}$$

where $k = i + \delta + a \bmod n$ and $\ell = j + \delta + b \bmod m$.

The inner sum allows votes from node pairs that would be missed if either or both polygons were missing up to Δs consecutive points. The outer sum is used to summarize support over both polygonal graphs for the node pair.

A Normalized Cost Function Given a voting function $V(\mu_i^P, \mu_j^Q)$ which increases as a function of similarity of $\mu_i^P \in G_P$ and $\mu_j^Q \in G_Q$, the maximum value for all pairs

is given by

$$V_{\max}(G_P, G_Q) = \max_{\substack{1 \le i \le n \\ 1 \le j \le m}} V(\mu_i^P, \mu_j^Q),$$

and the function

$$F(\mu_i^P, \mu_j^Q) = \frac{V_{\max}(G_P, G_Q) - V(\mu_i^P, \mu_j^Q)}{V_{\max}(G_P, G_Q)}, \tag{3.5}$$

is a normalized cost function, with $0 \le F(\mu_i^P, \mu_j^Q) \le 1$ for all $\mu_i^P \in G_P$ and all $\mu_j^Q \in G_Q$. The cost function $F(\mu_i^P, \mu_j^Q)$ decreases as a function of node similarity.

The Global Cost Function Given two polygonal graphs $G_P = \{\mu_i^P\}_{i=1}^n$ and $G_Q = \{\mu_j^Q\}_{j=1}^m$ we can define a match $M = \{(a_{1r}, a_{2r})\}_{r=1}^R$ of G_P and G_Q in terms of R pairs of indices of the nodes $\mu_{a_{1r}}^P$ and $\mu_{a_{2r}}^Q$ to be matched. Given a cost function $F(\mu_i^P, \mu_j^Q)$ for comparing nodes $\mu_i^P \in G_P$ and $\mu_j^Q \in G_Q$, the cost of a match M is defined as

$$C(M) = J(m + n - 2R) + \sum_{r=1}^{R} F(\mu_{a_{1r}}^P, \mu_{a_{2r}}^Q) \tag{3.6}$$

where J is the cost of a missing node in either G_P or G_Q and $m + n - 2R$ is the total number of nodes in G_P and G_Q unaccounted for in M. Because $F(\mu_i^P, \mu_j^Q)$ decreases as a function of similarity of μ_i^P and μ_j^Q, an optimal match of the polygonal graphs is one for which $C(M)$ is minimal. The definition of a best fit therefore depends on the template and spring functions with which $F(\mu_i^P, \mu_j^Q)$ is defined, and on the value of the jump cost J.

3.2.2 Line Based Templates and Springs

Representation of a polygonal graph G_P in terms of line segments $\mu_i^P = L_i^P$, as defined in Equation 3.1, lends itself to the definition of template and spring functions which are invariant under various affine transformations. In this section, we define measures which are invariant under translations and rotations but sensitive to changes in scale. They are particularly suitable when the camera position relative to the imaged plane is known a priori, but the exact location of the object is not.

Template Because the polygons being compared should not differ in scale, but may differ in position and orientation, length is the significant attribute of a line segment to be compared in a template function. Let $\ell(L_i^P) = \|p_{i+1} - p_i\|$. The ratio of the lengths of line segments L_1 and L_2 defined by

$$\lambda(L_1, L_2) = \frac{\min(\ell(L_1), \ell(L_2))}{\max(\ell(L_1), \ell(L_2))}, \tag{3.7}$$

satisfies $0 \leq \lambda(L_1, L_2) \leq 1$. It provides a measure of similarity for line segments $L_i^P \in G_P$ and $L_j^Q \in G_Q$, which increases as the difference in line length decreases, and is therefore a suitable template function. It is later exploited in the spring function as well.

Springs Because we aim to define a spring function $s\{r(\mu_i^P, \mu_k^P), r(\mu_j^Q, \mu_\ell^Q)\}$, which is sensitive to changes in scale, but not to translations and rotations, the relationship $r(L_i^P, L_k^P)$ should express the distance and relative difference in angle between two line segments in a graph G_P.

In defining these aspects of the geometric relationship between line segments, it is clear that the two points which determine each of the line segments should be given equal weight. We thus employ the midpoint

$$m_i = \frac{1}{2}p_i + \frac{1}{2}p_{i+1} \tag{3.8}$$

of a line segment $L_i^P \in G_P$ in examining the relationship between two line segments, as geometric comparisons based on this point are equally influenced by the two points which define the line segment.

The distance $d(L_i^P, L_k^P)$ between two line segments in G_P is now defined as $\|m_i - m_k\|$. Given a line segment $L_k^P \in G_P$ with $k \neq i$, we can define a connecting line from the midpoint m_i of L_i^P to the midpoint m_k of L_k^P with

$$L_{ik}^P = \{(1-v)m_i + vm_k : 0 \leq v \leq 1\}.$$

Now $\lambda(L_{ik}^P, L_{j\ell}^Q)$, as defined in Equation 3.7, provides a bounded measure which compares the distance between line segments L_i^P and L_k^P, with the distance between line segments L_j^Q and L_ℓ^Q.

Consider the size of the corner angles $\alpha_{ik} = \alpha(L_i^P, L_{ik}^P)$ and $\alpha_{ki} = \alpha(L_{ik}^P, L_i^P)$ shown in Figure 3.2.2. The two angles express the rotational and positional relationship of the line segments L_i^P and L_k^P. The angle $\alpha(L_i^P, L_{ik}^P)$ can be compared with a corresponding angle derived from a pair (L_j^Q, L_ℓ^Q) if we define

$$\Delta\alpha[(L_i^P, L_k^P), (L_j^Q, L_\ell^Q)] = \frac{\pi - |\alpha(L_i^P, L_{ik}^P) - \alpha(L_j^Q, L_{j\ell}^Q)|}{\pi}. \tag{3.9}$$

If the direction of the midpoint of L_ℓ^Q from the midpoint of L_j^Q as defined by the vector $L_{j\ell}^Q$ differs from the direction from m_i to m_k defined by L_{ij}^P, then $|\alpha(L_i^P, L_k^P) - \alpha(L_j^Q, L_\ell^Q)|$ will be nonzero, and will increase as the difference increases. Meanwhile, if the angle $\alpha(L_i^P, L_k^P)$ differs from $\alpha(L_j^Q, L_\ell^Q)$, then $|\alpha(L_i^P, L_k^P) - \alpha(L_j^Q, L_\ell^Q)|$ will be nonzero, and will increase as the difference does.

Thus $0 \leq \Delta\alpha[(L_i^P, L_k^P), (L_j^Q, L_\ell^Q)] \leq 1$ for all pairs $(L_i^P, L_k^P) \in P$ and $(L_j^Q, L_\ell^Q) \in Q$, and $\Delta\alpha(\cdot)$ increases as the difference in the corner angles decreases. $\Delta\alpha(\cdot)$ is invariant under affine transformations, and can thus be used in any cost function which tolerates one or more such transformations.

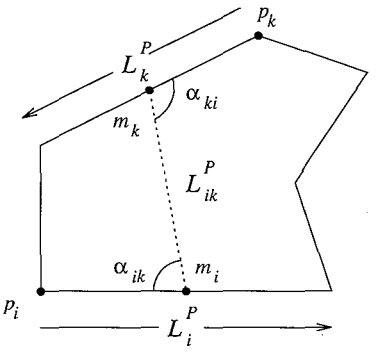


Figure 3.2: A polygonal graph G_P represented as a set of line segment nodes $\{L_i^P\}_{i=1}^n$. Relationships between line segments $L_i^P \in G_P$ and $L_k^P \in G_P$ can be evaluated in terms of the line L_{ik}^P connecting their midpoints. Here $\alpha_{ik} = \alpha(L_i^P, L_{ik}^P)$ and $\alpha_{ki} = \alpha(L_k^P, L_{ki}^P)$

A spring measure for line based polygonal graphs, which is sensitive to changes in scale, but invariant under translations and rotations, can now be expressed in terms of the relative distance and angles between line segments. Let

$$\begin{aligned} s\{r(L_i^P, L_k^P), r(L_j^Q, L_\ell^Q)\} = \; & \lambda(L_k^P, L_\ell^Q) + \lambda(L_{ik}^P, L_{j\ell}^Q) + \Delta\alpha[(L_i^P, L_k^P), (L_j^Q, L_\ell^Q)] \\ & + \Delta\alpha[(L_k^P, L_i^P), (L_\ell^Q, L_j^Q)], \end{aligned} \tag{3.10}$$

Note that we incorporate a term $\lambda(L_k^P, L_\ell^Q)$ which measures the relative of the two segments being evaluated. If the L_k^P and L_ℓ^Q are the relative neighbors of L_i^P and L_j^Q, and the latter match, then the lengths of L_k^P and L_ℓ^Q should be the same.

Template and Springs If we define

$$\Omega\{(L_i^P, L_k^P), (L_j^Q, L_\ell^Q)\} = \lambda(L_i^P, L_j^Q) + s\{r(L_i^P, L_k^P), r(L_j^Q, L_\ell^Q)\}, \tag{3.11}$$

using Equations 3.7 and 3.10, then we have template and spring measure for line based polygonal graphs, which satisfies the requirements outlined in Section 3.2.1. Because Ω is defined in terms of the functions λ and $\Delta\alpha$, we have

$$0 \leq \Omega\{(L_i^P, L_k^P), (L_j^Q, L_\ell^Q)\} \leq 5$$

for all pairs (L_i^P, L_k^P) and (L_j^Q, L_ℓ^Q), and $\Omega\{(L_i^P, L_k^P), (L_j^Q, L_\ell^Q)\}$ increasing as a function of the similarity of line segment pairs. It can therefore be incorporated in the voting function $V(\mu_i^P, \mu_j^Q)$ in Equation 3.4.

3.2.3 Point Based Templates and Springs

If the location of the points in both polygons being compared is relevant, then a point based representation $P = \{p_i\}_{i=1}^n$ of a polygonal graph $G_P = \{\mu_i^P\}_{i=1}^n$ is useful for making local comparisons. In this case, a template and spring function must be sensitive to any affine transformation of one polygon with respect to to the other.

Template A template measure $t(\mu_i^P, \mu_j^Q)$, intolerant of affine transformations can be defined in terms of the distance between the points $p_i \in P$ and $q_j \in Q$. Given $P = \{p_i\}_{i=1}^n$ and $Q = \{q_j\}_{j=1}^m$, the maximum distance between any pair of points on the two polygons is

$$D_{\max}(P, Q) = \max_{\substack{1 \le i \le n \\ 1 \le j \le m}} \|p_i - q_j\|.$$

The normalized measure of distance

$$d(p_i, q_j) = \frac{D_{\max}(P, Q) - \|p_i - q_j\|}{D_{\max}(P, Q)} \tag{3.12}$$

increases as a function of closeness of the points p_i and q_j and satisfies $0 \le d(p_i, q_j) \le 1$ for all $p_i \in P$ and $q_j \in Q$, and thus provides an appropriate template measure for polygonal graphs with point based nodes.

Springs The geometrical relationships between a point $p_i \in P$ and its neighbors, to be compared in the spring function should reflect the required constraints on transformations between the polygons to be compared. By incorporating measures of angle and orientation, we penalize rotations between the two polygons. Measures of length are incorporated to restrain scaling.

We can express the geometrical properties of the relationship between a point p_i and its neighbors on a polygon in terms of a triplet $C = \{c_1, c_2, c_3\}$, with $c_2 = p_i$ and $c_1, c_3 \in P$, which defines a corner at p_i if $c_1 \neq c_2 \neq c_3$. Likewise, if $d_2 = q_j$ and $d_1, d_3 \in Q$, then a triplet $D = \{d_1, d_2, d_3\}$ defines a corner at q_j.

If P and Q are isomorphic, and p_i and q_j should be matched, then the corners $C = \{c_1, c_2, c_3\} = \{p_{i-a}, p_i, p_{i+b}\}$ and $D = \{d_1, d_2, d_3\} = \{q_{j-a}, q_j, q_{j+b}\}$ should be similar in terms of angle, orientation and scale for nonzero pairs (a, b). Comparing corners C and D at p_i and q_j respectively for a set of pairs $\{(a, b)\}$ is similar to comparing node relationships for varying δ in the general spring function.

We denote the angle of a corner C with $\alpha(C) = \alpha(c_2 - c_1, c_2 - c_3)$. This is the angle α in Figure 3.3. Clearly if P and Q match and p_i and q_j should be paired, $\alpha(C) \approx \alpha(D)$ when C and D are generated with the same pair (a, b). The angles of C and D can be compared by applying $\Delta\alpha(\cdot)$ to $\alpha(C)$ and $\alpha(D)$. For corners, we therefore use

$$\Delta\alpha(C, D) = \frac{\pi - |\alpha(C) - \alpha(D)|}{\pi}, \tag{3.13}$$

as the spring component for corner angle.

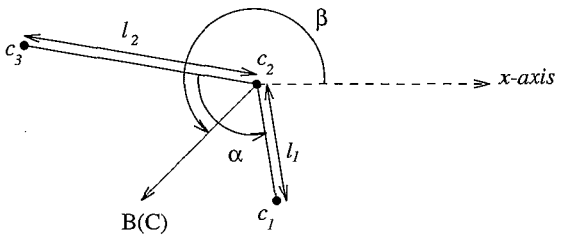


Figure 3.3: Significant properties of a corner $C = \{c_1, c_2, c_3\}$: the corner location, c_2, the small corner angle $\alpha = \alpha(C)$, the orientation $\beta = \beta(C)$, and $l_1 = \|c_2 - c_1\|$ and $l_2 = \|c_2 - c_3\|$, the length of the lines which meet at the point c_2.

Let $\beta(C)$ be the orientation of a corner C, defined by the angle the bisecting vector $B(C)$, makes with the x-axis (Figure 3.3). This definition of corner orientation is used because it gives equal weight to both line segments which define a corner C. Let $\beta_{\text{diff}}(C, D)$ denote the difference in orientation of corners C and D. Then

$$\beta_{\text{diff}}(C, D) = \cos^{-1}\left(\frac{B(C) \cdot B(D)}{\|B(C)\|\|B(D)\|}\right),$$

and

$$\Delta\beta(C, D) = \frac{\pi - \beta_{\text{diff}}(C, D)}{\pi} \tag{3.14}$$

provides a measure of the closeness of orientation. Since $0 \leq \beta_{\text{diff}}(C, D) \leq \pi$, for any corners C and D, we have $0 \leq \Delta\beta(C, D) \leq 1$. Note that $\Delta\beta(C, D)$ penalizes rotations, but is invariant to translation and scale. It can therefore be used for comparing corners in applications where an object's orientation is known a priori, as may be the case in a model based vision system in which the stable positions of an object are known.

We can now define a spring function to compare the relationships of points in two polygons with their respective neighbors. Let

$$s\{r(C), r(D)\} = \Delta\alpha(C, D) + \Delta\beta(C, D) + \lambda(c_1 - c_2, d_1 - d_2) + \lambda(c_2 - c_3, d_2 - d_3), \tag{3.15}$$

where $\Delta\alpha(\cdot)$ and $\Delta\beta(\cdot)$ are defined in Equations 3.13 and 3.14. The function $\lambda(\cdot)$ defined in Equation 3.7 will penalize a difference in scale.

Template and Springs We define

$$\Lambda(C, D) = d(c_2, d_2) + s\{r(C), r(D)\} \tag{3.16}$$

as a template and spring function to compare corners C and D derived from two polygons, where $d(\cdot)$ and $s(\cdot)$ are defined in Equations 3.12 and 3.15. Together, the components of $\Lambda(C, D)$ assure its sensitivity to any form of affine transformation.

For any corner triplet C defined by three distinct points in the polygon P, and any corner D defined by three distinct points in Q, we have

$$0 \leq \Lambda(C, D) \leq 5$$

and $\Lambda(C, D)$ increasing as a function of similarity of the corners C and D.

Voting Because the spring contribution is measured in terms of three points in each of the polygons, we introduce the following variation on the voting function in Equation 3.4. To evaluate the similarity of a point $p_i \in P$ and a point $q_j \in Q$ we want to compare the corners $P_i = \{p_{i-1}, p_i, p_{i+1}\}$ and $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$ generated by the immediate neighbors in P and Q. To obtain a good measure when up to Δs consecutive points may be missed in either P or Q, we also evaluate $\Lambda(C, D)$ for a range of corners about p_i and q_j. Let $P(k_1, i, k_2) = \{p_{k_1}, p_i, p_{k_2}\}$ and $Q(\ell_1, j, \ell_2) = \{q_{\ell_1}, q_j, q_{\ell_2}\}$. To evaluate the similarity of a point $p_i \in P$ and a point $q_j \in Q$, we define:

$$V(\mu_i^P, \mu_j^Q) = \sum_{a_1=1}^{\Delta s+1} \sum_{a_2=1}^{\Delta s+1} \sum_{b_1=1}^{\Delta s+1} \sum_{b_2=1}^{\Delta s+1} e^{\Lambda\{P(k_1,i,k_2),Q(\ell_1,j,\ell_2)\}} \tag{3.17}$$

where k_1, k_2, ℓ_1 and ℓ_2 are

$$\begin{aligned} k_1 &= i - a_1 \bmod n \quad \text{and} \quad k_2 = i + a_2 \bmod n \\ \ell_1 &= j - b_1 \bmod m \quad \text{and} \quad \ell_2 = j + b_2 \bmod m \end{aligned}$$

If $J_{\max}$ is the maximum number of jumps allowed in P or Q, to prevent the evaluation of collapsing corners ($C = \{c_1, c_2, c_3 = c_1\}$), we evaluate Equation 3.17 with

$$\Delta s \leq \min\left\{J_{\max}, \left\lfloor \frac{\min\{m, n\}}{2} \right\rfloor - 1\right\}.$$

3.3 An A* Algorithm to Find an Optimal Match

Given polygonal graphs $G_P = \{\mu_i^P\}_{i=1}^n$ and $G_Q = \{\mu_j^Q\}_{j=1}^m$, and a cost function $F(\mu_i^P, \mu_j^Q)$ as defined in Equation 3.5 to compare nodes, we can use it to compute the cost of all $n \times m$ node pairs which can be generated form G_P and G_Q, and store the results in a cost matrix C. An example is shown in Figure 3.4. Given such a matrix, the matching problem can now be approached by finding a minimum cost path through the matrix C which satisfies the following constraints specific to polygon matching.

Because each node $\mu_i^P \in G_P$ can be matched with at most one node $\mu_j^Q \in G_Q$, at most one element in a row or column can contribute to a path. The cyclic nature of polygons implies that a path through C for an acceptable match of G_P and G_Q must have a diagonal form. To be precise, any path through the (i, j) matrix element must

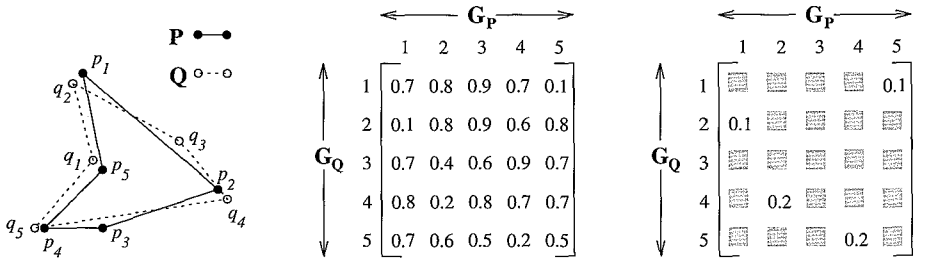


Figure 3.4: A sample matching problem. On the left, two polygons to be matched, in the middle the cost matrix C computed with a function $F(\mu_i^P, \mu_j^Q)$, and on the right, the minimum cost permissible path in the matrix.

have a preceding neighbor (i_p, j_p) with $i_p \leq i-1$ and $j_p \leq j-1$, and the next node in the path (i_n, j_n) must satisfy $i_n \geq i+1$ and $j_n \geq j+1$.[3]

If an isomorphism between G_P and G_Q could be assumed, then a minimum cost path could be found by simply summing the costs for the n pairs in each of the n diagonal paths in the cost matrix and choosing that with the minimum total, an $O(n^2)$ operation. However, if up to k nodes may be missed in P and ℓ may be missed in Q, then there are $k\ell \cdot m^k n^\ell$ paths to be inspected in C. Assuming $m \approx n$, an exhaustive search has run time $O(k^4 n^{4k+1})$, including the selection of the minimum cost path. To avoid this expense, we perform a heuristic search of the cost matrix to find a minimum cost diagonal path which represents a permissible match between the polygonal graphs G_P and G_Q.

Consider the cost matrix illustrated in Figure 3.5 containing the cost c_{ij} of pairing each element in $G_P = \{\mu_i^P\}_{i=1}^n$ with each element in $G_Q = \{\mu_j^Q\}_{j=1}^m$. A permissible match of G_P and G_Q should account for all nodes $\mu_i^P \in G_P$ and $\mu_j^Q \in G_Q$, either in a match pair, or by explicitly being skipped and adding a jump cost J to the total cost of the match. Thus, a diagonal path which represents a permissible match, must cover the entire length and breadth of the cost matrix.

To facilitate the search, we maintain a list of path nodes η, to store relevant data about a path from its start position to its current position. If up to S_Q nodes may be skipped in G_Q, then we don't know in which of the first S_Q rows of C an optimal path will start, nor on which of the $m - S_Q$ rows it will end. To address this issue, we introduce a set of virtual start nodes $S = \{\eta_s\}_{s=1}^n$ with corresponding positions $(s, 0)$ at the top of the matrix C, and a set of virtual goal nodes $G = \{\eta_g\}_{g=1}^n$ with positions $(g, m+1)$ at the bottom of C. A diagonal path which accounts for every node in G_P and G_Q starting at η_s must end at η_g with $g = s+1 \bmod n$. Mentally inserting a start and goal row for the example in Figure 3.4 helps clarify this.

[3]These statements should of course be interpreted in terms of the nodes in each polygon and thus modulo n and m should be understood where applicable.

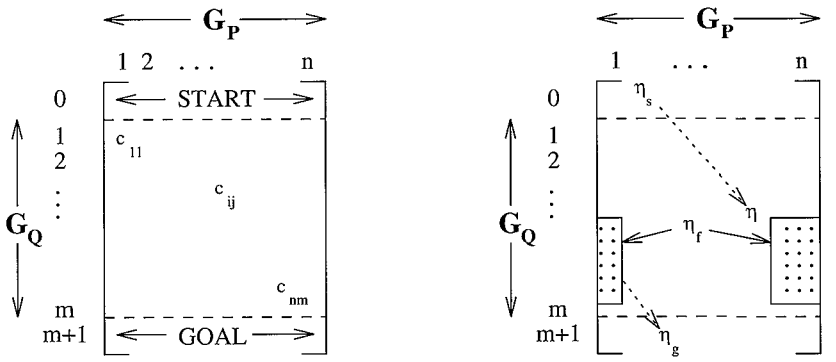


Figure 3.5: The cost matrix for matching G_P and G_Q starting with a set of virtual start nodes and ending with virtual goal nodes. On the right, the expansion of a node η in a path is illustrated.

To find an optimal path through the matrix we apply the general purpose heuristic search algorithm described in [Nil80], and listed in Figure 3.6. Beginning with the first start node, we expand it by generating a node for each of its candidate neighbors and place the new nodes on the OPEN list to be expanded, ordered according to how promising they appear. This process is repeated for each node on the list starting with the most promising, and halts when a goal node has been reached.

The position of a node η in the OPEN list depends on its *estimated* cost $\hat{f}(\eta)$. Every node η on the OPEN list has two cost functions associated with it: $g(\eta)$ is the cost of the path from the start node η_s to η, and $h(\eta)$ is the cost to reach the goal node η_g associated with η_s. The cost of a path starting at η_s constrained to pass through η is given by

$$f(\eta) = g(\eta) + h(\eta). \tag{3.18}$$

When a node η is created, the cost $g(\eta)$ is given by the sum of the costs including the jump costs, of the path from η_s to η, and is maintained cumulatively as the nodes are expanded. The cost $h(\eta)$ to reach the goal node is unknown. $h(\eta)$ might however be estimated with a heuristic function $\hat{h}(\eta)$, and the estimated cost of a path from η_s to η_g constrained to pass through η can be defined as

$$\hat{f}(\eta) = g(\eta) + \hat{h}(\eta) \tag{3.19}$$

If

$$\hat{h}(\eta) \leq h(\eta), \tag{3.20}$$

then the algorithm in Figure 3.6 is called A* and is guaranteed to produce an optimal path through C [Nil80].

Thus, to assure the path we find is optimal, there are two key points which must be considered. First, when expanding a node, all neighbors which might be part

1. For $1 \leq s \leq n$, create a start node η_s and add it to the OPEN list.
2. While OPEN is nonempty

   Remove first node η from OPEN.
   If $\eta \in G$ (goal node reached)
   Stop. Retrace path or evaluate match.
   Else
   (a) Expand η.
   (b) Place each successor η_f in OPEN list positioned according to $\hat{f}(\eta_f)$.
   (c) Place η on CLOSED list.
3. Destroy all items on OPEN and CLOSED lists.

Figure 3.6: The *best first* search algorithm.

of a permissible path, as determined by how many nodes may be skipped in G_P and G_Q must be inspected. Second, we need a heuristic function $\hat{h}$ which satisfies Equation 3.20.

3.3.1 Node Expansion

If the number of nodes in the polygonal graphs differ ($n \neq m$), then at least $|n-m|$ nodes must be skipped in the larger of the two graphs for any permissible match. Without loss of generality, assume $G_P = \{\mu_i^P\}_{i=1}^n$ is mapped horizontally, and $G_Q = \{\mu_j^Q\}_{j=1}^m$ is mapped vertically in the cost matrix, as in Figure 3.5. Skipping k nodes in G_P results in a path with k horizontal jumps over columns in the match matrix. For each path node η, we maintain $H_{\min}(\eta)$, the number of nodes $\mu_i^P \in G_P$ which must be skipped between η and its goal node η_g, if the path corresponds with a permissible match. This value is initiated for each start node as $H_{\min}(\eta_s) = \max\{n-m, 0\}$, because at least that many will have to be skipped in G_P to match it with G_Q. Additionally, to find matches such as those depicted in Figures 3.1 and 3.4, we allow up to $J_{\max}$ additional jumps in G_P and G_Q. Suppose we denote the number of additional horizontal jumps permitted between a node η and its goal node η_g with $H_{\max}(\eta)$. Then every path starts with $H_{\max}(\eta_s) = J_{\max}$.

Analogously, we maintain $V_{\min}(\eta)$, the number of nodes $\mu_j^Q \in G_Q$ which must be skipped between η and its goal node η_g, which is initiated for each start node with $V_{\min}(\eta_s) = \max\{m-n, 0\}$, and of course $V_{\max}(\eta_s) = H_{\max}(\eta_s)$.

When a node η positioned at (i,j) is expanded, the values of $H_{\min}(\eta)$, $V_{\min}(\eta)$, $H_{\max}(\eta)$, and $V_{\max}(\eta)$ determine the nodes to be generated in the expansion as follows. The candidate nodes for the next path element η_f will be positioned at (i_f, j_f) with

$$i+1 \leq \quad i_f \quad \leq i+1+H_{\min}(\eta)+\min\{H_{\max}(\eta), m+1-j, g-i \bmod n\} \quad (3.21)$$

$$j+1 \leq \quad j_f \quad \leq j+1+V_{\min}(\eta)+\min\{V_{\max}(\eta), m+1-j, g-i \bmod n\}. \quad (3.22)$$

The minimum prevents the expansion of invalid or nonexistent nodes near the bottom of the matrix.

For a newly generated node η_f, $H_{\min}(\eta_f)$, $H_{\max}(\eta_f)$, $V_{\min}(\eta_f)$ and $V_{\max}(\eta_f)$, depend on the number of jumps in the horizontal and vertical directions between the expanded node η and the new node η_f, and the values for the node η. They are computed using the formulas in Equations 3.30, 3.31, 3.32, and 3.33, based on the arguments in Appendix 3.A.

Using the above formulas to compute the expansion regions, if all nodes η_f are expanded when (i_f, j_f) satisfy Equations 3.21 and 3.22, then every permissible path will be investigated, and no impermissible paths will be generated.

3.3.2 The Heuristic Function

To be sure the path we obtain is optimal, we must guarantee satisfaction of Equation 3.20. The smaller the difference $h(\eta) - \hat{h}(\eta)$, the greater the heuristic power of $\hat{h}$, and in turn the more efficient the search process. We thus consider the knowledge we have about the path from η to its goal node η_g. We know there must be a minimum of $J_{\min}(\eta) = \max\{H_{\min}(\eta), V_{\min}(\eta)\}$ jumps made, each with a cost of J. $J \cdot J_{\min}(\eta)$ is therefore a lower bound on the cost of the path from η to its goal node η_g. Further, the path must reach the bottom of the matrix, either by adding a node for the kth row in the matrix or by jumping over it. Another lower bound on the cost of a path from a node η to its goal η_g is thus

$$\eta_{\min}(\eta) = \sum_{k=j+1}^{m} \min\{J, \min_{1 \le i \le n} F(\mu_i^P, \mu_k^Q)\}. \tag{3.23}$$

Because $\eta_{\min}(\eta)$ depends only on the row position j for the node η, it can be computed once for all rows prior to starting the search. If we now define

$$\hat{h}(\eta) = \max\{J \cdot J_{\min}(\eta), \eta_{\min}(\eta)\}, \tag{3.24}$$

then Equation 3.20 is satisfied, and using Equation 3.19 to order the OPEN list will result in the first path found being an optimal path.

When there is a good match between the two polygons, the cost of the pairs (μ_i^P, μ_j^Q) in the match will correspond to the elements which contribute to $\eta_{\min}(\eta)$ in Equation 3.23, and the heuristic power of $\hat{h}$ is very high, making the algorithm quite efficient. If it is unknown whether the two polygons match, or a match may be poorly behaved (as in the experiments in Sections 3.5 and 3.6), then variations on $\hat{h}(\eta)$ which reduce the evaluation of the minimum to the elements of C which can be reached from η in moving to η_g may have substantially higher heuristic power, preventing extensive (costly) node expansion. Unless a good match is known to exist (as in the application in Chapter 2), there is a direct trade off between the heuristic power of $\hat{h}$ and the computational requirements to compute it. This will be considered further in Section 3.7.

Further, if the method is used to decide whether G_P and G_Q match, then a decision

threshold T, appropriate for the application, will be necessary to decide whether the cost of a match is low enough. Because $\hat{f}(\eta_f)$ is a lower bound on the cost of the path constrained to pass through η_f in reaching its goal node, η_f needn't be added to the OPEN list for expansion if $\hat{f}(\eta_f) > T$. This modification is clearly useful for reducing the number of nodes maintained and expanded in the decision process.

3.4 Experimental Methods and Error Measures

Depending on the application, the best match of two graphs will be that for which the number of one or both of the following errors is minimized:

false positive – the presence of an incorrect node pair; and

false negative – the absence of a node pair which should have been in the match.

In this section, we describe the experimental methods and error measures used to tune the parameter J, and evaluate the algorithm for a specific application. The general scheme is to create a pair of polygons, one of which is a controlled random distortion of the other, so that the polygons are well matched and we know the correct pairing of nodes. After running our algorithm on the polygon pair, we compare the resulting match with the correct match, and measure the number of false positives and negatives.

The polygon distortion mechanism described here is designed to simulate a worst case user for the experiments in Chapter 2. Clearly for other applications, permitted affine transformations should be incorporated in the distortion scheme.

Random Polygon Distortions Suppose, given a polygon $P = \{p_i\}_{i=1}^n$, and some integer $c \geq 0$, we have a polygon $Q = \{q_j\}_{j=1}^n$ where for $1 \leq i \leq n$,

$$\|q_j - p_i\| \leq \varepsilon_i \quad \text{if} \quad j = i + c \bmod n \tag{3.25}$$

If $\varepsilon_i = 0$, Q is simply a cyclic permutation of P. Let $\varepsilon_{\max}$ be the maximum error a user is permitted in specifying a corner on a polygon. Further, let d_i^P denote the minimum distance from a point $p_i \in P$ to any other point $p_k \in P$:

$$d_i^P = \min_{\substack{1 \leq k \leq n \\ k \neq i}} \|p_i - p_k\|.$$

If, for $1 \leq i \leq n$, we choose

$$\varepsilon_i = \min\left\{\frac{d_i^P}{2}, \varepsilon_{\max}\right\}, \tag{3.26}$$

then given a polygon $P = \{p_i\}_{i=1}^n$ and some $c \geq 0$, for every polygon $Q = \{q_j\}_{j=1}^n$ for which each $q_j \in Q$ satisfies Equation 3.25, there is a one to one mapping between the points in P and those in Q, which can be expressed in terms of distance. For $\varepsilon_{\max} > 0$, such a polygon Q might be thought of as a disturbed permutation of P.

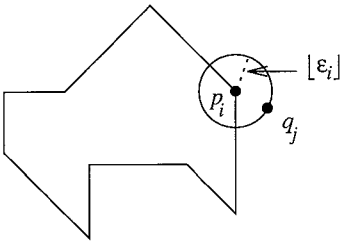


Figure 3.7: The polygon P used for tuning the jump cost J for the user experiments, and the selection of a point $q_j \in Q$ from $p_i \in P$.

Given a polygon $P = \{p_i\}_{i=1}^{n}$ and some integer $c \geq 0$, we can produce a *worst case* polygon Q which satisfies Equation 3.25 as follows. For each $i \in [1, n]$, we generate a point $q_j \in Q$, where $j = i + c \bmod n$, by selecting a random point from those on a digital approximation (generated with [Bre77]) to the circle of radius $\lfloor \varepsilon_i \rfloor$ centered at p_i (Figure 3.7).

Error Measures Suppose now that we create X such polygons $Q = Q(x)$, using a randomly chosen $c = c(x)$ for each $x \in [1, X]$. For each polygon, we can apply our algorithm to obtain a match $M[P, Q(x), J]$ using J as the jump cost in Equation 3.6. Let $E^{+}_{\text{false}}(M[P, Q(x), J])$ denote the number of false positives $(\mu_i^P, \mu_j^{Q(x)})$ in $M[P, Q(x), J]$. In this case, any pair $(\mu_i^P, \mu_j^{Q(x)})$ for which $j \neq i + c(x) \bmod n$ is a false positive. We define the percentage of false positives in X experiments as

$$E^{+}(P, X, J) = 100 \cdot \frac{\sum_{x=1}^{X} E^{+}_{\text{false}}(M[P, Q(x), J])}{\sum_{x=1}^{X} n(M[P, Q(x)])}, \tag{3.27}$$

where $n(M[P, Q(x)])$ is the number of pairs found in a correct match of P and $Q(x)$.

Likewise, let $E^{-}_{\text{false}}(M[P, Q(x), J])$ be the number of false negatives in a match. These are the points found in the correct match of P and $Q(x)$, but not in found in $M[P, Q(x), J]$. We can define the percent of false negatives in X experiments as

$$E^{-}(P, X, J) = 100 \cdot \frac{\sum_{x=1}^{X} E^{-}_{\text{false}}(M[P, Q(x), J])}{\sum_{x=1}^{X} n(M[P, Q(x)])}. \tag{3.28}$$

If $M[P, Q(x), J]$ is a match of P and $Q(x)$ to be evaluated, and $C[P, Q(x), J]$ is the subset of all pairs in $M[P, Q(x), J]$ also found in the correct match $M[P, Q(x)]$, then

$$n(C[P, Q(x), J]) = n(M[P, Q(x), J]) - E^{+}_{\text{false}}(M[P, Q(x), J]).$$

The percentage of pairs correctly found for a series of X experiments is thus given by

$$F(P, X, J) = 100 \cdot \frac{\sum_{x=1}^{X} n(C[P, Q(x), J])}{\sum_{x=1}^{X} n(M[P, Q(x)])}. \tag{3.29}$$

Depending on the application at hand, we may want to minimize $E^+(P, X, J)$, $E^-(P, X, J)$, or both, thereby maximizing $F(P, X, J)$.

Missing Points Because the algorithm is designed to match polygons each of which may be missing points found in the other, we want to examine the errors when points are eliminated from either P or Q, or both. Given a reference polygon $P = \{p_i\}_{i=1}^{n}$ as above and a polygon $Q = \{q_j\}_{j=1}^{n}$ generated from it, we can generate two new polygons $P_{\text{cut}} = \{p_k\}_{k=1}^{K}$ and $Q_{\text{cut}} = \{q_\ell\}_{\ell=1}^{L}$ by deleting $cut = 2n - K - L$ randomly selected points from either P or Q or both.

To evaluate the behavior for a set of X randomly generated polygon pairs $P_{\text{cut}}, Q_{\text{cut}}$, we can apply defined Equations 3.27, 3.28 and 3.29. To indicate that *cut* points have been randomly eliminated from the polygon pair, the measures are then referred to as

$$E^+_{\text{cut}}(P, X, J),\ E^-_{\text{cut}}(P, X, J), \text{and}\ \ F_{\text{cut}}(P, X, J),$$

respectively.

3.5 The Optimal Jump Cost J

In Equation 3.5, we defined the cost of matching nodes from two polygonal graphs. An equally relevant question is when two nodes should *not* be matched. As used in Equation 3.6, the *jump cost* J is the value for which it should be cheaper to skip a point pair in matching G_P and G_Q, than to include it. Combined with the cost function $F(\cdot)$, the value of J determines whether a fit such as that depicted in Figure 3.4 can be found.

The criteria used to determine the optimal jump cost for a particular application, are application dependent, and closely related to the criteria used to design the cost function. We determine the jump cost empirically, by varying its value, and choosing a value for which specific criteria are met.

Given a representative reference polygon $P = \{p_i\}_{i=1}^{n}$ for a specific application, we might seek a value for J, for which the number of false positives, the number of false negatives or the total number of errors is minimized. If $T(n)$ nodes might be missing in either polygonal graph for a specific application, then the number of errors should be minimal for matches between two polygons with up to $T(n)$ missing points. For most applications, the tolerated number of missing points will increase with the number of points on the polygons to which it is applied.

To find the optimal value of J for matching the user defined polygons in Chapter 2, we use the polygon $P = \{p_i\}_{i=1}^{9}$ in Figure 3.7, and set $T(n) = \lfloor n/4 \rfloor = 2$. The polygon is similar to, but simpler than those in the user tests. $T(n) = \lfloor n/4 \rfloor$ is more than the

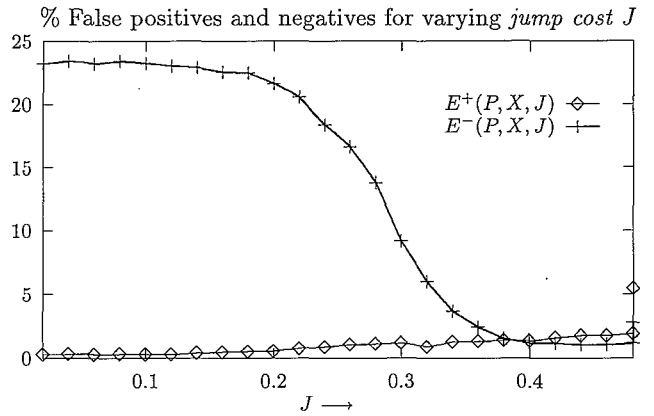


Figure 3.8: The result of varying J on $E^+(P, X, J)$, the percent of false positives, and on $E^-(P, X, J)$, the percent of false negatives, when Q is a distorted version of P, and $P = \{p_i\}_{i=0}^{9}$ is the polygon depicted in Figure 3.7. The results are summarized for $X = 500$ experiments for each value of $cut \in \{0, 1, 2\}$.

number of misses than occurred in practice, and therefore sufficient for tuning the jump cost for this application.

For each value of J tested, we performed $X = 500$ experiments for each of the permissible *cut* values $\{0, 1, 2\}$. The results are pooled for all three cut values in Figure 3.8, where we show $E^+(P, X, J)$ and $E^-(P, X, J)$, the percent of false positives and negatives found the 1500 experiments for each value of J. For the evaluation of user corner errors, we choose the jump cost $J = 0.38$ as this is the point where the false positives and negatives are nearly equal.

3.6 Results

Having obtained an optimal value $J = 0.38$ for the jump cost, we proceeded to apply the method to the polygons depicted in Figure 2.2 to which the user experiments were applied. Both polygons are made up of points with connecting line lengths $\ell(p_{i+1} - p_i) \in \{15, 30, 45, 60, 75\}$, as measured in screen pixels. In the experiments, we allowed a maximum distance between generated point pairs to be $\varepsilon_{\max} = 7$ (see also Equation 3.26). Each of the polygons contains $n = 17$ points, and we allow $T(n) = \lfloor n(P)/4 \rfloor$ points to be missed in either polygon. We therefore tested the method for $cut \in \{0, 1, \ldots 4\}$. The percentage of false positives for each value of *cut*, is given in Table 3.1. The tests were performed on $X = 100$ polygon pairs per *cut* value for each of the polygons.

As expected, the performance deteriorated slightly as the number of points deleted from the polygons increased. Note that in all cases tested, less than 8% of the matches

% False Positives – $E^{+}_{\text{cut}}(P, X, J)$						
Points *cut*	0	1	2	3	4	All
% Polygon 1	0.18	2.00	3.78	6.31	7.36	3.69
% Polygon 2	0.00	0.62	2.13	4.27	6.51	2.50
% Both	0.09	1.31	2.95	5.29	6.94	3.09

Table 3.1: The percentage of false positives, as defined in Equation 3.27, found in matching a set of polygon pairs. For each of the polygons in Figure 2.2, there were $X = 100$ polygon pairs P_{cut}, $Q_{\text{cut}}(x)$ matched for each value of $cut \in \{0, 1, \ldots, 4\}$. The individual and combined results are presented for all cut values and for both polygons tested.

% Correct Pairs Found – $F_{\text{cut}}(P, X, J)$						
Points *cut*	0	1	2	3	4	All
% Polygon 1	99.35	97.62	95.62	93.12	93.47	96.05
% Polygon 2	97.52	96.69	96.28	94.59	93.26	95.80
% Both	98.44	97.16	95.95	93.85	93.37	95.92

Table 3.2: The percentage of node pairs correctly identified with our algorithm. For each of the polygons in Figure 2.2, there were $X = 100$ polygon pairs P_{cut}, $Q_{\text{cut}}(x)$ matched for each value of $cut \in \{0, 1, \ldots, 4\}$. The individual and combined results are presented for all cut values and for both polygons tested.

were bad, and that on average only 3% were. This means that 97% of the matched pairs were correct.

In Table 3.2, we show the percentage of point pairs in a correct match that were identified with our algorithm. On average we find 96% of the point pairs in a correct match.

Given the random nature of the input data, this is quite good. Recall that corner angle and line length both weigh heavily in the voting function (see Equation 3.16). These factors can be badly influenced by the local differences in shape between P and Q. This problem is accentuated when points are eliminated from either polygon, which accounts for the deterioration for increasing cut values. In practice, users are far less random in the geometric deformations of the input polygon in relation to the reference polygon, and the percentage of points correctly matched is close to 100%.

3.7 Complexity Analysis

The algorithm for inexact matching described in this chapter is a two step procedure. In the first step, a matrix of costs is computed with an element for each pair of nodes which can be generated from the two polygons, and in the second, a best fit is determined by finding a minimum cost path through the matrix. We consider the complexity of each step separately and conclude with a comparison with geometric hashing.

3.7.1 Cost Computations

The computations required to compute the cost of matching a node in G_P with one in G_Q depend primarily on the number of consecutive nodes which may be missed in either of the polygonal graphs. Based on the voting function in Equation 3.4, voting requires $n \cdot (2\Delta s + 1)^2$ steps. Since this is required for all node pairs in the two polygons, the cost computation is of order $O(n^3 \Delta s^2)$. Likewise, the variant introduced in Equation 3.17 and used in our tests is of order $O(n^2 \Delta s^4)$.

3.7.2 Matching Computations

The complexity of the matching step can be evaluated in terms of the total number of nodes expanded, and in terms of the number of nodes generated in the process. The number of nodes expanded depends on how well the polygons fit with respect to the cost function. For every node η expanded, there are $(H_{\min}(\eta) + H_{\max}(\eta))(V_{\min}(\eta) + V_{\max}(\eta)) \approx J^2_{\max}$ nodes generated, each of which must be inserted in the ordered list OPEN.

The run time performance of the *best first* search algorithm shown in Figure 3.6 is clearly dominated by the sorted insertion of the newly generated nodes η_f. The insertion is accomplished with a merge sort (cf. [CLR90]). We first sort the $J^2_{\max}$ newly generated nodes η_f using an insert sort with worst case run time $\Theta(J^4_{\max})$, and then merge the two sorted node lists.

Best Case When there is a clear match between the nodes in the two polygons, then after the n start nodes, the only nodes expanded are the roughly n nodes on a best match. In the best case, there are roughly $2n \cdot J^2_{\max}$ nodes generated, and each must be inserted at the appropriate position in the OPEN list. Since the size of the OPEN list is of order $O(n \cdot J^2_{\max})$ (for the best case), the merge step has worst case run time $\Theta(n \cdot J^2_{\max})$

So we may conclude that algorithm has run time $\Omega(n^2 J^2_{\max})$ in the best case, that is when the number of nodes expanded is of order $O(n)$. The best case for the number of nodes expanded is examined because it does arise in practice (e.g. Chapter 2), and its run time is therefore of interest. However, there is no reason to believe that it is coupled with best case sorting, which is why we use worst case run time for sorting when evaluating the *best case* run time.

Worst Case When no good match exists, the number of expanded nodes can increase dramatically. In practice, this case arises when a decision must be made as to whether two polygons match. To reach a conclusion, there must be a decision threshold T, which determines whether the cost of a path as defined in Equation 3.5 is low enough to to infer a match, as in [Ant93].

In the worst case, we expand every node generated until we reach a goal. For the start nodes, we generate $n \cdot J_{\max}^2$ nodes. Expanding all of those, generates at most $n \cdot J_{\max}^2 \cdot J_{\max}^2$ nodes. Continuing this process until a goal node is reached will result in the generation and expansion of at most $n \cdot J_{\max}^{2n}$ nodes . The algorithm therefore has worst case run time $\Theta(n^2 J_{\max}^{4n})$.

Preventing the Worst Case Given T, an application dependent threshold (which can be selected in much the same fashions as was jump cost in Section 3.5), it can be used to reduce the run time costs of the matching procedure significantly. Suppose we know a no match conclusion should be reached, if the minimum cost path has cost $f(\eta) > T$. Since $\hat{f}(\eta) \leq f(\eta)$, it is unnecessary to insert a freshly generated node η_f in the OPEN list if $\hat{f}(\eta_f) > T$. The length of the OPEN list can thus be reduced significantly if T is small. In the most extreme case, if $T \leq \eta_{\min}(0)$ as defined in Equation 3.23, then a no match decision will be made before the first start node is generated. On the other hand, if $T \geq \sum_{j=0}^{n} \max_{1 \leq i \leq n} F(\mu_i^P, \mu_j^Q)$, then until a goal node is reached, all nodes generated will be expanded.

When the presence of a good match is in question, and no small threshold T has been defined, we can prevent unnecessary growth of the OPEN list if we modify the heuristic function $\hat{h}(\eta)$ to provide a better estimate of $h(\eta)$. From Equation 3.23, it is easy to see that $\eta_{\min}(\eta)$, and therefore $\hat{h}(\eta)$, increases as the as number of elements contributing to the minimum for each row is decreased. However, the minimum must be evaluated for all row members which the path from η to its goal η_g can pass through or it may not satisfy $\hat{h}(\eta) \leq h(\eta)$, as required.

For every start node η_s, the set of elements in each row which can contribute to a permissible path from η_s to its goal node η_g can be determined at a cost of $O(nJ_{\max})$, Computing $\eta_{\min}(\eta)$ as a function of the start position as well as the row position can be done with $O(n^3 J_{\max})$ operations, rather than the $O(n^2)$ operations required to compute $\eta_{\min}(\eta)$ solely as a function of row position as in Equation 3.23. This additional cost is clearly worthwhile if worst case behavior may occur. If however, best case behavior is expected, then the additional computations are superfluous, and the formula used to compute $\eta_{\min}(\eta)$ in Equation 3.23 is more efficient.

3.7.3 Evaluation

In Table 3.3, we summarize the best ($\Omega(\cdot)$), the average ($O(\cdot)$), and the worst ($\Theta(\cdot)$) case run time for the experiments performed in Section 3.6 on the polygons in Figure 2.2. Note first that the best case behavior is $\Omega(n^2 J_{\max}^2)$ as predicted in the discussion above. Secondly, the worst case performance $\Theta(n^6 J_{\max}^2)$ is much better than predicted, in spite of the random nature of the data. The worst case prediction is

Run time complexity: # expanded × # generated					
$\Omega(n)$	$\Omega(n, J_{\max})$	$O(n)$	$O(n, J_{\max})$	$\Theta(n)$	$\Theta(n, J_{\max})$
n^3	$n^2 J_{\max}^2$	n^4	$n^3 J_{\max}^2$	n^7	$n^6 J_{\max}^2$

Table 3.3: The runtime behavior of the algorithm for the 500 matches performed in Section 3.6 on each of the two polygons in Figure 2.2. The values $\Omega(n)$ and $\Omega(n, J_{\max})$ show the best case behavior, $O(n)$ and $O(n, J_{\max})$, the average behavior, and $\Theta(n)$ and $\Theta(n, J_{\max})$ the worst case. The variations due to the value of *cut* were small enough not to influence the above values. Each of the polygons contain $n = 17$ points and the maximum number of points allowed to be skipped in a match was $J_{\max} = 4$. The combined results are presented for all cut values and for both polygons tested.

based on a breadth first search ($\hat{h}(\eta) = 0$), where in fact we perform a depth first search, which, depending on the definition of $\hat{h}(\eta)$, is far more efficient.

In the experiments here, using $\hat{h}(\eta)$ defined in terms of $\eta_{\min}(\eta)$ as defined in Equation 3.23, the worst case runtime is far better than $O(n^{4J_{\max}+1} J_{\max}^4)$, the computation time required for a brute force decision.

Based on the complexity analysis performed by Gavrila and Groen in [GG92], the voting procedure required by geometric hashing for matching two polygons as considered in this chapter has worst case run time $\Theta(n^6)$, assuming a point pair in each polygon is used to define a basis set. Given that our method produces a match if it is present, and quantifies its quality, in addition to performing a decision task, the run time behavior as summarized in Table 3.3 compares very well with that of geometric hashing.

3.8 Inexact Polyhedra Matching

It is interesting to note that with minor modifications, the method described in this chapter can be applied to the problem of matching polyhedra in three dimensions. Suppose we have a polyhedron $L_P = \{L_P^i\}_{i=1}^n$ represented in terms of n line segments L_P^i between points $p_{i_1}, p_{i_2} \in \mathbf{R}^3$:

$$L_i^P = \{(1 - v)p_{i_1} + vp_{i_2} : 0 \leq v \leq 1\}$$

The major difference between a polyhedron and a polygon is that it is not cyclic, and thus no ordering or connectivity of the set L_P can be assumed. A vertex v_j at position $p_j = (x_j, y_j, z_j)$ is connected to its neighbors by some unknown number of line segments L_i^P and thus the position alone is insufficient to evaluate and compare vertices from two polyhedra. If a vertex is defined in terms of its position p_j and the position of each vertex to which it is connected by a line segment, then a set of templates and spring might be defined to compare it with a vertex from another polygon.

It is of course possible to represent a polyhedron as a set of connected faces, each of

which can be described in terms of a polygonal graph. The template contribution to a cost function might then be computed with one of the functions defined in Section 3.2. Because the run time of the algorithm depends on the size of the node sets, both of these alternatives can be used to reduce the run time considerably, as they can be used to represent a polyhedron with substantially fewer nodes than required if line segment representation is used.

For the sake of discussion, suppose we view a polyhedron as a graph $G_P = \{\mu_P^i\}_{i=1}^n$ made up of nodes $\mu_P^i = L_P^i$. Given a second polyhedron represented by $G_Q = \{\mu_Q^j\}_{j=1}^m$, we can define a template and spring cost function $F(\mu_P^i, \mu_Q^j)$ to compare each pair of nodes which can be generated from the two graphs G_P and G_Q. Depending on the transformations tolerated for a specific application, template and spring functions can be defined which are minor variations on those defined in Section 3.2.2 for line based polygonal graphs.

We may thus produce an $n \times m$ cost matrix containing the value of the metric $F(\mu_P^i, \mu_Q^j)$ for comparing the ith and jth members of G_P and G_Q respectively. To find an optimal match using the cost matrix, we again note that every node in G_P and G_Q should be accounted for, either by being explicitly skipped or by being included in a match. This means that any match will cover the entire width and breadth of the matrix. We can thus start and end with virtual start and goal nodes as in Section 3.3 forcing the polyhedron which is vertically mapped to be fully accounted for. Since no row or column can be included more than once in a match, the columns considered in the expansion should be those not yet accounted for in the path up to node η. Skips can be permitted, by allowing jumps over rows.

The restriction that the path through the cost matrix be diagonal was necessary for polygons due to the cyclic nature of the data sets being matched. Dropping the restriction has two consequences for the search algorithm. First, $\hat{h}(\eta)$ resulting from the definition of $\eta_{\min}(\eta)$ in Equation 3.23 is the best estimation which can be obtained prior to the actual search procedure. Second, the run time behavior is somewhat worse, as any two nodes might be neighbors. This means in the estimations presented in Section 3.7, $J_{\max}$ should be replaced by n.

3.9 Conclusions

We have described a technique for inexact matching of polygons in which the evaluation of node pairs is separated from the optimal fit algorithm. The characteristics of a cyclic graph representing a polygon in terms of either points or line segments are incorporated in a "template and spring" cost function used to measure the similarity of nodes from different graphs. Further, we presented an efficient A* algorithm guaranteed to produce an optimal match with respect to the cost function. A method for selecting an optimal jump value is described, and the method is shown to work successfully in the user experiment application described here as well as for the model based vision system described in [GAW93]. Finally, we analized the run time complexity of the algorithm, and showed that unnecessary growth can be prevented in various

application contexts. Our algorithm can easily be extended to match 3D polyhedra as illustrated in Section 3.8.

In addition to matching the user input sketches with polygonal object boundaries in Chapter 2, in the model based robot vision system described in [GAW93], our matching algorithm provides a reliable and efficient mechanism for deciding whether a polygonal object boundary detected with the sensor system should be identified as the boundary of an object expected in the scene, the description of which is obtained from a database.

3.A Node Expansion

In the A* algorithm described in Section 3.3, the set of nodes generated when a node is expanded determine which paths through the cost matrix, and therefor which matches of two polygonal graphs, can be obtained. When a node η positioned at (i, j) is expanded, the values of $H_{\min}(\eta)$, $V_{\min}(\eta)$, $H_{\max}(\eta)$, and $V_{\max}(\eta)$ determine the positions of the nodes to be generated, based on Equations 3.21 and 3.22. These values are determined when a node is first generated as follows.

The values of $V_{\min}(\eta_f)$, $H_{\min}(\eta_f)$ $V_{\max}(\eta_f)$, and $H_{\max}(\eta_f)$ associated with each newly generated node η_f, depend on its position in the cost matrix with respect to η, and on the values of $V_{\min}(\eta)$, $H_{\min}(\eta)$ $V_{\max}(\eta)$, and $H_{\max}(\eta)$ for its predecessor. A candidate node η_f created to follow η involves $J_h = i_f - (i+1)$ horizontal and $J_v = j_f - (j+1)$ vertical jumps in the matrix. This corresponds to skipping J_h nodes in the polygonal graph G_P and J_v nodes in G_Q.

To simplify the discussion, suppose $J_h \geq 0$ and $J_v = 0$. Then if $H_{\min}(\eta) \geq 0$, at least some of the J_h horizontal jumps were required, and the number of required horizontal jumps in the new node will be

$$H_{\min}(\eta_f) = \max\{H_{\min}(\eta) - J_h, 0\}.$$

If $J_h \geq H_{\min}(\eta)$, then $J_h - H_{\min}(\eta)$ permitted, but not strictly required, horizontal jumps were made to reach (i_f, j_f), and the number of permitted horizontal jumps in the remainder of the path through η_f is

$$H_{\max}(\eta_f) = H_{\max}(\eta) - (J_h - H_{\min}(\eta)).$$

Note, however, if jumps not strictly required, as maintained in $H_{\min}(\eta)$, are made in the horizontal direction, then in the remainder of the path, the same number of vertical jumps *must* be made to guarantee a permissible matching of the polygonal graphs. This restriction can be enforced if

$$\begin{aligned} V_{\min}(\eta_f) &= V_{\min}(\eta) + (J_h - H_{\min}(\eta)) \\ V_{\max}(\eta_f) &= V_{\max}(\eta) - (J_h - H_{\min}(\eta)) \end{aligned}$$

Extending the above arguments, we can define formulas for $H_{\min}(\eta_f)$, $H_{\max}(\eta_f)$, $V_{\min}(\eta_f)$, $V_{\max}(\eta_f)$ for the general case ($J_h \geq 0$ and $J_v \geq 0$). First, in moving from η to

η_f, the number of not strictly required jumps in the horizontal and vertical directions is given by

$$H_f = H_{\min}(\eta) - J_h \quad \text{and} \quad V_f = V_{\min}(\eta) - J_v$$

It is easy to see that

$$H_{req} = \max\{V_f - H_f, 0\}$$

steps must later be made in the horizontal direction. Likewise

$$V_{req} = \max\{H_f - V_f, 0\}$$

must be made in the vertical direction. The minimum jumps required and the maximum jumps allowed to get from η_f to its goal η_g is given by:

$$H_{\min}(\eta_f) = \max\{(H_{\min}(\eta) - J_h + H_{req}), 0\} \tag{3.30}$$

$$H_{\max}(\eta_f) = H_{\max}(\eta) - \max\{J_h - H_{\min}(\eta), 0\} - H_{req} \tag{3.31}$$

$$V_{\min}(\eta_f) = \max\{(V_{\min}(\eta) - J_v + V_{req}), 0\} \tag{3.32}$$

$$V_{\max}(\eta_f) = V_{\max}(\eta) - \max\{J_v - V_{\min}(\eta), 0\} - V_{req} \tag{3.33}$$

If these formulas are used to compute the expansion regions, then expanding all nodes η_f when (i_f, j_f) satisfy Equations 3.21 and 3.22 prevents investigation of impermissible paths, while it guarantees that every admissible match can be obtained with the algorithm.

Chapter 4

Model Based Corner Detection

4.1 Introduction [1]

If an object boundary can be described by a set of line segments connected at points, then the connection or corner points are sufficient to fully reconstruct the object boundary. In [FW94], Fischler and Wolf show that corners and points of high curvature form the best point set for recovering more general curves. They also show that users select these points when asked to specify the most significant points on a curve.

A sketch derived from a user-defined point set which is specified with a connect-the-dots interaction tool as described in Section 2.3.1, cannot be considered more than an approximation to an object boundary, and is thus not suitable for analysis of object features, such as shape, grey level moments and area. This is due to the indirection of hardware devices for screen location specification, and the variation in human motor capabilities described in [HHN86]. For the task of polygonal boundary specification, variation in user errors is illustrated in Chapter 2 of this thesis. We thus seek a user adaptable method for correcting a user defined polygonal sketch of an image object boundary. Even if users could specify boundaries to a satisfactory degree of accuracy, correction mechanisms reduce the tedium inherent to this task by allowing users to sketch less carefully.

In this chapter, we address issues involved with correcting the corner points on a user sketch of a polygonal image object boundary. Because the corner points fully determine the boundary path for polygonal objects, the accuracy of all object measurements depend on the accuracy of these boundary points. When a polygonal boundary path is specified by a user, we therefore view the user sketch as a set of straight lines connected at corners, and concentrate our attention on correcting the corner locations. This approach is particularly suitable for images in which the object shapes are close to polygonal, as is frequently the case in industrial applications.

Initial attempts to detect and locate corners on image object boundaries were based on inspection of boundary paths already extracted from the image (see for instance [RJ73, FD77]). Because of the local nature of the inspection performed, these methods

[1] Portions of this chapter are reprinted, with permission, from [OG93] ©1993 IEEE.

often fail to detect corners with wide angles. Moreover, because in general, the path on which corners are detected has been extracted using edge detection methods or region based segmentation techniques, those corners which are detected will be poorly localized. By definition, edge detectors are intended to detect points on a smooth boundary path [MH80]. Because the smoothness condition is violated at corners on the object boundary, edge detectors perform poorly at these points. Meanwhile, most region based segmentation methods classify pixels after some form of local averaging. Depending on the size of the region which influences the decision, a pixel near an acute corner will often be misclassified due to the heavy influence of background pixels in the decision process. An object boundary extracted from the partitioning will thus be rounded near corners, and if the corners are detected, they will necessarily be poorly localized.

To address these issues, more recent approaches are geared towards detection of corners in the original grey value image, as opposed to on a geometric path already derived from it [KR82, ZH83, LSWM86, RSB89, MNR90, LT90, SS90]. While the proposed methods vary, they share two basic assumptions:

1. The corner is located between two relatively smooth regions which differ significantly in average grey value. That is, a priori knowledge is assumed about the behavior of the image function on the object and background regions, which can be used to distinguish the two.

2. No a priori knowledge is available regarding the geometrical aspects of the corner. Thus corners of arbitrary angle, orientation, and location must be identified with the corner detector.

The role of interactive segmentation in image analysis is most important when the validity of the first assumption is unknown. In new applications, there is generally no a priori knowledge of the image function on the images being analyzed.

Meanwhile, in interactive segmentation, a user can provide an approximation of the contour in terms of location and form. Detection of corners for which there is a good estimate of the angle, orientation, and location has been neglected in the literature, although this information is frequently available. This is not only the case in interactive segmentation, but also in model based segmentation techniques, and in variations of the active contour method [KWT88], such as that described in [WS92].

Given an approximate geometric model for a corner on an image object contour, we want to extract a model of the image function in the immediate region. The two models can then be combined and exploited to localize the corner on the object boundary as follows. We design a template which should look like the corner in the image based on the geometric and image function models. This is used to locate the corner in the image, using a variation of the cross correlation technique described in [RK76]. We demonstrate our method by employing it to model and locate a variety of corners in a grey value image.

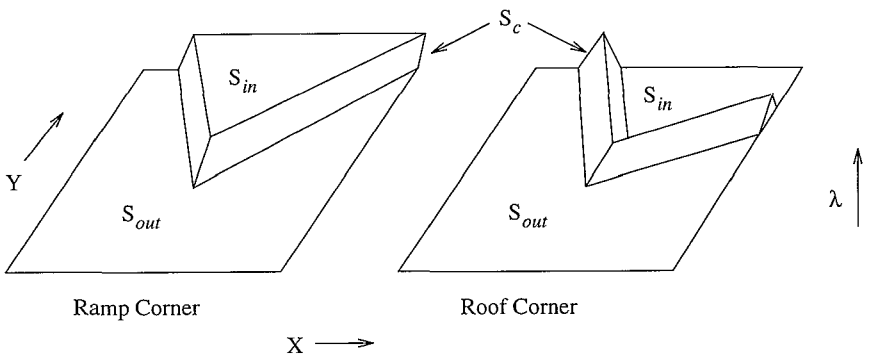


Figure 4.1: Ramp and roof corner models for an image feature measure λ.

Our method provides a context in which a variety of segmentation models can be evaluated for their capacity to locate a corner. With very little effort, it can be extended to other geometrical forms. With this approach, a segmentation model for a new application can be derived far more easily than with traditional methods (often trial and error), and the need for human intervention quickly eliminated.

4.2 From User to Corner Model: an Overview

Suppose we have an image which contains an object, the boundary of which can be described by the set of line segments which connect neighboring points in the ordered set $Q = \{q_j\}_{j=1}^{m}$, with $q_j \in \mathbf{R}^2$ for all $q_j \in Q$. Suppose a user, in the process of specifying the object boundary generates the point set $P = \{p_i\}_{i=1}^{n}$, where for each $p_i \in P$, we have $p_i \in \mathbf{Z}^2$ corresponding to the position of a pixel in a digital image.

For every $p_i \in P$, the user implicitly specifies a corner in the object boundary, the geometrical properties of which are determined by the triplet $P_i = \{p_{i-1}, p_i, p_{i+1}\}$. From the results presented in Chapter 2, we can expect there to be a corresponding corner in the image object boundary defined by a triplet $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$ of neighboring points in the point set which defines the boundary. Unless the user specifies superfluous points near the corner Q_j, or neglects to specify one of the points which define Q_j, then we may assume the relationship between P_i and Q_j satisfies the constraints on position, angle, orientation and scale described by the user error model in Chapter 2.

If there is a corner Q_j in the image object boundary, not occluded by another object, or disturbed by other sources, then based on some image feature measure λ, the image data in a region near the point q_j should be well modeled as a ramp or roof corner as shown in Figure 4.1. For example, λ may measure the grey value intensity, but more complex feature measures (e.g. graininess) may be appropriate to model the image function near the corner.

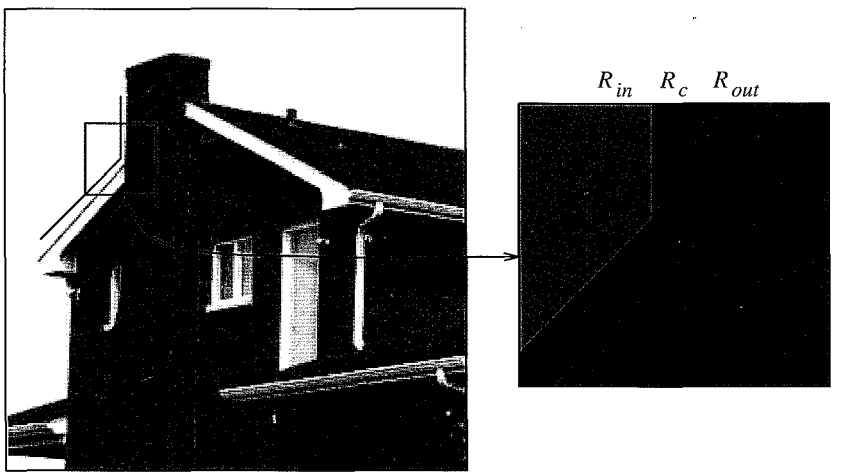


Figure 4.2: Division of a subimage R in three regions of interest, based on a corner P_i in a user input sketch.

In an ideal image, this means that there is some neighborhood S centered at the point q_j, which can be divided in three sections depending on the response to the image feature measure λ. If the object boundary follows the path defined by Q_j in the neighborhood S, then there should be two regions in S separated by the path of Q_j. We call the region which corresponds to the cone of the corner S_{in}, and its counterpart S_{out}. These regions are labeled in Figure 4.1, and are separated by a central or transition region S_c along the path Q_j, in which the image function changes. If λ is an image feature measure which can be used to described the change in the image function near the point q_j, then one of the following must hold.

A) The constant value λ_{in} of the function $\lambda(x, y)$ on S_{in} is distinct from the constant value λ_{out} of $\lambda(x, y)$ on S_{out}. In the central region S_c, the value varies between λ_{in} and λ_{out}. This criterion corresponds with the ideal ramp corner model in Figure 4.1.

B) $\lambda(x, y)$ has the same value in S_{in} and S_{out}. The average value of $\lambda(x, y)$ in the transition region S_c is higher or lower than $\lambda_{in} = \lambda_{out}$. A corner which satisfies these conditions is described by the roof corner model in Figure 4.1.

C) One of the above two statements holds when some portion of S is measured, but not for the entire region. This may be due to an occluding object near the corner, such as a pen lying near or across the corner of a piece of paper, or to changes in the object itself, such as that in the house near the corner outlined in Figure 4.2. Further, nearby objects and lighting conditions such as shadows and reflections may result in interference with the corner model.

D) The section of the contour associated with the corner Q_j is *subjective*. There is no measurable change in the image data in the three regions, but Q_j would be part of the boundary were it fully visible. A subjective corner may be due to low contrast in the image region or due to the presence of an occluding object. Often a human or a high level process (e.g. [GAW93]) can hypothesize the presence of a corner which would be nondetectable based only on the behavior of $\lambda(x, y)$ in the region S.

Based on the results in Chapter 2, given a triplet $P_i = \{p_{i-1}, p_i, p_{i+1}\}$ produced by a user as part of an object boundary specification, we can assume there is a similar corner Q_j on the boundary for which one of the above holds for some image feature measure λ. Based on the corner defined by the triplet P_i, and the user error model in Section 2.5, we can define three regions R_{in}, R_c, and R_{out} which should correspond to the regions S_{in}, S_c, and S_{out} about the corner point q_j on the image object boundary. (See Figure 4.2.) The regions R_{in}, R_c, and R_{out}, can then be used to estimate parameters for a given image feature measure λ in the corresponding image corner regions S_{in}, S_c, and S_{out}.

The results can be used to evaluate the likelihood of a ramp or roof corner Q_j which is similar to P_i, in terms of angle, orientation and location. A model of the image corner can then be constructed based on the likelihood of a ramp or roof model for a measure λ, and the region parameters. The model might be incorporated in a simple template and subsequently used to locate the corner in the image, as illustrated in Sections 4.4 and 4.5. However, more complete corner models which take into account the user error models for in corner angle and orientation derived in Section 2.5 may be constructed.

4.3 The Segmentation Model

Suppose we have an image $I(x, y)$ containing an object, the boundary of which can be expressed as a polygonal path defined by $Q = \{q_j\}_{j=1}^m$. Suppose that for some image feature measure λ, the difference in the response on the object and background regions is significant. Then, for each point $q_j \in Q$, there will be a small region centered about q_j, for which the image function when measured with λ will fit a ramp corner model. Alternatively, the response to λ along the path defined by Q may be measurably different from the response to λ in the object and background regions. The image data will then fit a roof corner model in the regions about the points $q_j \in Q$.

In both cases there will be a transition region along the path of Q near q_j, in which the value of the image feature measure λ changes. The geometry of the transition region depends on the size of the neighborhood over which λ is measured, the width of the point spread function of the optical system used for image formation, and the geometry of the boundary near the point q_j as determined by the triplet $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$. For roof corners, line thickness also contributes to the geometry of the transition region.

Given a corner triplet $P_i = \{p_{i-1}, p_i, p_{i+1}\}$ in a user sketch of the object boundary, let R be an $N \times N$ subimage centered at the point p_i. Based on the user error model

derived in Chapter 2, we can expect there to be a corner on the object boundary described by a triplet $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$ which satisfies a number of constraints with respect to the user defined corner triplet P_i. We can therefore extract a region $R_c \subset R$ centered about the two lines which meet at p_i which is known to contain the path section $Q_j \cap R$ of the object boundary in the subimage R. Given the size of the neighborhood required to measure λ, the region R_c can be defined to contain the entire subset $S_c \cap R$ in which the object to background transition takes place.

If the size N of the subimage R satisfies certain conditions (defined below) with respect to the length of the lines which define the corner at q_j, and the corner angle is *not too small*, then the remainder $R \backslash R_c$ of the region R will be made up of two disjoint regions R_{in} and R_{out}, separated by R_c. See Figure 4.2. If R_{in} and R_{out} are sufficiently large, then measurements on the image data in the respective regions can be used to estimate the parameters of an image function model in the object and background regions.

Based on the user error model and the image corner model, we therefore aim to construct a region R_c for a postulated corner $P_i = \{p_{i-1}, p_i, p_{i+1}\}$ in the user sketch. We then estimate the image function parameters based on the resulting division of R ($R = R_{in} \cup R_c \cup R_{out}$).

4.3.1 The Image Corner Model

Given a corner on an image object boundary defined by a triplet $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$, the following geometrical properties are of interest. Let a and b be the vectors defined by

$$a = q_{j-1} - q_j \quad \text{and} \quad b = q_{j+1} - q_j.$$

The triplet Q_j can then be expressed as $Q_j = \{q_j + a, q_j, q_j + b\}$, and we can derive:

$S(Q_j)$ – the corner scale, which we define as the length of the shorter of the two line segments which meet at the corner,

$$S(Q_j) = \min\{\|a\|, \|b\|\}. \tag{4.1}$$

$\alpha(Q_j)$ – the angle of the corner defined by Q_j,

$$\alpha(Q_j) = \cos^{-1}\left(\frac{a \cdot b}{\|a\|\|b\|}\right) \tag{4.2}$$

$\beta(Q_j)$ – the corner orientation, as given by the angle of the line which bisects the corner in the cone direction.[2]

The set of points $p \in \mathbf{R}^2$ on the two line segments which meet at the corner is given by

$$C_j^Q = L_{j-1}^Q \cup L_j^Q, \tag{4.3}$$

[2]We choose this definition of orientation because it gives equal weight to the direction of both vectors which define the corner.

where

$$L_j^Q = \{(1-s)q_j + sq_{j+1} : 0 \leq s \leq 1\} \tag{4.4}$$

Let S be an $M \times M$ region of the image centered about the point q_j. Suppose that disturbances such as those due to lighting or occluding objects do not occur in the region S. If other sections on the path of the polygonal boundary do not intersect S, then there should be three distinguishable regions S_{in}, S_c and S_{out} in which a ramp or roof corner model describes the image data based on some image feature measure λ. This condition can be stated more formally as follows.

Let $U \subset \mathbf{R}^2$, and let $d(v, U)$ denote the minimum distance from the point $v \in \mathbf{R}^2$ to any point $u \in U$, so

$$d(v, U) = \min_{u \in U} \|v - u\|.$$

Given a polygonal image object boundary path $Q = \{q_j\}_{j=1}^m$, if for each $q_j \in Q$, there is an $M \times M$ region S centered at q_j for which

$$d(q_j, L_k^Q) \geq \frac{M}{\sqrt{2}} \quad \text{for} \quad |k-j| > 1, \tag{4.5}$$

then in the region S, the image data should behave as a simple corner model with the geometry defined by Q_j.

Corner Geometry

Locally, the nonintersection assumption imposes restrictions on the geometry of the neighboring corners Q_{j-1} and Q_{j+1}. Intuitively stated, the positions q_{j-1} and q_{j+1} of these corners must be outside S, and the path of Q cannot turn so sharply at these points, that it passes through S on the segment between q_{j-1} and q_{j-2} or on the segment between q_{j+1} and q_{j+2}. This means the corner angles $\alpha(Q_{j-1})$ and $\alpha(Q_{j+1})$ must be wide enough so the line segments L_{j-2}^Q and L_{j+1}^Q will not intersect S. See Figure 4.3.

To guarantee q_{j-1} and q_{j+1} are outside an $M \times M$ neighborhood S centered at q_j, we need only require

$$M < \sqrt{2} \cdot S(Q_j), \tag{4.6}$$

because $M/\sqrt{2}$ is the distance from q_j to the corner points on S, and we need only assure this is smaller than $S(Q_j)$, the distance to the closer point.

Without loss of generality, let $\|a\| \leq \|b\|$, so q_{j-1} is the closer point, and the corner scale $S(Q_j) = \|a\|$, based on Equation 4.1. Influence from the corner in the image defined by the triplet Q_{j-1} in an $M \times M$ region S centered at q_j, can only occur if L_{j-2}^Q intersects the region S.[3]

Given the point q_j, if the nearest neighbor $q_{j-1} \in Q$ is positioned outside the region S, then L_{j-2}^Q will not intersect the region S if $\alpha(Q_{j-1}) \geq \alpha$, where α is the angle shown for the two extreme cases in Figures 4.4a and 4.4b. The lower bound α on $\alpha(Q_{j-1})$

[3]This condition is weak in that it neglects the width of the strip influenced by the transition along L_{j-2}^Q.

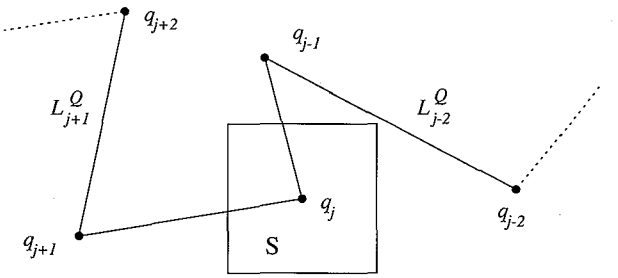


Figure 4.3: To prevent intervention in the region S with the corner model defined b $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$, the points q_{j-1} and q_{j+1} must be outside the region S, and th corner angles $\alpha(Q_{j-1})$ and $\alpha(Q_{j+1})$ must be large enough to prevent L^Q_{j-2} and L^Q_{j-} from intersecting the region S.

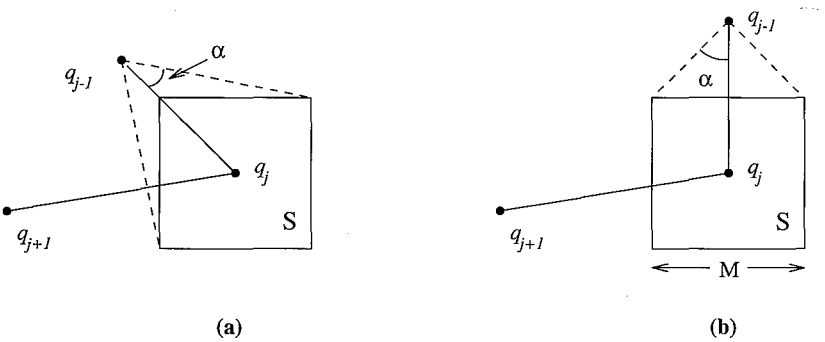


Figure 4.4: To prevent intervention in the region S with the corner model defined by $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$, the angle $\alpha(Q_{j-1})$ of the corner at q_{j-1} cannot be smaller than the angle α shown in the two figures. It is clear that if the line segment from q_{j-1} to q_j is vertical (or horizontal), the lower bound α on the corner angle $\alpha(Q_{j-1})$ is maximal. Further as the distance $d(q_{j-1}, S)$ decreases, the lower bound increases.

depends on the distance $d(q_{j-1}, S)$ from the point q_{j-1} to the region S and on the direction of the vector $a = q_{j-1} - q_j$. In Figure 4.4, it is easy to see that if the $\|a\|$ is fixed, the lower bound α increases as the vector a becomes vertical (or horizontal).

Assuming $a = q_{j-1} - q_j$ is vertical (worst case), given the distance $\|a\|$ from q_{j-1} to q_j, we can express the lower bound on α as a function of the width M of the region S.

$$\alpha(M) = \tan^{-1}\left(\frac{M/2}{S(Q_j) - M/2}\right) \tag{4.7}$$

Note that the restriction in Equation 4.6 guarantees that the denominator is nonzero.

This condition can be met for all corners $Q_j \in Q$ if we use

$$M \leq S(Q_j), \tag{4.8}$$

and if for all $Q_j \in Q$,

$$\alpha(Q_j) \geq \pi/4. \tag{4.9}$$

where $S(Q_j)$ is the corner scale as defined in Equation 4.1, and $\alpha(Q_j)$ is the corner angle defined in Equation 4.2.

Regions in S

Given an $M \times M$ region S centered at a point q_j, let $S_c \subset S$ be the set of points influenced by the transition along the path of Q in S as measured by λ. The geometry of the region S_c depends on the geometry of the corner defined by the line segments L^Q_{j-1} and L^Q_j and on the point spread function. If the image boundary fits a roof model, then the width of the line or roof will also influence the geometry of S_c. Finally, for any given image feature measure λ, the geometry of S_c depends on the radius of influence of the measure λ. For example, if the value of λ at a point p is defined as a function of the image data in a $k \times l$ window centered at p, then the radius of influence is $r(\lambda) = \sqrt{k^2 + l^2}/2$.

Given a point $q \in C^Q_j \cap S$ on the object boundary in S (Equation 4.3), let $W(q)$ be the minimum distance to any point $p \in S$ which is not influenced by the presence of the boundary on S. Then if we let

$$W = \max\{W(q) : q \in C^Q_j \cap S\}, \tag{4.10}$$

all points $p \in S$ which satisfy $d(p, C^Q_j) \geq W$ are not influenced by the image function along the path C^Q_j of the object boundary in S.

We thus define the transition region S_c as the set of points $p \in S$ within distance W of C^Q_j. Formally,

$$S_c = \{p \in \mathbf{R}^2 : p \in S \text{ and } d(p, C^Q_j) \leq W\}. \tag{4.11}$$

If Equations 4.9 and 4.8 are satisfied, and if $W < M/4$, the remaining members $p \in S \backslash S_c$ of the set S fall in two disconnected regions S_{in} and S_{out} on either side of

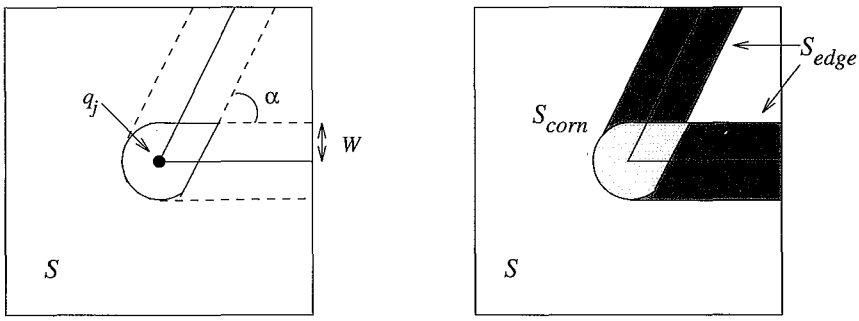


Figure 4.5: The behavior of the measure λ in the immediate region about the path of Q is different for those $p \in S_c$ within distance r of the corner point q_j, from those further away ($\|p - q_j\| > r$) where the transition should be well modeled by either a ramp or roof edge model.

the region S_c, and we have

$$S_{in} \cup S_{out} = S \backslash S_c = \{p : p \in S \text{ and } p \notin S_c\}. \tag{4.12}$$

In both the ramp and roof corner model, there exist constants c_{in}, c_{out} and c_L, such that:

$$\lambda(x) = \begin{cases} c_L & \text{if } x \in C_j^Q \\ c_{in} & \text{if } x \in S_{in} \\ c_{out} & \text{if } x \in S_{out} \end{cases} \tag{4.13}$$

The behavior of λ on $S_c \backslash C_j^Q$, the region influenced by the boundary between the object and its background, but not on the boundary itself, will differ for two regions on S_c. In the region immediately about the point q_j (see Figure 4.5), the value of λ will be influenced both by the transition along the path from q_j to q_{j-1} and from the transition along the path from q_j to q_{j+1}. This is because many of the points in that region are within distance W from both of these edges. Outside the immediate neighborhood of the corner, it will only be influenced by the transition along one of these line segments, and the behavior of λ in these regions will fit a simple ramp or roof model described below.

Consider Figure 4.5. Given the width $2W$ of S_c, the region influenced by the object to background transition, points in the set

$$S_{corn} = \{p \in S_c : d(p, L_{j-1}^Q) \leq W \quad \text{and} \quad d(p, L_j^Q) \leq W\},$$

may be influenced both by the transition in the image function on L_{j-1} and in the transition on L_j^Q.

Then the set of points not in S_{corn} will be influenced by the transition along only

one of the line segments which meet at q_j. We can define this set simply as

$$S_{edge} = \{p \in S_c : p \notin S_{corn}\}. \tag{4.14}$$

The image function in the S_{edge} region should behave as a simple edge or roof model, and be uninfluenced by the presence of the corner point. We can thus define ramp and roof image corner models as follows.

The ramp model A corner is described by a ramp corner model, when in addition to Equation 4.13, one of the following two criteria is met:

1. $c_{out} > c_{in}$, $c_L = (c_{out}+c_{in})/2$, and in the region S_{edge}, λ is a strictly monotonically decreasing function of the distance to the set S_{out}. More precisely stated, for all $p, q \in S_{edge}$, $c_{in} \leq \lambda(p), \lambda(q) \leq c_{out}$, and

$$d(p, S_{out}) > d(q, S_{out}) \Longrightarrow \lambda(p) < \lambda(q) \tag{4.15}$$

2. $c_{out} < c_{in}$, $c_L = (c_{out}+c_{in})/2$, and in the region S_{edge}, λ is a strictly monotonically increasing function of the distance to the set S_{out}, or formally, for all $p, q \in S_{edge}$, $c_{in} \geq \lambda(p), \lambda(q) \geq c_{out}$, and

$$d(p, S_{out}) > d(q, S_{out}) \Longrightarrow \lambda(p) > \lambda(q) \tag{4.16}$$

The roof model A corner is described by a roof corner model, when in addition to Equation 4.13, one of the following two criteria is met:

3. $c_{in} = c_{out}$, $c_L > c_{out}$, and for all $p \in S_{edge}$, λ is a monotonically decreasing function of the distance to the set C_j^Q. Formally, for all $p, q \in S_{edge}$,

$$d(p, C_j^Q) > d(q, C_j^Q) \Longrightarrow \lambda(p) \leq \lambda(q) \tag{4.17}$$

4. $c_{in} = c_{out}$, $c_L < c_{out}$, and for all $p \in S_{edge}$, λ is a monotonically increasing function of the distance to the set C_j^Q. Formally, for all $p, q \in S_{edge}$,

$$d(p, C_j^Q) > d(q, C_j^Q) \Longrightarrow \lambda(p) \geq \lambda(q) \tag{4.18}$$

Note that the use of $\leq$ and $\geq$ in Equations 4.17 and 4.18, allows a wide line or roof corner to be modeled with a roof corner model.

4.3.2 The User Corner Model

Suppose a user produces a polygonal point set $P = \{p_i\}_{i=1}^n$ in the course of sketching an image object boundary defined by $Q = \{q_j\}_{j=1}^m$. In particular, suppose the corner triplet $P_i = \{p_{i-1}, p_i, p_{i+1}\}$ corresponds to the corner defined by the triplet $Q_j = \{q_{j-1}, q_j, q_{j+1}\}$ on the object boundary. Let R be an $N \times N$ region centered about the

point p_i. Based on the user error model obtained in Chapter 2, we can define a region $R_c \subset R$ which contains all points on the boundary path Q in the region R. Given the image corner model derived above, we can in fact define the regions R and R_c so that $S_c \cap R \subset R_c$, where S_c is the region influenced by the change in the image function in the transition from object to background.

This results in a division of the region R, much like that shown in Figure 4.2. $R \backslash R_c$, the remainder of R, is made up of two disjoint regions R_{in} and R_{out} which correspond to the image regions S_{in} and S_{out} described above. In fact, if

$$R \subset S \quad \text{and} \quad S_c \cap R \subset R_c, \tag{4.19}$$

then $R_{in} \cap S \subset S_{in}$ and $R_{out} \cap S \subset S_{out}$.

If R and R_c are so defined, then the regions R_{in}, R_c and R_{out} can be used to estimate the behavior of the image feature measure λ in the object, transition, and background regions[4] S_{in}, S_c and S_{out} respectively. Our task is therefore to define the region R_c in terms of the user error model and the image corner model so that Equation 4.19 holds. To model the image boundary corner defined by Q_j, however, we must be sure the regions R_{in} and R_{out} are large enough to estimate the behavior of λ in that region.

The Region R_c

Given a triplet $P_i = \{p_{i-1}, p_i, p_{i+1}\}$, let R be an $N \times N$ subimage centered about the point p_i. Let ε be the maximum likely error in distance between a point $p_i \in P$ and its corresponding point $q_j \in Q$. Based on Equation 2.19 derived in Chapter 2, we know that for each individual user, an appropriate value of ε can be determined. We use this value as the maximum permitted error in distance for a given user.

Let δS denote the maximum amount the user may *overestimate* the corner scale. Then the scale of the corner Q_j on the image object boundary satisfies

$$S(Q_j) \geq S(P_i) - \delta S. \tag{4.20}$$

A user dependent upper bound on δS was derived in Section 2.5 (Equation 2.23).

Suppose S is an $M \times M$ region centered at the corner point q_j on the image object boundary. If $M = S(Q_j)$, then S is the largest square region guaranteed to contain a nondisturbed model of the corner defined by the triplet Q_j, assuming Equations 4.5 and 4.9 hold.

Since $\|p_i - q_j\| \leq \varepsilon$, if the size N of the region R centered at p_i satisfies

$$N \leq M - 2 \cdot \varepsilon$$

then we have $R \subset S$. Therefore, given $S(P_i)$, based on Equation 4.20, we define R to

[4]This labeling of the regions in S has been chosen arbitrarily to simplify the discussion. Of course S_{in} might correspond to the background region, S_{out} to the object region, etc.

be an $N \times N$ region centered at p_i with

$$N = S(P_i) - \delta S - 2 \cdot \varepsilon. \tag{4.21}$$

Given W, as defined in Equation 4.10, we define

$$R_c = \{p \in R : d(p, C_i^P) \leq \varepsilon + W\}, \tag{4.22}$$

where C_i^P is defined analogously to C_j^Q in Equation 4.3. If the user error model and the image corner model are valid, then with this definition of R_c, we have $S_c \cap R \subset R_c$.

The Region R_{in}

Having defined R_c as the minimum neighborhood of the path P in R known to contain S_c, consider the remainder of the region R,

$$R \backslash R_c = \{p \in R : p \notin R_c\}. \tag{4.23}$$

If the size N of R is sufficiently large, then there will be two nonempty disjoint regions R_{in} and R_{out} such that $R \backslash R_c = R_{in} \cup R_{out}$.

Suppose there is an $M \times M$ region S centered about the corner point q_j, in which the image object boundary fits a ramp or roof corner model, as defined in the previous section. If the size M of S is the largest for which we know there can be a clear corner model, then from Equation 4.8, $M = S(Q_j)$. If N satisfies Equation 4.21, but is large enough so that R_{in} is nonempty, then

$$R_{in} \subset S_{in} \quad \text{and} \quad R_{out} \subset S_{out}.$$

If R_{in} is sufficiently large, then R_{in} and R_{out} can be used to model the behavior of λ on the object and background regions in the image.

We thus turn our attention to the area of the smaller region R_{in}. We want to specify a set of conditions under which we can be sure the area $A(R_{in})$ of the smaller region will be greater than some lower bound $B(\lambda)$, where $B(\lambda)$ is the smallest region in which we can hope to model the behavior of λ.

Consider Figure 4.6a. The area of R_{in}, decreases with $\alpha(P_i)$, the angle of the corner at p_i. We thus limit our attention to corners P_i with angle $\alpha(P_i) < \pi/2$. Further, it is clear that the area of R_{in} is minimal for corners which are horizontally or vertically oriented. So the horizontal case is sufficient to determine the lower bound of the area $A(R_{in})$ for a given angle α.

To simplify the discussion, suppose $p_i = (0, 0)$, then from Figure 4.6b, it is easy to see that

$$A(R_{in}) = (N/2 - \Delta x)(y(p_b) - \Delta y) \tag{4.24}$$

where $y(p_b)$ is the y coordinate of the point on L_i^P with $x = N/2$, and

$$\Delta x = \frac{W + \varepsilon}{\sin[\alpha(P_i)/2]} \quad \text{and} \quad \Delta y = \frac{W + \varepsilon}{\sin[(\pi - \alpha(P_i))/2]}. \tag{4.25}$$

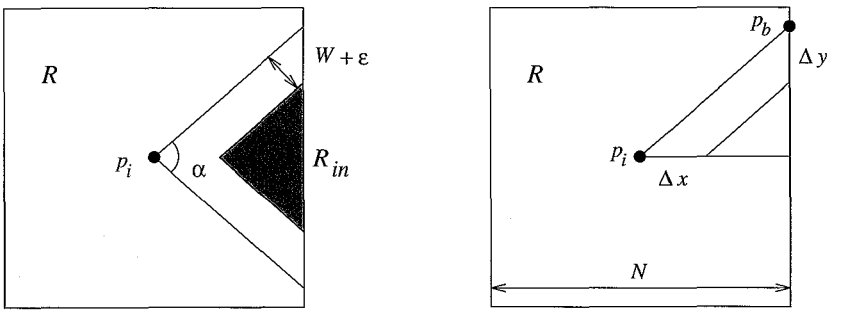


Figure 4.6: For a given corner angle $\alpha < \pi/2$, the area of the inner region R_{in} is smallest for a horizontally (or vertically) oriented corner P_i. The area can be computed in terms of the user point p_i, the position p_b of the boundary intersection on the line segment L_i^P, and Δx and Δy. These in turn are determined by $\alpha = \alpha(P_i)$ and $W + \varepsilon$.

For a given user and image measure λ, $W + \varepsilon$ is fixed. If a lower bound $B(\lambda)$ on the area necessary to measure λ is known, then we can specify the set of corners which can be modeled by defining and splitting a region R in terms of the minimum corner scale $S(Q_j)$, and the minimum corner angle $\alpha(Q_j)$. The latter can be predicted $(\alpha(Q_j) \geq \alpha(P_i) - \delta\alpha)$ based on Equation 2.20 of the user error model in Section 2.5.

4.3.3 Evaluating λ

Suppose given a user-defined corner P_i, we want to evaluate the likelihood of a ramp or roof corner Q_j on the image object boundary based on some image feature measure λ. If we construct and divide an image region R as suggested in the previous section, we can estimate parameters for the measure λ in the three regions of R. Because the region R is divided so that $R_{in} \subset S_{in}$ and $R_{out} \subset S_{out}$, parameters estimated in R_{in} and R_{out} are representative of the image feature measure λ in the corresponding regions S_{in} and S_{out} in the neighborhood S, which is centered at the true location q_j of the corner on the image object boundary. Because $S_c \subset R_c$, however, only part of the data which contributes to parameters estimated in the central region is known to come from the boundary region of S. Still, if there is a corner Q_j near P_i which can be detected with λ, the data in R_c should satisfy certain conditions associated with the ramp and roof corner models described in Section 4.3.1.

The actual parameters to be estimated for a given image feature measure λ depend on the statistical model associated with it. As an example, suppose λ is a simple measure of image intensity, and that the image is known to contain added Gaussian noise with a maximum deviation of σ_{noise}. The image function in R_{in} and R_{out} should fit the model $f_N(p; \mu_{in}, \sigma_{in})$ and $f_N(p; \mu_{out}, \sigma_{out})$, respectively, where $f_N(p; \mu, \sigma)$ denotes

a Gaussian density function with mean value μ and standard deviation σ. If image intensity is a good measure for a ramp or roof corner, then the triplet $\{\mu_{in}, \mu_c, \mu_{out}\}$ should satisfy one of the four conditions outlined in Section 4.3.1. Further, unless there is disturbance to the model from another image object in the region, or there is a poor fit, then we should have

$$\sigma_{in} \approx \sigma_{out} < \sigma_{noise}.$$

Higher deviations cannot be attributed simply to the presence of noise.

For more complex functions λ, the expected distribution may not be so easily evaluated. In this case, various parameter estimation methods may be applied, or the Kolmogorov-Smirnov test [vM64] may be applied to examine the significance of difference between the data sets in R_{in} and R_{out} when measured by λ (see Section 2.5).

4.3.4 The Corner Detection Model

Given a corner P_i on a user defined polygonal sketch P of an image object boundary, suppose we have an image feature measure λ which, based on measurements in an $N \times N$ region R centered at P, appears to fit either a ramp or roof model as described in Section 4.3.1. Then, based on Equation 4.13, there will be three constants λ_{in}, λ_c, and λ_{out}, which may be associated with the image corner regions S_{in}, S_c, and S_{out} respectively. If for a given user, the user error model is expressed in a set $U = \{\varepsilon, \delta\alpha, \delta\beta, \delta S\}$, then the set

$$D = \{P_i, \lambda, U\} \tag{4.26}$$

determines a *detection model* for the corner Q_j on the image object boundary. As is illustrated in the following section, the detection model can be used to locate the corner position q_i in the image.

4.4 Example: Constructing a Corner Template

Suppose we have identified a detection model D for a particular corner P_i in a user sketch P. One method to localize the true corner point q_j in the image is to construct a template from the derived corner model, and match it in the neighborhood of p_i using standard matching techniques. The geometry of P_i combined with the characteristic values λ_{in}, λ_c, and λ_{out} can be used to define a template which should match the image data near q_j when measured by λ. In the example in Section 4.3.3, where λ simply measured grey value intensity, μ_{in}, μ_c and μ_{out} are the characteristic values for λ.

Let

$$a_p = p_{i-1} - p_i \quad \text{and} \quad b_p = p_{i+1} - p_i \tag{4.27}$$

be the vectors defined by the line segments which meet at the point p_i in the user sketch. We can localize the corner point q_j if we define a $K \times K$ digital template T centered at the origin.

Let T_c be all points in T which fall on a Bresenham approximation to a_p or b_p. In

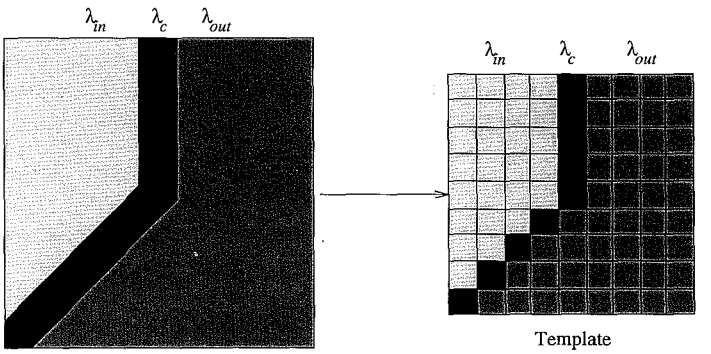


Figure 4.7: Designing a template based on a detection model

the template T, we assign the value λ_c to these points. T_c, like its counterparts R_c and S_c splits the remainder of the template T into two disjoint regions

$$T \backslash T_c = T_{in} \cup T_{out}.$$

To these regions we assign the constants λ_{in} and λ_{out} respectively. See Figure 4.7.

Given ε, the maximum error in distance allowed for this user in the specification of the point p_i, we define $\varepsilon_{max} = \lceil \varepsilon \rceil$. The point q_j is then sought in the n^2 region centered at p_i with $n = 2 \cdot \varepsilon_{max} + 1$. In practice, this allows a user to make a slightly larger error in the specification of any single point than was determined likely in the user experiments. For all pixels in this n^2 neighborhood centered at p_i, we measure the value of the normalized cross correlation $C(m, n)$ between the template T and the image data $\lambda(x, y)$, as defined by Rosenfeld and Kak[RK76, p. 302]. We select the pixel with the maximum response as the corner location.

Note that the actual values of the parameters in the template is not essential for this matching method. As long as we have been able to identify one of the four image corner models described in Section 4.3.1, the parameters can be replaced with members of $\{-1, 0, +1\}$ for the template matching procedure.

4.5 Results

To evaluate the effectiveness of our method, we performed a number of experiments designed to evaluate:

- The accuracy of the image evaluation method in identifying the presence of ramp and roof corners.
- The effectiveness of the template matching to localize the corner when the appropriate corner model (roof or ramp) has been identified.

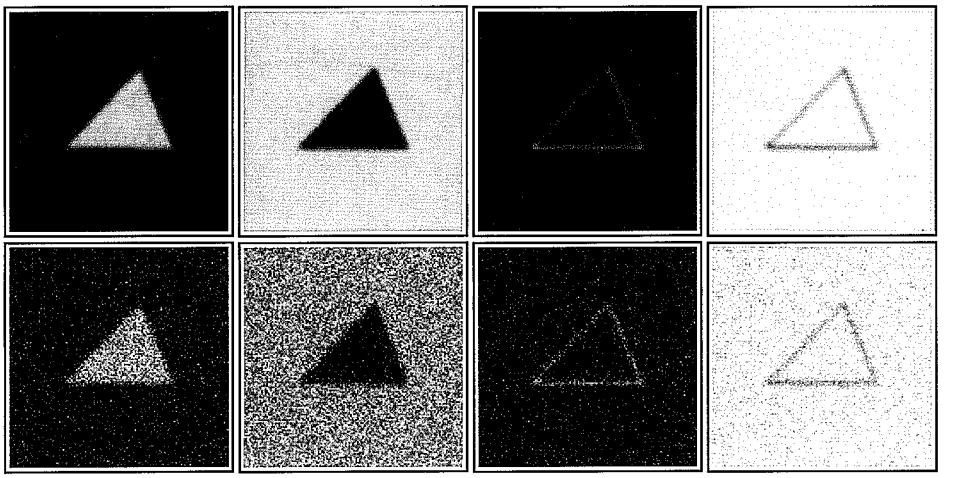


Figure 4.8: Examples of the test images, without noise and with SNR = 5.

Test Images
We performed our evaluation on a set of test images containing ramp corners and roof corners. These were created by quantizing filled triangles for ramp corners and two lines which meet at a known point for roof corners. The edges of the filled triangle are assigned the value half way between the object and boundary values to improve our estimation of the true corner location. We did this for corner angles ranging from $\pi/8$ to $3\pi/8$, and performed our experiments both for dark objects on a bright background and vice versa. In order to simulate a ramp or roof edge which may be formed by a camera, we applied a Gaussian smoothing with $\sigma = 1.5$ to the images.

Finally, for each of the images, we tested our method with added white Gaussian noise, with signal to noise ratios ranging from 5 to 20. We use

$$SNR = \frac{\{(min - max)/2\}^2}{\sigma^2_{noise}}$$

to compute the signal to noise ratio. Some example images are shown in Figure 5.

Experiments Our experiments are intended to simulate a user specifying a corner in an image. Given three points which define the corner in the test image, we select a random point in an n^2 neighborhood about each, where $n = 2 \cdot \varepsilon_{max} + 1$, with $\varepsilon_{max} = \lceil \varepsilon \rceil$.

The length of the lines used to define the corners in our experiments is 80. There was no significant variation in the results as the length was varied until it became quite short with respect to ε, as predicted in Equations 4.24 and 4.25. In the experiments presented here, we use $\varepsilon_{max} = 8$ which is far larger than necessary in practice (see

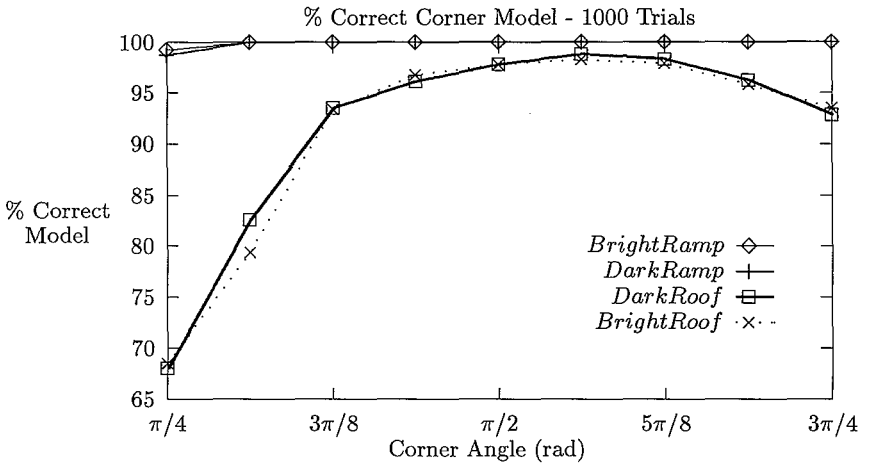


Figure 4.9: Percent of time correct segmentation model is selected, $SNR = 10$.

Chapter 2). The size of the region in which the corner point will be selected is therefore 17×17. Because this is the degree of error the user may make in specifying each of the points which define the contour, the random points in our experiment are selected from a 17×17 neighborhood around each of the three points which correctly define the corner. Note that using random points introduces an error not only in the position of the points, but a significant error in the angle and orientation of the corner. Therefore, the geometrical errors allowed in our experiments are far worse than users make in practice (see Chapter 2).

For each of the test images, we select 1000 sets of three random points to which our algorithm is applied. We consider all points in the n^2 neighborhood of the random point near the corner point, to be candidate points. The template used to model the corner is 11×11.

In Figure 4.9, we show the percentage of cases we correctly select the detection model with $SNR = 10$. Note that both dark and bright ramp corners are correctly identified in 100% of the cases at all but the smallest corner scale ($\pi/4$). Roof corners are correctly identified in the majority of the cases (more than 90%), but performance deteriorates severely for very small corners.

In Figure 4.10, we present the average distance to the true corner location selected by our method when the correct model has been selected. Of course the *true* corner location is not well defined on a discrete grid. To simplify our experiments, we consider it to be the corner location in the filled triangle which is always set to the object value, or for ramp corners to the value halfway between the object and background values. A number of points in the 8-connected neighborhood are however also good candidates.

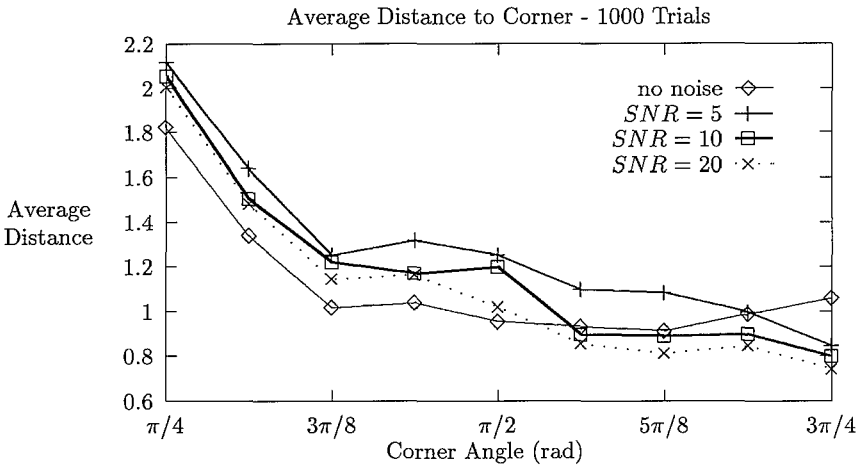


Figure 4.10: Average distance to correct corner location.

For this reason, in Figure 4.11, we present the percentage of the points selected that lie in this neighborhood.

Finally, in Figure 4.12, we show the adjustment of some contours specified by a user in two images.

Evaluation The results in Figure 4.9 show that we detect the appropriate model consistently both for dark and bright objects, with corner angles $\theta > \pi/4$ used in these experiments. The method is particularly effective for correct identification of both bright and dark ramp corners.

Because we allowed a substantial error in distance ($\varepsilon_{max} = 8$), roof corners were harder to identify than ramps. This is because of the substantial influence of the background regions in the transition region. The data along the path of the roof in the image gets averaged out.

In these experiments, identifying the corner model for small corners is more difficult due to the increased effect of the angle variation in the random point selection, resulting in the collapse of the region $R_{in} \subset R$.

When the corner model is correctly identified, the difference in line angles in the template and the image is severe, resulting in poorer localization for small angle corners. Localization mechanisms which incorporate the user angle errors derived in Chapter 2 are more suitable in this case. These effects are less significant for larger corners due to the increased influence from the inner corner region.

Figures 4.10 and 4.11 show the precision of the template we build from our detection model to be quite satisfactory in the range $(3\pi/8, 3\pi/4)$. There was little variation in the results for the ramp and roof corners.

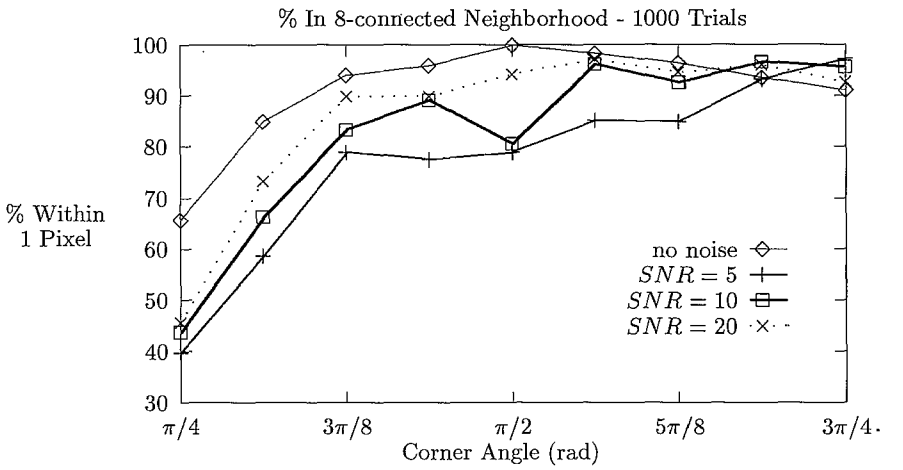


Figure 4.11: Percent of locations selected in the 8-connected neighborhood of the correct location.

In further experiments not displayed in the graphs, we discovered the segmentation model can be determined for larger corners, but the corner is not well localized due to the response of nearby edge points. This implies that for larger corners, template matching is not an appropriate localization technique.

4.6 Conclusions & Further Research

We have presented a context in which the segmentation model appropriate to an image region can be determined, given a geometrical model of a corner on a polygonal object boundary. For corner angles in the range $(3\pi/8, 3\pi/4)$, the method identifies 100% of both dark and bright ramp corners, and more than 90% of the roof corners. The simple template design and matching procedure used to localize a corners is effective at pixel accuracy for the same range of angles, but other mechanisms should be employed to localize very small and very wide angled corners.

The approach to corner modeling developed in this chapter can be applied to any image feature measure λ with a limited radius of influence. The results suggest we may extract significant information about a segmentation model from a simple user sketch. The results for wide angle corners also indicate that the method can be used to determine the appropriate segmentation model for edges and curves.

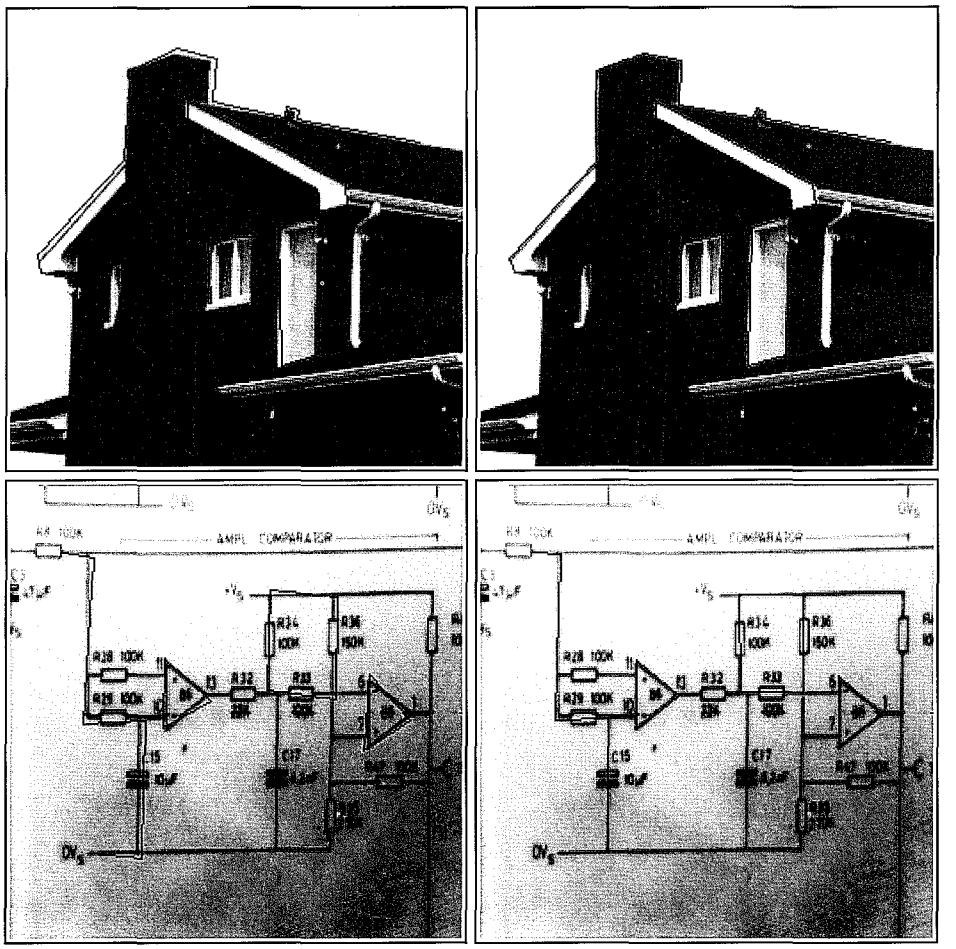


Figure 4.12: Examples: On the left, user input. On the right, the corrected contour.

Chapter 5

Magnetic Contour Tracing

5.1 Introduction [1]

Because the segmentation of unknown images remains a critical unsolved problem in image analysis, it is often useful to allow a human expert to influence the segmentation process interactively. Discontinuities in an image due to significant object boundaries can be distinguished by a human expert from those due, for instance, to lighting conditions such as shadows and reflections, or to boundaries of image objects which are not of interest for the application at hand. Further, a human is often able to *continue* an object boundary through areas where it is not visible, and therefore not detectable using solely automatic means. Such areas will occur, for example, whenever an object of interest is partially occluded by another image object, such as a house behind a tree. To date, no machine vision system has been developed which matches the human capability to perform virtually unlimited image segmentation.

Due to limitations in human motor control and the indirect methods available for screen location specification (see for example [HHN86]), the data obtained from a human expert cannot be considered accurate. Thus, while the user may provide information suggesting the location and geometrical properties of an image object boundary, measurements based on this data are subject to an unknown error. Before object analysis which depends on the boundary can be performed, the user data must somehow be corrected to provide accurate object boundary descriptions.

In the area of graphical user interfaces, much progress has been made in the development of techniques to aid the user in the specification of precise screen locations, or to correct a screen location specified by a user according to some predefined rules. By far the best known is the *grid* technique used in packages such as MacDraw[Cla92], which allows a user to precisely line up objects along horizontal or vertical lines. *Constraint-based* methods [Bor81, Sut63] were introduced to assist in the precise definition of a broader range of geometric forms and relationships. For example, constraints can be used for specifying an equilateral triangle, or to line up objects along the path of an arbitrary curve. Recent efforts such as the *snap-dragging* techniques

[1] Portions of this chapter are reprinted, with permission, from [OG94] ©1994 IEEE.

introduced in [BS86, Bie90], and the techniques for interactive constraint specification discussed in [Bor86] have aimed to reduce the effort required of the user to specify a broad range of geometrical forms and relationships between objects.

The central goal of grid and constraint based methods is to make certain screen locations *attract* the mouse pointer, while others repel it. The result is that attractive locations act as gravitational masses for screen location specification. Hudson[Hud90] introduced the concept of *semantic snapping* techniques to make particular screen locations attractive based on nongeometric application semantics. For example, within the context of visual programming, an icon representing a function may have connection points for input parameters and return values. If a connection is initiated from some function A, which returns a valid type for input to function B, the input connection on B's icon will be made attractive when the connection is initiated. If, on the other hand, B requires input of another type, the input connection will be made repellent when a connection is initiated at A.

If the concept of semantic snapping is applied to the problem of image segmentation, and in particular to the problem of image object boundary formation, then locations along the boundary of an object should be made more attractive than those "inside" it. For a significant body of images, the magnitude of the gradient of the image intensity function can be used as a measure for boundary strength. In general, if the region corresponding to an image object differs from neighboring regions in grey value intensity, then the gradient magnitude will be high along the boundary of the object, and low on either side of it. The gradient magnitude can then be used as a measure for the strength of the boundary, and therefore to determine how attractive an image location is.

In the context of *active contours*, Kass, Witkin and Terzopoulos [KWT88] incorporate user input in their method for obtaining an optimal object boundary based on a number of criteria. Criteria designed to embody the characteristics of an image object boundary are expressed as components of an energy functional and simultaneously minimized using calculus of variations. Their criteria are as follows: a) the distance between points along the path must be minimal to encourage continuity; b) the curvature at each point must be minimal to encourage smoothness; and c) the average gradient magnitude along the path should be maximal so it will tend to follow the image object boundary.

The user can make use of tools for pulling and pushing the contour, resulting in it jumping between local minima in the functional. In this sense it is similar to a tool which jumps to the nearest point on a grid. This is achieved by incorporating a term in the functional which either minimizes or maximizes the distance to the user location, thereby resulting in a pulling or pushing of the boundary. The method may be considered a semantic snapping technique in which the semantics are a combination of the expected geometrical (continuity and smoothness) and nongeometrical (magnitude of the image intensity gradient) properties of an object boundary.

The method introduced in [KWT88] can play an important role in correcting an object boundary. Some form of pre-segmentation, which produces an initial estimate of the object boundary is necessary, however, before the method can be applied. The

tools for pulling and pushing a boundary may therefore be seen as editing tools for an already existent boundary path.

As illustrated by Amini, Weymouth and Jain in [AWJ90], the continuity criterion (a) above) as posed in [KWT88] makes shorter paths more attractive and results in large variations in the distance between the points which define the contour. Further they show calculus of variations to be an instable optimization technique, because the solution produced often corresponds to a local rather than a global minimum of the functional. They suggest dynamic programming be applied to optimize the functional, and show that it addresses both problems. This is because it allows the incorporation of *hard* constraints which can be used to regulate the distance between path points, and because dynamic programming always produces a global minimum (or maximum), and thus provides a stable solution given the criteria used to define the problem. In [Ger88], Gerbrands also used a dynamic programming algorithm to extract the path of an object boundary given an initial estimation (see also Section 5.3.3).

In this chapter, we introduce a method for tracing an object boundary when no a priori estimate of its path is available. Rather than correcting or editing an initial estimate of a boundary path, we produce a path from scratch based on user input. We allow a user to trace a contour by means of *freehand-drawing*. As supported in most graphical drawing packages, freehand drawing is very similar to drawing with a pen on paper, and thus is an intuitively attractive input model for the task of image object boundary specification. Depending on the underlying interaction model supported in the system, the user's screen location may be known at regular or irregular intervals in terms of either space or time. To simulate a pen on paper, if the user's location is not known frequently enough to define a connected path, straight lines are interpolated between points.

If we incorporate the concept of semantic snapping in the free-hand drawing model, then we may envision a pen containing magnetic ink. As the user traces a path in the image, the ink is attracted to those locations near the tip of the pen which are most attractive. For image object boundary specification, we want locations along image object boundaries to be highly attractive, and those away from the boundary to be somehow repellent. As with active contours [KWT88], the contour must remain both connected and smooth unless otherwise indicated by the user. We present a method called *magnetic contour tracing* which feels like a freehand drawing tool, but satisfies constraints based on the semantics of image object boundary formation.

Our method generates the correct path of the object boundary in real time as the user traces. To do so, we examine the relationship of the user path with the correct boundary path. The boundary path is then produced by incorporating this relationship and the semantics of image object boundary formation in a particularly efficient variation on Gerbrands' dynamic programming method [Ger88], which is guaranteed to produce an optimal path with respect to these criteria.

Extracting the correct path of the object boundary as it is traced rather than as a postprocessing correction is well justified from the viewpoint of human computer interaction, because it results in an interactive method which satisfies the requirements of *direct manipulation* as outlined in [Shn83]. The feeling of direct control over the ob-

ject of interest[HHN86] (in this case an object contour), which results from immediate feedback on the effects of a user's action, is of particular importance in this regard.

The magnetic ink paradigm supported in the contour tracing tool is achieved by interpreting the path traced by the user as an estimation of the position and direction of the image object boundary. Based on the user input, in Section 5.2, we determine the set of 8-connected paths through the region which may be part of the object boundary. These criteria are balanced with those related to the magnitude of the gradient in a cost function (Section 5.3.2), and the path selected is that with the minimum total costs as determined with a dynamic programming algorithm (Section 5.3.1).

Our method produces a smooth path which follows the user closely in regions with little or no contrast and the object boundary closely in areas where the magnitude of the gradient is significant. Achieving this in the presence of noise requires an optimal balance in the weighting parameters used in the cost function. Experiments used to tune these parameters for specific conditions are described in Sections 5.4 and 5.5. The tuning method may easily be applied to user data, making the method suitable for incorporation in an adaptive user interface (see for example [KDMSH92]), for image segmentation.

Quantitative results obtained for a simulated user using random data are presented in Section 5.6 as are qualitative results for an actual user tracing boundaries in actual images.

5.2 Theory

We describe a method with which we extract a digital approximation to an image object boundary while it is traced by the user. The path is extracted in a piecewise fashion, by repeatedly finding a minimum cost path segment from the last known point in the approximation to some point near the most recent user location.

Before describing our algorithm in depth, we provide a theoretical argument for our approach. In particular, we focus on the class of paths $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ for which the method is applicable, and the constraints which must be satisfied by the user input data if it is to lead to a digital approximation of the boundary path $\boldsymbol{\rho}$.

5.2.1 Background

Definition 5.1 *A path* $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ *is called* $\boldsymbol{C}^n$ *if it is continuous on* $[a,b]$*, and the nth partial derivatives of* $\boldsymbol{\rho}$ *exist and are continuous on* (a,b)*.*

Definition 5.2 *A path* $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ *is* **non-intersecting** *if for all* $t_0, t_1 \in (a,b)$*,* $\boldsymbol{\rho}(t_0) = \boldsymbol{\rho}(t_1)$ *implies* $t_0 = t_1$*.*

A non-intersecting path is one which does not cross itself. The path of the digit 2 is non-intersecting whereas the path of the digit 8 is not. The path of the digits 0 and 6 are both non-intersecting, because Definition 5.2 does not restrict the path at its endpoints $\boldsymbol{\rho}(a)$ and $\boldsymbol{\rho}(b)$.

Definition 5.3 *A C^2 path $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ is* **closed** *if its endpoints $\boldsymbol{\rho}(a)$ and $\boldsymbol{\rho}(b)$ satisfy the following conditions:*

$$\begin{aligned} \boldsymbol{\rho}(a) &= \boldsymbol{\rho}(b) \\ \boldsymbol{\rho}'(a) &= \boldsymbol{\rho}'(b) \\ \boldsymbol{\rho}''(a) &= \boldsymbol{\rho}''(b), \end{aligned}$$

A C^2 path $\boldsymbol{\rho} : [a,b] \longrightarrow \mathbf{R}^2$ which is not closed is **open.**

Because $\boldsymbol{\rho}$ is defined only on $[a,b]$, the derivatives in the above definition should of course be understood as one sided limits. For example, $\boldsymbol{\rho}'(a) = \lim_{t \downarrow a} \boldsymbol{\rho}'(t)$.

Definition 5.4 *A non-intersecting open C^2 path $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ is* **non-interfering with respect to distance δ** *if for all $t_0 \in (a,b)$, the two points p_δ on the principal normal vector to the path $\boldsymbol{\rho}$ at the point $\boldsymbol{\rho}(t_0)$ which satisfy*

$$\| p_\delta - \boldsymbol{\rho}(t_0) \| = \delta \tag{5.1}$$

also satisfy

$$\| p_\delta - \boldsymbol{\rho}(t) \| > \delta \text{ for all } t \in [a,b], t \neq t_0.$$

A non-intersecting closed C^2 path $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ is **non-interfering with respect to distance δ** *if the above conditions are also satisfied for $t_0 = a$ and $t_0 = b$.*

For all $t_0 \in (a,b)$, the existence of the principle normal vector to $\boldsymbol{\rho}$ at $\boldsymbol{\rho}(t_0)$ is guaranteed by the existence ($\boldsymbol{\rho}$ is C^2) and uniqueness ($\boldsymbol{\rho}$ is non-intersecting) of the tangent vector to $\boldsymbol{\rho}$ at $\boldsymbol{\rho}(t_0)$. Thus, for all $t_0 \in (a,b)$, there exist exactly two points p_δ on the principle normal vector to $\boldsymbol{\rho}$ at $\boldsymbol{\rho}(t_0)$ which satisfy Equation 5.1. By Definition 5.3, the same holds for all $t_0 \in [a,b]$ if $\boldsymbol{\rho}$ is closed.

Intuitively, a path $\boldsymbol{\rho}$ which is noninterfering with respect to δ is one which may be traced with a pen of width $w \leq 2\delta$, without effecting its shape. A straight line between any two points is noninterfering with respect to δ for all $\delta > 0$. For all $\delta \in (0,r)$, an arc of a circle of radius r is noninterfering with respect to δ. A path $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ which is noninterfering with respect to δ has curvature $\kappa(t) \leq 1/\delta$ for all $t \in (a,b)$ (see Struik [Str50, pp 14-15]).

Definition 5.4 could also have been expressed in terms of mathematical morphology (see [Ser82]). Let $B_\delta = \{p \in \mathbf{R}^2 : \| p \| \leq \delta/2\}$ be a disc of diameter δ centered at the origin. The *dilation* of a set $P \subset \mathbf{R}^2$ with B_δ is given by

$$P \oplus B_\delta = \{p \in \mathbf{R}^2 : \| p - x \| \leq \delta \text{ for some } x \in X\}$$

Its complement, the *erosion* of a set $X \subset \mathbf{R}^2$ with B_δ is given by

$$P \ominus B_\delta = \{p \in \mathbf{R}^2 : \| p - x \| \geq \delta \text{ for some } x \notin X\}$$

Now, if P is the set of points on a path $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ which is noninterfering with respect to δ

$$P = \{\boldsymbol{\rho}(t) : a \leq t \leq b\},$$

then

$$(P \oplus B_\delta) \ominus B_\delta = P.$$

Therefore the medial axis of $P \oplus B_\delta$ which can be obtained with an erosion with B_δ is simply P. In morphological terminology, P is said to be closed with respect to B_δ (not to be confused with Definition 5.3 above).

We choose distance from the path on the principal normal axis to express the concept of noninterference in Definition 5.4, however, because it allows us to express the requirements an object boundary path must meet, if we are to approximate it based on a point set obtained from a user. (This assumes the latter satisfies the requirements of an ordered approximation to $\boldsymbol{\rho}$ as defined in Definitions 5.5 and 5.6 below.)

Lemma 5.1 *Let $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ be a C^2 path which is non-interfering with respect to δ. Then for all $\epsilon > 0$, if $\epsilon < \delta$, $\boldsymbol{\rho}$ is noninterfering with respect to ϵ on $[a,b]$.*

Proof Suppose false. Then there is a $t_0 \in (a,b)$ such that for one of the two points p_ϵ on the principal normal vector to $\boldsymbol{\rho}$ at t_0, with $\| p_\epsilon - \boldsymbol{\rho}(t_0) \| = \epsilon$ there exists a $t_1 \in [a,b], t_1 \neq t_0$, such that

$$\| p_\epsilon - \boldsymbol{\rho}(t_1) \| \leq \epsilon$$

Then we have

$$\| p_\epsilon - p_\delta \| + \| p_\epsilon - \boldsymbol{\rho}(t_1) \| \leq \| p_\delta - \boldsymbol{\rho}(t_1) \|$$

which violates the triangle inequality. □

Definition 5.5 *Let $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ be a C^2 path which is non-interfering with respect to δ. Given $\epsilon \leq \delta$, we call $Q = \{q_1, q_2, \ldots, q_M\}$* **an approximation to $\boldsymbol{\rho}$ with respect to $\boldsymbol{\epsilon}$** *if the following conditions hold:*
1) Q is close to $\boldsymbol{\rho}$: $\| q_i - \boldsymbol{\rho}(t) \| \leq \epsilon$ for all $q_i \in Q$.
2) q_1 and q_M approximate the end points $\boldsymbol{\rho}(a)$ and $\boldsymbol{\rho}(b)$ respectively:

$$\begin{aligned} \| q_1 - \boldsymbol{\rho}(a) \| &< \| q_1 - \boldsymbol{\rho}(t) \| \text{ for all } t \in (a,b) \\ \| q_M - \boldsymbol{\rho}(b) \| &< \| q_M - \boldsymbol{\rho}(t) \| \text{ for all } t \in (a,b). \end{aligned}$$

3) For $1 \leq i < M$, let $l_i(s) = (1-s)q_i + sq_{i+1}$. The points $p \in \{l_i(s) : 0 \leq s \leq 1\}$, on the line segment between neighboring points of Q satisfy $\| p - \boldsymbol{\rho}(t) \| \leq \epsilon$ for some $t \in [a,b]$.

Lemma 5.2 *Let $\boldsymbol{\rho}$: $[a,b] \longrightarrow \mathbf{R}^2$ be a C^2 path which is non-interfering with respect to δ. If for some $\epsilon \leq \delta$, $Q = \{q_1, q_2, \ldots, q_M\}$ is an approximation to $\boldsymbol{\rho}$ with respect to ϵ, then for all $q_i \in Q$ there exists a $t_i \in [a,b]$ such that for $t \in [a,b]$, if $t \neq t_i$, then*

$$\| q_i - \boldsymbol{\rho}(t_i) \| < \| q_i - \boldsymbol{\rho}(t) \| . \tag{5.2}$$

Proof For q_1 and q_M, the above is guaranteed by the second condition in Definition 5.5. For $1 < i < M$, Since $\| q_i - \boldsymbol{\rho}(t) \| \geq 0$ for all t, there is a $t_0 \in [a,b]$ such

that

$$\| q_i - \boldsymbol{\rho}(t_0) \| \leq \| q_i - \boldsymbol{\rho}(t) \| \text{ for all } t \in [a, b].$$

It is easy to show q_i is on the principal normal vector to $\boldsymbol{\rho}$ at t_0. (Let $f(t) = \| q_i - \boldsymbol{\rho}(t) \|$, differentiate $f(t)$ and inspect the derivative of $\boldsymbol{\rho}$ at the minimum points.) Because $\| q_i - \boldsymbol{\rho}(t_0) \| \leq \delta$, by Lemma 5.1, $\boldsymbol{\rho}$ is noninterfering with respect to $\| q_i - \boldsymbol{\rho}(t_0) \|$, and therefore, by Definition 5.4, for all $t \in [a, b]$, if $t \neq t_0$,

$$\| q_i - \boldsymbol{\rho}(t_0) \| < \| q_i - \boldsymbol{\rho}(t) \| .$$

If for each $q_i \in Q$, we take $t_i = t_0$, this completes the proof. □

Definition 5.6 *Let $Q = \{q_1, q_2, \ldots, q_M\}$ be an approximation with respect to ϵ to a path $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$ which is noninterfering with respect to δ. For each $i \in [1, M]$, let $\boldsymbol{\rho}(t_i)$ be the point on $\boldsymbol{\rho}$ to which q_i is closest. Q is* **ordered** *if $i < j$ implies $t_i < t_j$.*

Note that if Q is an ordered approximation with respect to ϵ to a path $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$ which is noninterfering with respect to δ, then $q_i = q_j$ implies $i = j$.

Lemma 5.3 *An ordered approximation $Q = \{q_1, q_2, \ldots, q_M\}$ with respect to $\epsilon \leq \delta$ to a path $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$ which is noninterfering with respect to δ introduces a partition*

$$T = \{t_1 = a < t_2 < \cdots < t_M = b\}$$

of the interval $[a, b]$.

Proof The second condition in Definition 5.5 guarantees the existence of two points q_1 and q_M which are uniquely associated with the end points of the path $\boldsymbol{\rho}$ on $[a, b]$ and therefore with a and b. For $1 \leq i \leq M$, by Lemma 5.2 there exists a unique $t_i \in [a, b]$ for which Equation 5.2 holds. Definition 5.6 guarantees $t_i < t_j$ if $i < j$ for an ordered approximation Q. It can therefore be used to generate a partition of $[a, b]$. □

Definition 5.7 *Let $Q = \{q_1, q_2, \ldots, q_M\}, q_i \in \mathbf{Z}^2$ be a point set. If $\| q_i - q_{i+1} \| \in \{1, \sqrt{2}\}$ for $1 \leq i < M$, then Q is an* **8-connected** *digital path.*

Definition 5.8 *Let $Q = \{q_1, q_2, \ldots, q_M\}$ be an ordered approximation with respect to $\epsilon = \sqrt{2}/2$ to a path $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$ which is noninterfering with respect to some $\delta \geq \sqrt{2}/2$. If Q is an 8-connected digital path, we call Q a* **discrete** *or* **digital** *approximation to $\boldsymbol{\rho}$.*

5.2.2 Interpreting User Data

Suppose now that a user traces the boundary of an object in an image, that can be described by a C^2 path $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$ which is noninterfering with respect to some $\delta \geq \sqrt{2}/2$. Assume that as the user moves the pointer about in the image, the user position is recorded in the point set $U = \{u_k\}_{k=1}^{K}$. We want to show that if U is an ordered approximation to $\boldsymbol{\rho}$ with respect to some $\varepsilon \leq \delta$, it can be used to extract

a digital approximation to the path $\boldsymbol{\rho}$. To provide immediate feedback and simulate magnetic ink, we want to extract and display a digital approximation to the boundary path $\boldsymbol{\rho}$ as the user is tracing.

By Lemma 5.3, there is a partition $T = \{t_1 = a < t_2 < \cdots < t_K = b\}$ on $[a, b]$ defined by the user approximation U. Imagine that when the user point $u_k \in U$ is acquired, a digital approximation $V = \{v_1, v_2, \ldots, v_n\}$ to $\boldsymbol{\rho}$: $[a, t_{k-1}] \longrightarrow \mathbf{R}^2$ has already been extracted based on the first $k-1$ points in U. In this section, we want to show that based on the last known position $v_n \in V$ and the most recently obtained user position $u_k \in U$, we can describe a region known to contain a digital approximation to the kth boundary section $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$. Further, we will derive additional properties of a digital approximation V_k to $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$, which can be used to restrict the number of solutions.

Goal Set We begin by considering the candidate end points for the kth path segment V_k. We assume the user point u_k is roughly the same distance from $\boldsymbol{\rho}(t_{k-1})$ as the boundary point $\boldsymbol{\rho}(t_k)$ it approximates. Thus $\boldsymbol{\rho}(t_k)$ should fall on or near the path of a circle centered at $\boldsymbol{\rho}(t_{k-1})$ with radius $\| \boldsymbol{\rho}(t_{k-1}) - u_k \|$. Given points $q_1, q_2 \in \mathbf{R}^2$, let $O(q_1, q_2)$ denote the set of points on a circle of radius $\| q_1 - q_2 \|$ centered at q_1. So

$$O(q_1, q_2) = \{p \in \mathbf{R}^2 : \| p - q_1 \| = \| q_1 - q_2 \|\}.$$

Since, by assumption, $\| u_k - \boldsymbol{\rho}(t_k) \| \leq \varepsilon$, every candidate goal point should also satisfy $\|p - u_k\| \leq \varepsilon$. Let

$$C(q_1, q_2, \varepsilon) = \{p \in O(q_1, q_2) : \| p - q_2 \| \leq \varepsilon\}, \tag{5.3}$$

then $\boldsymbol{\rho}(t_k) \in C(\boldsymbol{\rho}(t_{k-1}), u_k, \varepsilon)$. Because we seek a digital path segment V_k, we define the goal set G_k as the Bresenham approximation [Bre77] to the arc $C(v_n, u_k, \varepsilon)$. An example is shown in Figure 5.1.

The Half-plane Containing $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$ The assumption that the set U is an ordered approximation to the path of the boundary is intuitively equivalent to saying the user moves along a strip centered about the object boundary, without making u-turns, just as one might drive on a multilane highway, occasionally changing lanes. Meanwhile, one can see a path $\boldsymbol{\rho}$ which is non-interfering with respect to some distance δ, as a highway containing $2\delta + 1$ lanes of traffic in the direction of movement. The following theorem states this formally.

Theorem 5.1 *Let $U = \{u_1, u_2, \ldots, u_K\}$ be an ordered approximation with respect to $\varepsilon \leq \delta$ to a path $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$ which is noninterfering with respect to some $\delta \geq \sqrt{2}/2$. Let $T = \{t_1 = a < t_2 < \cdots < t_K = b\}$ be the partition on $[a, b]$ introduced by U. All points $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$ are in the half-plane starting from $\boldsymbol{\rho}(t_{k-1})$ and moving towards $u_k \in U$, where the half-plane is defined as all points on the lines normal to $\ell(s) = (1-s)\boldsymbol{\rho}(t_{k-1}) + s u_k$ which contain a point in $L = \{\ell(s) : s \geq 0\}$. See Figure 5.2.*

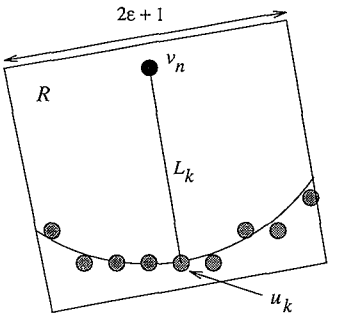


Figure 5.1: The region R about the vector L_k from v_n to u_k in which we seek the digital approximation V_k to the path $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$ of the image object boundary. The shaded points are the goal points $g \in G_k$, the Bresenham approximation to the circular arc $C(v_n, u_k, \varepsilon)$ defined in Equation 5.3.

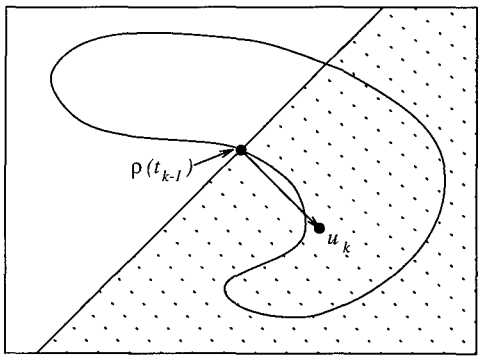


Figure 5.2: The planar region, or half-plane known to contain the kth section of the boundary path $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$.

Proof The theorem is true for $\boldsymbol{\rho}(t_{k-1})$ by the definition of the half-plane. Let $\ell(s) = (1-s)\boldsymbol{\rho}(t_{k-1}) + su_k$ be the line defined by $\boldsymbol{\rho}(t_{k-1})$ and $u_k \in U$, the point most recently obtained from the user. The third condition in Definition 5.5 assures that all points on the line segment

$$L = \{\ell(s) : 0 \leq s \leq 1\}$$

satisfy $\|\ell(s) - \boldsymbol{\rho}(t)\| \leq \varepsilon$ for some $t \in [t_{k-1}, t_k]$. Therefore any partition of the unit interval can be used to generate an ordered approximation to $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$ with respect to ε, made up of the points $\ell(s) \in L$. Let $S = \{0 = s_1 < s_2 < \cdots < s_J = 1\}$ be an arbitrary partition of the unit interval $[0, 1]$. For some j with $1 < j \leq J$, let $\ell(s_j)$ be a point on L corresponding to the jth member of S. By Lemma 5.2, there is some $t_0 \in [t_{k-1}, t_k]$ such that

$$\|\ell(s_j) - \boldsymbol{\rho}(t_0)\| < \|\ell(s_j) - \boldsymbol{\rho}(t)\| \quad \text{for all} \quad t \in [t_{k-1}, t_k] \quad \text{with} \quad t \neq t_0.$$

If the point $\boldsymbol{\rho}(t_0)$ is not in the half-plane defined by $\boldsymbol{\rho}(t_{k-1})$ and u_k, then

$$\|\ell(s_j) - \boldsymbol{\rho}(t_{k-1})\| \leq \|\ell(s_j) - \boldsymbol{\rho}(t_0)\|$$

This contradicts Lemma 5.2, and therefore completes the proof. □

Given a digital approximation $V = \{v_1, v_2, \ldots, v_n\}$ to $\boldsymbol{\rho}$: $[a, t_{k-1}] \longrightarrow \mathbf{R}^2$, Theorem 5.1 implies that in the search for a digital approximation V_k to the kth path section $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$, we can restrict our attention to the half plane starting at the last known point v_n and moving towards the user point u_k.

Path Direction Let

$$\ell_k(s) = (1-s)v_n + su_k$$

be the line determined by the points v_n and u_k. Then

$$L_k = \{\ell_k(s) : 0 \leq s \leq 1\} \tag{5.4}$$

is the set of points on the segment of $\ell_k(s)$ between v_n and u_k. Let r_k denote the length of L_k, given by

$$r_k = \|v_n - u_k\|, \tag{5.5}$$

and let θ_k denote the angle the directed line segment L_k makes with the x-axis, so

$$\theta_k = \tan^{-1}\left(\frac{y(u_k - v_n)}{x(u_k - v_n)}\right). \tag{5.6}$$

Because U is an ordered approximation to $\boldsymbol{\rho}$ with respect to ε, from Theorem 5.1 we know that for all $t \in [t_{k-1}, t_k]$, $\boldsymbol{\rho}(t) \in R$, where R is the region centered about L_k depicted in Figure 5.1. Now, because $\boldsymbol{\rho}$ is noninterfering with respect to δ on $[a, b]$, we can derive limitations on the direction of the tangent $\boldsymbol{\rho}'(t)$ to $\boldsymbol{\rho}$ at $\boldsymbol{\rho}(t)$ for all $t \in [t_{k-1}, t_k]$. Clearly any limitations on the direction of the tangent vector $\boldsymbol{\rho}'(t)$

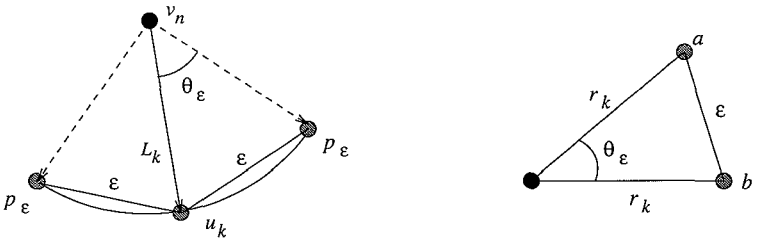


Figure 5.3: The angle θ_ε the vector a makes with the x-axis is the same as the corner angle at the vertex v_n of the triangle $T = \{v_n, u_k, p_\varepsilon\}$.

introduce corresponding restrictions on the local direction of a digital approximation $V_k = \{v_n, v_{n+1}, \ldots, v_m\}$ to $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$.

Limitations on $\boldsymbol{\rho}'(t)$ can be expressed in terms of ε, the maximum distance between the user point u_k and $\boldsymbol{\rho}(t_k)$, the point it approximates, δ, which determines the maximum curvature of $\boldsymbol{\rho}$ on $[a, b]$, and r_k, the distance covered in the kth path section.

In the remainder of this section, we derive a set of conditions which allow the set of candidate paths for a digital approximation V_k to be reduced. When these conditions are met, we can guarantee a digital approximation to $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$ is among the reduced set of candidates for V_k. This allows us to exploit a particularly efficient dynamic programming algorithm (described in Section 5.3) to extract a digital approximation V_k to $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$.

The Direction of $\boldsymbol{\rho}$ Let $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$ be a path which is noninterfering with respect to δ and let $U = \{u_k\}_{k=1}^K$ be an ordered approximation to $\boldsymbol{\rho}$ with respect to $\varepsilon \leq \delta$. Let $\theta(t)$ be the direction of the tangent vector $\boldsymbol{\rho}'(t)$ of $\boldsymbol{\rho}$ at $\boldsymbol{\rho}(t)$. Let $\Delta\theta$ denote the maximum difference in the direction $\theta(t)$ of the path $\boldsymbol{\rho}$ on $[t_{k-1}, t_k]$, and θ_k, the direction of the line segment L_k. From the argument in the proof to Theorem 5.1, we know that for all $t \in [t_{k-1}, t_k]$, if $\Delta\theta = \pi/2$, then

$$\theta_k - \Delta\theta \leq \theta(t) \leq \theta_k + \Delta\theta \tag{5.7}$$

We now derive conditions on $\boldsymbol{\rho}$ and U, with which the value of $\Delta\theta$ in Equation 5.7 can be reduced.

Let θ_{ave} denote the direction of the cord from $\boldsymbol{\rho}(t_{k-1})$ to $\boldsymbol{\rho}(t_k)$. Let p_ε be one of the points which satisfies

$$\|p_\varepsilon - v_n\| = r_k \quad \text{and} \quad \|p_\varepsilon - u_k\| = \varepsilon.$$

So p_ε is an extreme point on the arc shown in Figure 5.3.

Clearly, if θ_ε is the corner angle at the vertex v_n of the triangle $T = \{v_n, u_k, p_\varepsilon\}$,

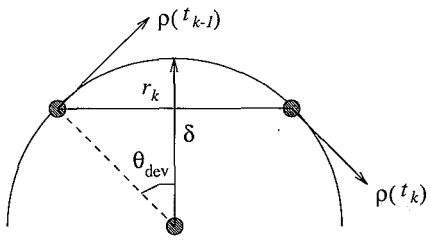


Figure 5.4: The extent to which $\theta(t)$ can differ from θ_{ave} depends on r_k and δ. In this example, $\theta_{\text{ave}} = 0$.

then

$$\theta_k - \theta_\varepsilon \leq \theta_{\text{ave}} \leq \theta_k + \theta_\varepsilon. \tag{5.8}$$

Consider Figure 5.3. By the law of similar triangles, we have

$$\theta_\varepsilon = \cos^{-1}\left(1 - \frac{\varepsilon^2}{2r_k^2}\right). \tag{5.9}$$

Let θ_{dev} be the maximum difference between the angle $\theta(t)$ of the tangent vector $\boldsymbol{\rho}'(t)$ on $[t_{k-1}, t_k]$, and θ_{ave}, the average direction of the tangent vector on $[t_{k-1}, t_k]$. Then

$$\theta_{\text{dev}} = \max_{t \in [t_{k-1}, t_k]} \{|\theta(t) - \theta_{\text{ave}}|\}$$

Because $\boldsymbol{\rho}$ is noninterfering with respect to δ, the maximum curvature of the path $\boldsymbol{\rho}$ on $[t_{k-1}, t_k]$ is δ. Therefore, θ_{dev} can be expressed in terms of the relationship of r_k to δ. From Figure 5.4, it is clear that the maximum deviation in angle on $[t_{k-1}, t_k]$ is given by

$$\theta_{\text{dev}} = \sin^{-1}\frac{r_k}{2\delta}. \tag{5.10}$$

Theorem 5.2 *Let $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$ be a C^2 path which is non-interfering with respect to δ. Let $U = \{u_1, u_2, \ldots, u_K\}$ be an approximation to $\boldsymbol{\rho}$ with respect to $\varepsilon \leq \delta$. Let $T = \{t_1 = a < t_2 < \cdots < t_K = b\}$ be the partition of $[a, b]$ introduced by U. Further, suppose $V = \{v_1, v_2, \ldots, v_n\}$ is a digital approximation to $\boldsymbol{\rho}$: $[a, t_{k-1}] \longrightarrow \mathbf{R}^2$. Let $\Delta\theta$ denote the maximum absolute difference between the angle θ_k defined in Equation 5.6, and the angle $\theta(t)$ of a vector $\boldsymbol{\rho}'(t)$ tangent to $\boldsymbol{\rho}$ on $[t_{k-1}, t_k]$. If $r_k \leq 2\delta$, then*

$$\Delta\theta \leq \cos^{-1}\left(1 - \frac{\varepsilon^2}{2r_k^2}\right) + \sin^{-1}\left(\frac{r_k}{2\delta}\right), \tag{5.11}$$

where r_k is defined in Equation 5.5.

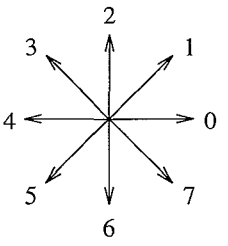


Figure 5.5: The Freeman codes for the 8 possible directions between two neighbors on an 8-connected digital path.

Proof Clearly the absolute difference between the path direction and the direction θ_k is bounded above by the sum of the maximum difference θ_ε between the average angle of the path θ_{ave} and θ_k, and the maximum deviation (θ_{dev}) from the average angle on $[t_{k-1}, t_k]$. The theorem thus follows from the arguments leading to Equations 5.8, 5.9 and 5.10 above. □

As is clear from the arguments leading to Theorem 5.2, there are two key factors which contribute to the upper bound on $\Delta\theta$.

1. θ_ε increases as a function of ε/r_k.

2. θ_{dev} increases as a function of r_k/δ.

If r_k is fixed, then $\Delta\theta$ decreases a function of δ/ε.

What about V_k? The upper bound on $\Delta\theta$ derived in Theorem 5.2 introduces corresponding restrictions on a digital approximation V_k to $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$. We now examine the effect on V_k of Equations 5.7 and 5.11, and show how the results can be exploited to restrict the set of 8-connected paths which may be seen as candidates for V_k. In the next section, an efficient algorithm is described to find an optimal path among the reduced set of candidates.

Given two points $p, q \in \mathbf{Z}^2$, let $\phi(p, q)$ be the Freeman chain code for the direction from p to q if p and q are 8-connected neighbors, and be undefined otherwise. In $\mathbf{Z}^2$, the direction of movement from p to an 8-connected neighbor q is limited to $\gamma_i \in K \cdot \pi/4$ for $K \in \{0, 1, ..., 7\}$, where the value of K is the Freeman code for the path movement from p to q as shown in Figure 5.5.

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a digital approximation to a segment $L \in \mathbf{R}^2$ of a line $\ell(s) = (1 - s)p_1 + sp_2$ (with $p_1, p_2 \in \mathbf{R}^2$). From Bresenham [Bre65], we know two Freeman codes are sufficient to express the direction between neighboring elements of a straight line segment. An example is shown in Figure 5.6.

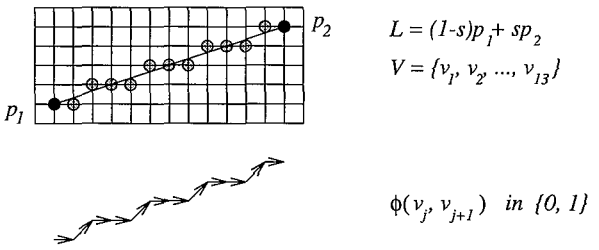


Figure 5.6: A line segment $L \in \mathbf{R}^2$ with a digital (Bresenham) approximation V superimposed on it, and the path direction $\phi(v_j, v_{j+1})$ between neighbors in V indicated with arrows. Two Freeman codes are sufficient to encode the direction from one member to the next for all members $v_j \in V$.

Clearly the set of Freeman codes required to express the direction between neighbors of a digital approximation V_k to the kth section of a path $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$, is limited to those required for the tangent vectors $\boldsymbol{\rho}'(t)$ to $\boldsymbol{\rho}$ on $[t_{k-1}, t_k]$.

Assume the direction from v_n to u_k given by θ_k in Equation 5.6 satisfies[3]

$$3\pi/2 \leq \theta_k \leq 7\pi/4 \tag{5.12}$$

We can transform any region in a digital image defined about a vector $\mathbf{v}$, with angle $\theta_k \notin [3\pi/2, 7\pi/4]$ to a region defined about a vector $\mathbf{v}_t$ with angle $\theta_t \in [3\pi/2, 7\pi/4]$ with a rotation of $m \cdot \pi/2$ for $m \in \{1, 2, 3\}$, and/or a reflection about the x or y axis. Since neither of these transformations require a resampling of the image data, we may restrict our attention to $\theta_k \in [3\pi/2, 7\pi/4]$ without loss of generality.

Suppose $\Delta\theta \leq \pi/4$. From Equation 5.7, we have

$$5\pi/4 \leq \theta(t) \leq 2\pi \quad \text{for all} \quad t \in [t_{k-1}, t_k]. \tag{5.13}$$

This means the direction from a point v_j to its neighbor v_{j+1} in a best approximation (in the Bresenham sense) to $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$ can be coded by some member of the set $F = \{5, 6, 7, 0\}$, as is the case for every vector $\boldsymbol{\rho}'(t)$ tangent to $\boldsymbol{\rho}$ on $[t_{k-1}, t_k]$.

In the following section, we describe an algorithm with which we can extract any 8-connected path from v_n to some $g \in G_k$ (see Figure 5.1) with the Freeman code for the path direction between neighboring points satisfying

$$\phi(v_j, v_{j+1}) \in \{5, 6, 7, 0\}. \tag{5.14}$$

If ε is the error permitted the user in specifying the set $U = \{u_k\}_{u=1}^{K}$, then from Theorem 5.2, we can specify conditions on r_k and δ, for which we can be certain

[3]The preference for this octant is because it is used to express the path direction in our algorithm in the next section.

that an approximation to $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$ is among the set of digital paths which can be obtained with our algorithm. Clearly a lower bound on δ limits the set of paths for which an approximation to the boundary path is guaranteed to be among the candidates. On modern workstations, we can manipulate r_k to be as small as required. It may therefore be seen as the sampling frequency on the user input, and can be manipulated freely.

From Equations 5.9 and 5.10, we can be certain that if

$$2\varepsilon \leq r_k \leq \delta/2,$$

then by Theorem 5.2, we then have $\Delta\theta \leq \pi/4$ as desired.

In Section 5.6, the method is shown to work well even when these conditions are not met. In particular, we use $\varepsilon = 4$ and $r_k = 4\varepsilon/3 = 6$, and the method works even for $\delta = 5r_k/3 = 10$. The results in this section should be interpreted as describing the combination of boundary path and user input conditions for which we can *guarantee* a good digital approximation is within the set of candidates. As such, they express limitations ($\delta \geq 4\varepsilon$) which must be met if we are to find a path in the worst case. Our results show the method to work quite effectively for a far wider range of conditions in practice.

5.3 Dynamic Programming to Find V_k

In the previous section, we showed that given an approximation U to a boundary path $\boldsymbol{\rho}$: $[a, b] \longrightarrow \mathbf{R}^2$, characteristics of a digital approximation to the kth section of the boundary path are known when the conditions in Theorem 5.2 are met. We now look for an digital approximation V_k to the kth boundary section $\boldsymbol{\rho}$: $[t_{k-1}, t_k] \longrightarrow \mathbf{R}^2$ starting from v_n, the last point on a digital approximation V to $\boldsymbol{\rho}$: $[a, t_{k-1}] \longrightarrow \mathbf{R}^2$. V_k should end at some point $g \in G_k$, as described in Figure 5.1. Further, assuming Equation 5.12, the Freeman code of the direction between neighboring elements in the path must satisfy Equation 5.14. Assuming we have a cost function $c(x, y)$ which is low on boundary points and high elsewhere, we now develop a dynamic programming algorithm with which we can extract a minimum cost path among the candidates.

5.3.1 The Cost of a Path

Given the point v_n and the goal set G_k, consider the region R in Figure 5.1 made up of the points in the minimum rectangle oriented about L_k containing G_k. Let R_w be the vertically oriented rectangle shown in Figure 5.7 defined so that based on the points in R_w, the cost function $c(x, y)$ can be computed for all points $(x, y) \in R$. To illustrate the algorithm, we compute the cost of each point $(x, y) \in R$ using a diagonal directional difference operator to estimate the gradient magnitude $c(x, y) = Q - |g'(x, y)|$, and choose $Q \geq |g'(x, y)|$ for all $(x, y) \in R_w$ to assure $c(x, y) \geq 0$ for all $(x, y) \in R_w$ (see Section 5.3.2 for the real cost function).

To eliminate paths not fully contained in R, we redefine the cost of each element

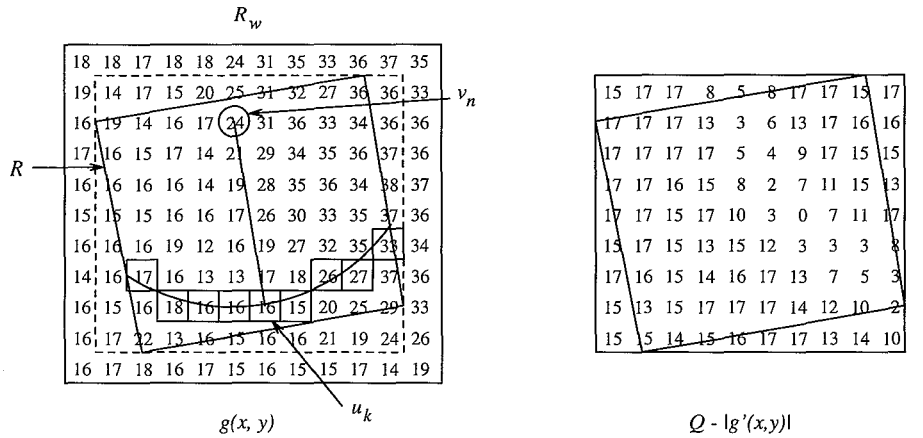


Figure 5.7: The image function $g(x, y)$ in the image matrix R_w, and the cost matrix $c(x, y) = Q - |g'(x, y)|$ with $Q = 17$. To compute $|g'(x, y)|$ with the diagonal difference operator for all points $(x, y) \in R$, extra columns and rows of image data are used. In the matrix on the left, the circled point is the start point v_n; the line from v_n leads to the point u_k proposed by the user, and the boxed points which are those on the Bresenham approximation to the circular arc, make up the goal set G.

in the image matrix R_w to be:

$$C(x, y) = \begin{cases} c(x, y) & \text{if } (x, y) \in R \\ \infty & \text{otherwise} \end{cases} \tag{5.15}$$

Let Ψ denote the set of candidate paths for V_k. From Equation 5.14, for every path $\psi \in \Psi$, we have the Freeman code for the direction of movement from ψ_i to ψ_{i+1} limited to those in the set $F = \{5, 6, 7, 0\}$. Therefore, the valid forward neighbors ψ_{i+1} for a path element $\psi_i \in \psi$ are those points $\psi_{i+1} \in \mathbf{Z}^2$ for which $\phi(\psi_i, \psi_{i+1}) \in F$, and the valid backward neighbors ψ_{i-1} are those for which $\phi(\psi_i, \psi_{i-1}) \in B$, where $B = \{1, 2, 3, 4\}$. See Figure 5.8. To enforce connectivity and path direction requirements, we now define

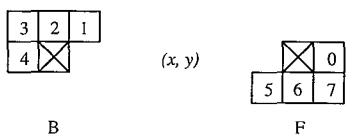


Figure 5.8: Given $\psi_i = (x(\psi_i), y((\psi_i))$, the candidate preceding path neighbors ψ_{i-1} (left), and subsequent neighbors ψ_{i+1} (right).

the cost of a path element ψ_i given ψ_{i-1} as

$$\Phi(\psi_{i-1}, \psi_i) = \begin{cases} C(x(\psi_i), y(\psi_i)) & \text{if } \phi(\psi_i, \psi_{i-1}) \in B \\ \infty & \text{otherwise} \end{cases} \tag{5.16}$$

Note that we could also have expressed the conditional statement in terms of the set of permitted forward movements F. If we now define the cost of a path ψ as

$$\Phi_*(\psi) = \sum_{i=1}^{n(\psi)} \Phi(\psi_{i-1}, \psi_i)$$

we know that every path ψ for which $\Phi_*(\psi)$ is finite, is 8-connected, satisfies $\psi \subset R$, and for all $\psi_i \in \psi$, $\phi(\psi_i, \psi_{i+1}) \in F$ as required.
To assure the path begins at v_n, we use[4]

$$\Phi(\psi_0, \psi_1) = \begin{cases} C(x(\psi_1), y(\psi_1)) & \text{if } \psi_0 = v_n \text{ and } \phi(\psi_0, \psi_1) \in F \\ \infty & \text{otherwise} \end{cases} \tag{5.17}$$

With the use of the following recursive functions, for all points $(x, y) \in R_w$, we can compute the cumulative cost of the minimum cost subpath leading from the start point v_n to $(x(\psi_i), y(\psi_i))$, as follows. We start with $\varphi(x, y) = \infty$ for all $(x, y) \in R_w$.

$$\varphi(x(v_n), y(v_n)) = 0 \tag{5.18}$$

$$\varphi(x(\psi_i), y(\psi_i)) = \min[\varphi(\psi_{i-1}) + \Phi(\psi_{i-1}, \psi_i)] \tag{5.19}$$

where Equation 5.19 is computed for all $(x, y) \in R_w$ with $y \geq y(v_n)$. Note that because $\Phi(\psi_{i-1}, \psi_i)$ is finite only for path neighbors which satisfy $\phi(\psi_i, \psi_{i-1}) \in B$, for each point $(x, y) \in R_w$, we need only compute the values for the four preceding neighbors (Figure 5.8). This means that the cumulative costs $\varphi(x(\psi_i), y(\psi_i))$ can be computed with a single pass through the cost matrix $C(x, y)$.

As the cumulative cost matrix $\varphi(x(\psi_i), y(\psi_i))$ is computed, we maintain a matrix of pointers containing the backward direction $\phi(\psi_i, \psi_{i-1})$ to the neighbor which resulted in the minimum value.

$$\phi(x, y) = \arg\min[\varphi(\psi_{i-1}) + \Phi(\psi_{i-1}, \psi_i)] \tag{5.20}$$

We have $\phi(x, y) \in B$ for all $(x, y) \in R_w$, which are valid candidate path elements for V_k.

The effect of applying these computations to our example is shown in Figure 5.9, along with the optimal path found by scanning the matrix of pointers by means of the recursion:

$$\psi_m = \arg\min_{p \in G} \varphi(p), \tag{5.21}$$

[4]See paragraph on join points in Section 5.3.1.

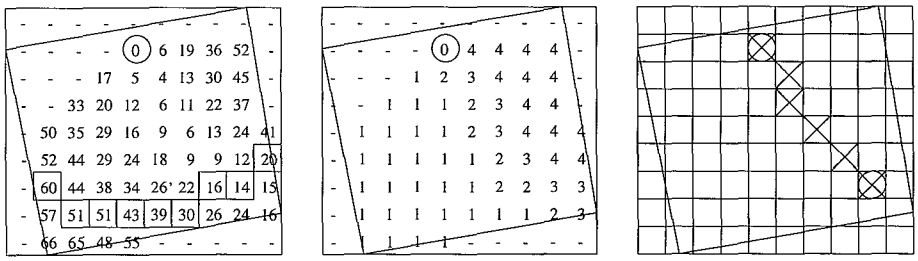


Figure 5.9: Each element $\varphi(x, y)$ in the leftmost matrix contains the minimum cumulative cost for a path from v_n to (x, y). The boxed elements make up the goal set G. A dash indicates $\varphi(x, y) = \infty$. The elements $\phi(x, y)$ in the central matrix contain the pointers (stored while computing $\varphi(x, y)$) necessary to retrieve the path. On the right is the path $\psi \subset R$ from v_n to the point $p \in G$ with the minimum costs, retrieved by following the pointers in $\phi(x, y)$ from p to v_n.

$$\psi_{i-1} = \phi(x(\psi_i), y(\psi_i)) \tag{5.22}$$

where $m = n(\psi)$ is the number of points in the minimum cost path ψ from the point v_n leading to some point $p \in G$, the goal set.

Combined with the recursive cost computations, it is the maintenance of $\phi(x, y)$, later used to retrieve the minimum cost path, which characterizes the algorithm presented here as a *dynamic programming* algorithm [CLR90].

Notes

End Points The begin point for the path extracted is fixed to v_n in Equations 5.17 and 5.18. The algorithm can easily be adjusted to permit a set S of start points with a slight modification to the if statement in Equation 5.17 (if $\psi_0 \in S$), and by applying Equation 5.18 to all points $s \in S$. Of course, it could also be made to end at a specific point $(x, y) \in R_w$ if we replace Equation 5.21 with $\psi_m = p$, which is equivalent to reducing the goal set G to a single point.

We draw attention to these alternatives because they are used to determine the end points of the path V. Further, they are minor modifications to the algorithm which can be used to make it applicable to a wide range of "one-way" path search problems other than that considered here.

Join Points Each point v_n for $n > 1$, from which we seek a kth path segment is also an end point for the $(k-1)$st path segment. To assure the behavior of the path is consistent with respect to the restrictions on both path segments at the join point, the set of permitted points for ψ_1 in the kth segment must satisfy the restrictions on

the $(k-1)$st segment as well as those on the kth. Denote the set of permitted forward movements on the kth path section in its initial orientation as F_k, and define

$$F_0 = F_k \cap F_{k-1}.$$

Unless the user turns 180°, $F_0 \neq \emptyset$. If we rotate F_0 along with F_k, the path at each joint point v_n will satisfy both sets of restrictions, if the set F_0 replaces the set F in Equation 5.17.

5.3.2 The Cost of a Path Element

In the dynamic programming algorithm described in Section 5.3, it is the cost function that determines which of the candidate paths will be extracted from the region of interest R. Thus, the cost function must embody the properties of the object boundary to the best of our knowledge. In this section, we develop a cost function based on the following image model known to be appropriate for a variety of imaging methods [GW92, Pra91]. The cost function can be adapted to extract object boundaries when a priori knowledge supports the presence of a different model.

The Image Model

Let $g(x, y)$ denote the image intensity function, defined on an $X \times Y$ image I. Assume $g(x, y)$ is be corrupted with independent Gaussian noise $n(x, y)$:

$$g(x, y) = f(x, y) + n(x, y) \tag{5.23}$$

where $f(x, y)$ denotes the noise free image function, with constant value f_0 in the object region and f_1 in the background region. The object and background regions are assumed to be separated by a connected transition region, with values between f_0 and f_1, corresponding to the image object boundary. (The image data in $R_w \subset I$ used in the example in Figure 5.7 follows this model with $f_0 = 36$, $f_1 = 16$ and the standard deviation of the Gaussian noise, $\sigma_{noise} = 2$).

Because the transition region is earmarked by a change in image values, we aim to define a cost function which will have a high value in smooth areas of the image and a low value in transition areas. We further assume that the noise free image function may consist of additional values $f_2, f_3, \ldots$ which correspond with the intensity of objects which are not of interest for the segmentation task at hand. These objects may intersect or be near the transition region we seek to detect, thereby interfering with its detectability based solely on the model in Equation 5.23. We therefore aim to suppress sensitivity to both the noise function $n(x, y)$ and the presence of other image objects.

The Hypothesized Direction α_k

When a point u_k is obtained from the user, the direction θ_k, defined in Equation 5.6 from v_n to u_k, is also obtained. This provides an indication of the boundary direction

on the kth path section. Full reliance on θ_k, however, results in sensitivity to jitteryness on the part of a user. To estimate the boundary direction, we therefore want to average it with the mean direction of the last contour segment, to obtain the hypothesized direction α_k. To this end, we define

$$\alpha_k = \nu\theta_k + (1-\nu)\beta_{k-1}, \tag{5.24}$$

where β_k denotes the average direction of movement in the kth extracted path section. Note that as ν increases, the user is followed more closely, whereas small values of ν make it hard to change direction. To encourage a smooth path which follows the user when the boundary is weak, α_k is incorporated in the cost of a path element as described below. These values are optimized for our rather jittery user simulations in Section 5.5.1. One may argue that in practice, these values should be adapted for a particular user. Our optimization method could be applied to user data to achieve this.

The Cost Function

We now design a cost function with which an optimal path will follow the object boundary when the image gradient is high, and the user otherwise. This is accomplished by defining the cost of a path element in terms of a weighted combination of factors depending on the strength of the image gradient and the hypothesized boundary direction α_k, described in Equation 5.24 above.

Consider the following function defined in terms of α_k (see Section 5.3.2),

$$\Theta(x,y) = |\alpha_k - \theta(x,y)| \pmod{\pi}$$

where $\theta(x,y)$ is the direction from v_n to the point (x,y). By definition

$$0 \leq \Theta(x,y) \leq \pi \tag{5.25}$$

and $\Theta(x,y)$ decreases as the direction to the point (x,y) from v_n gets close to α_k.

If α_k is the direction of the contour, then the directional derivative defined by

$$Dg_\alpha(x,y) = \nabla g(x,y) \cdot (\cos\alpha, \sin\alpha)$$

will be high along areas of the contour either for $\alpha = \alpha_k + \pi/2$, or for $\alpha = \alpha_k - \pi/2$. It will be low for transitions in the image function in other directions. Let Q be a large positive number which satisfies

$$Q \geq |\nabla g(x,y)| \text{ for all } (x,y) \in I. \tag{5.26}$$

Then,

$$0 \leq Q - Dg_\alpha(x,y) \leq 2 \cdot Q \tag{5.27}$$

for all α and for all $(x, y) \in R_w$. If we choose

$$Q = \max_{(x,y)\in I} g(x, y) - \min_{(x,y)\in I} g(x, y))$$

then Equation 5.26 is satisfied and we have a positive bounded function $Q - Dg_\alpha(x, y)$ which decreases as the magnitude of the gradient increases and its direction approaches that expected. Figure 5.7 contains an example of this function with $Q = 17$, and $\alpha = 5\pi/4$. The use of the directional derivative rather than the gradient magnitude, results in large values of $Q - Dg_\alpha(x, y)$, if the gradient direction differs significantly from α, even when the gradient magnitude is large. This results in a suppression of interference from points on (nearly) perpendicular object boundaries. If incorporated in the cost function, it will therefore make transitions which agree in direction with the hypothesized boundary direction α_k have low costs. By selecting

$$\alpha = \alpha_k + \pi/2 \text{ or } \alpha = \alpha_k - \pi/2$$

depending on which results in a positive value for $Dg_\alpha(x, y)$ at the point v_n (which we assume is correct), disturbance from nearby parallel boundaries is also suppressed.

If we multiply Equation 5.27 by $\pi/2Q$, then, given Equation 5.25, we have both cost functions restricted to the range $[0, \pi]$. Because we want to balance the influence of the user input and the image data, assuming the actual direction of the contour is close to α_k, we define the cost of a point $(x, y) \in R_w$ as the weighted sum

$$c(x, y) = \omega\Theta(x, y) + (1 - \omega)\left(\frac{\pi}{2Q}[Q - Dg_\alpha(x, y)]\right) \tag{5.28}$$

The parameter ω determines the balance of user influence and the influence of image data in the cost function. Increasing ω will reduce sensitivity to noise in the image, but increase sensitivity to user error. In Section 5.5.2, criteria to optimize ω are defined and experiments are described with which we obtain its optimal value. If a priori knowledge of the image objects or a specific user is available, the experiments may be repeated to tune ω for the specific problem.

5.3.3 Related Work

Dynamic Programming

The method developed in Section 5.3 to extract a path $\psi \in R$ based on user input is similar to that described by Gerbrands in [Ger88] to extract the path of a boundary given an initial estimation. A key benefit of our method as compared to Gerbrands' is that ours does not require a resampling of the image data. Permitting movement in the four Freeman directions of a half-plane and using the directional derivative in the cost function makes the resampling required in Gerbrands' method superfluous. Elimination of the resampling step produces a sufficient performance improvement to allow extraction of the boundary path in real time as the user traces.

Our method can easily be applied to the problem posed in [Ger88] of extracting a

smooth 8-connected boundary path given an initial estimation. If Gerbrands' resampling step is replaced by a step to obtain a rough polygonalization of the boundary, then we have the points $U = \{u_k\}$ required for our method. Using Wall and Danielsons algorithm [WD84], this can be done in $O(N)$ time where N is the number of pixels in the initial estimation. Because the resampling step is the most computationally intensive step in Gerbrands' method, this results in a substantial performance improvement. Furthermore, because our method produces an 8-connected path $V \subset \mathbf{Z}^2$ it can be efficiently stored, and the wealth of efficient analysis methods which have been developed for this class of paths (see for example [GW92, Pra91, RK76]) may be applied.

Path Planning

We have formulated the question of finding a boundary section V_k as a minimum cost path problem. As such, it is closely related to path planning problems which arise in the field of robotics. In particular, the dynamic programming algorithm developed in Section 5.3.1 can be seen as a variation on the cost wave propagation method for path planning proposed by Dorst and Trovato in [DT88].

The path planning problem is viewed in their work as one of finding the minimum cost path through a configuration (parameter) space, by means of propagating waves of minimum cost paths in that space. The location of a path element and the direction of its preceding neighbor: $C = \{x(\psi_i), y(\psi_i), \phi(\psi_i, \psi_{i-1})\}$ may be seen as the points of a parameter space, and the recursive function $\varphi(x, y)$ defined in Equation 5.19 may be seen as a wave propagation. We then propagate in a forward direction based on Equation 5.16.

The cost of a path in their method requires a metric or cost function such as that introduced in Equation 5.28. However, the algorithm proposed in [DT88], walks through each wave of minimum cost points, generates the next wave, and stops at the first goal point, under the assumption that the first hit will be a goal point with minimum costs. This requires the cost function to contain a heuristic element which assures this assumption holds, and thus, their method can be classified an A^* algorithm (see for example [Nil80]) rather than a dynamic programming method. Computing the minimum costs to all goal points is simple and efficient for the path search we perform, and allows the use of a substantially simpler cost function.

5.4 Experiments

In this section, we describe the experimental methods used in Section 5.5 to optimize the parameters ν and ω used in Equations 5.28 and 5.24, which control the cost of path elements and ultimately the selection of the optimal path. The same methods are used in Section 5.6 to evaluate the magnetic contour tracing tool described in this chapter.

The cost parameters ν and ω are tuned using artificial test images containing discs of varying radii as may have been produced by a camera. We simulate the user by

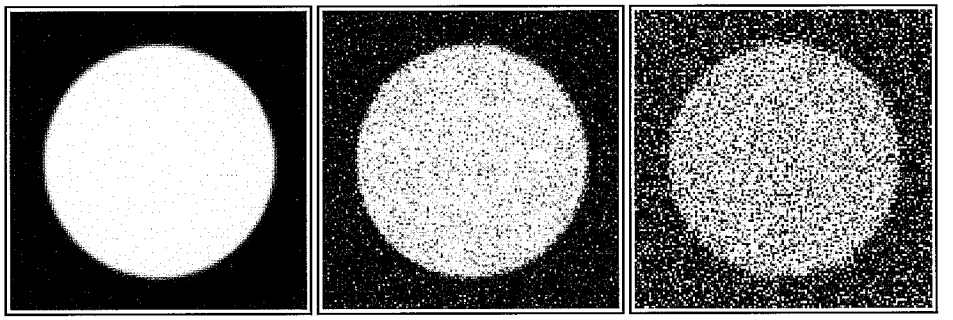


Figure 5.10: Test Images: A disc with object value $f_0 = 200$, and background value $f_1 = 100$. The middle and right images have added Gaussian noise with $\sigma_{noise} = 20$, ($SNR = 5$, $SNR_{dB} = 26$) and $\sigma_{noise} = 50$, ($SNR = 2$, $SNR_{dB} = 6$) respectively.

choosing random locations near the test image object boundary for the points u_k. To evaluate the performance of the method using these parameters for a range of image objects, we used test images containing hexagons of various sizes and a range of ellipses in addition to the discs used to tune the parameters.

The quality of the boundary produced is measured by a variation of Baddeley's Δ^2 metric [Bad92b].

5.4.1 The Images

Disc images for a range of radii are used to tune ν and ω so the method will be applicable for path sections of high and low curvature. Later we evaluate the method for a range of radii to measure the influence of curvature on the method. In accordance with our image model (Equation 5.23) we add independent Gaussian noise to evaluate the influence of the signal to noise ratio (SNR).

We create a disc of radius r in a 256×256 image as follows. In a 2048×2048 image we assign the object image value f_0 to all points p which satisfy $\| c - p \| < 8 * r$ where c is the center point of the image: $c = (1024, 1024)$. All other points are assigned the background value f_1. After performing a Gaussian averaging using $\sigma = 8$, we sample the image by extracting one point for each 8×8 neighborhood in the original image. Using intensity $f_0 = 200$ for the object, and $f_1 = 100$ for the background, results in the leftmost image in Figure 5.10. We experiment with discs of radii $r \in \{10, 20, ..., 100\}$. The hexagon and ellipse images used in the evaluation in Section 5.6 are created in an analogous fashion.

If we add independent Gaussian noise to our test images, we have the signal to noise ratio (SNR) defined by

$$SNR = \frac{|f_0 - f_1|}{\sigma_{noise}}$$

where σ_{noise} is the standard deviation of the Gaussian noise. Alternatively, the signal to noise ratio may be expressed in decibels as defined in

$$SNR_{dB} = 20 \log \left(\frac{|f_0 - f_1|}{\sigma_{noise}} \right).$$

In our experiments, the object and background values are fixed to be 200 and 100 (not necessarily in that order). We use values $\sigma_{noise} \in \{1, 5, 10, 15, ..., 50\}$, resulting in values for $SNR \in \{100, 20, 10, ...2\}$, or as measured in decibels $SNR_{dB} \in \{40, 26, 20, ...6\}$. Examples of the resulting images are shown in the center and rightmost images in Figure 5.10.

5.4.2 User Simulation

Recall that the user is presented with a tool which records the position of the pointer in the image as the pointing device (mouse) is moved. To simulate a user tracing a boundary, we walk around the image object boundary in a test image, and at regular intervals, extract a random point in the immediate neighborhood.

Let s denote the step size, which in practice depends on the speed of the user's movement. Let $B = \{b_1, b_2, ..., b_{n(B)}\}$ be the best digital approximation to the image object boundary, which consists of $n(B)$ points. For discs and hexagons, this corresponds to the Bresenham point set [Bre65, Bre77]. For ellipses, we use the point set produced by the algorithm in [FvDFH90, pages 88-91]. Beginning at b_1, we move through the point set in steps of s, and at each point $b_i \in \{b_1, b_{1+s}, b_{1+2s}, ...\}$, we choose random point on the line perpendicular to the path of the object boundary which passes through the point b_i. The point selected is forced to satisfy $\| b_i - p \| \leq \varepsilon$, where ε is the distance the user is allowed to stray from the true path of the image object boundary.

In Figure 5.11, we illustrate the extraction of a user point p given a point $b_i \in B$ for a disc, and a point set extracted with this method is shown.

5.4.3 The Error Measure

In [Bad92b], Baddeley introduced the Δ^p error measure to quantify the difference between two binary images A and B:

$$\Delta^p(A, B) = \left[\frac{1}{n(X)} \sum_{x \in X} (d_c(x, A) - d_c(x, B))^p \right]^{1/p} \tag{5.29}$$

Here X is the raster image in which A and B are embedded. A binary image $A \subseteq X$ is simply a subset of the points $x \in X$. $n(X)$ denotes the size of the raster X, and $d_c(x, Y)$ is the *cutoff* distance transform:

$$d_c(x, Y) = \begin{cases} d(x, Y) & \text{if } d(x, Y) \leq c \\ c & \text{otherwise} \end{cases}. \tag{5.30}$$

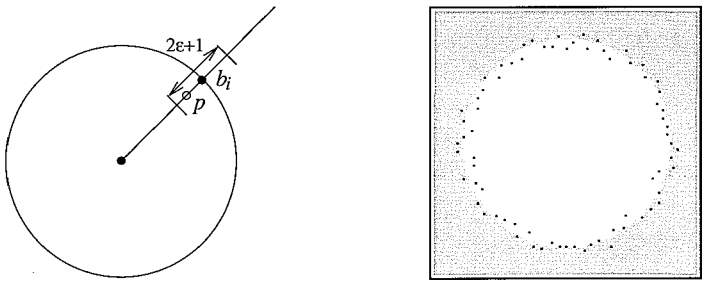


Figure 5.11: On the left, the random selection of a point p near the point b_i generated by the Bresenham algorithm. A simulated user point set for a disc of radius 50, with $s = 4$, and $\varepsilon = 4$.

An example of the cutoff distance transform is shown in Figure 5.12.

Baddeley shows that $\Delta^2(A, B)$, ($p = 2$ in Equation 5.29) has some key properties required of an error measure on binary images. Suppose the image A contains the correct path of an object boundary, and the image B contains the boundary to be evaluated. Due to the use of the cutoff distance transform, and the balance of influence of A and B in Equation 5.29, Δ^2 is sensitive to false positives ($B \backslash A$), to false negatives ($A \backslash B$), and to the influence of an error on boundary shape, thereby addressing previously noted shortcomings of FOM, Pratt's figure of merit [Pra77]. In particular, if the boundary B is missing points in A (false negatives), this will not effect the value of $FOM(A, B)$. As a result, FOM is insensitive to some significant differences in boundary shape, as noted in both [PM82] and [vVYB89].

A careful look at Equations 5.29 and 5.30, however, shows that if more pixels are set in the images A and B, the number of pixels which contribute to the sum in Equation 5.29 increases. Given A, the image of the true object boundary, suppose we define the *cutoff area* of A as $n(A_c)$, the number of pixels in the set $A_c = \{x : d(x, A) <= c\}$. For a disc, the cutoff area depends on the radius r and is given by

$$n(A_{r,c}) = \pi(r + c)^2 - \pi(r - c)^2 = 4\pi rc. \tag{5.31}$$

For a hexagon, the cutoff area depends on the side length l and is given by

$$n(A_{l,c}) = 4\pi c^2 + 12c\left(l - \frac{2c}{\sqrt{3}}\right), \tag{5.32}$$

and for an ellipse defined by

$$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0, \tag{5.33}$$

we have

$$n(A_{a,b,c}) = \pi(b + c)(a + c) - \pi(b - c)(a - c) = 2\pi c(a + b). \tag{5.34}$$

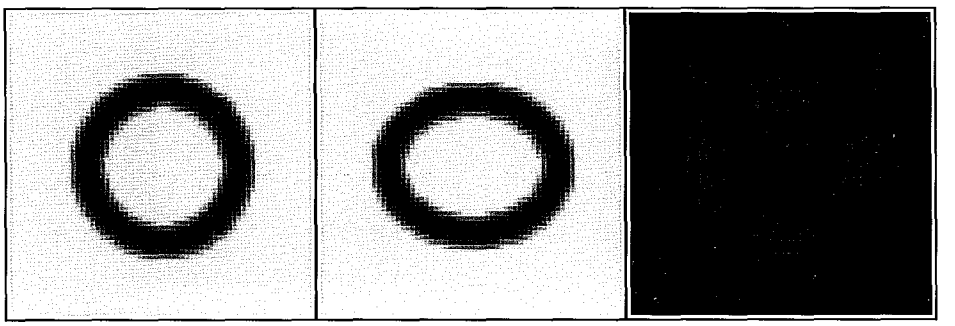


Figure 5.12: The cutoff distance transform, with $c = 4$. On the left, $d_c(x, A)$, where A is a circle of radius $r = 16$. In the middle, $d_c(x, B)$ where B is an ellipse with $a = 18$ and $b = 14$ (See Equation 5.33). On the right, $|d_c(x, A) - d_c(x, B)|$. Note that only the points near the boundaries A and B have a positive value (black= 0), and will contribute to the summation in Equation 5.29.

To obtain the average distance to the true contour for each of the test images we use the following variation on Baddeley's Δ^2 error measure:

$$\delta_c^2(A, B) = \left[\frac{1}{n(A_c)} \sum_{x \in X} (d_c(x, A) - d_c(x, B))^2\right]^{1/2} \tag{5.35}$$

where $n(A_c)$ is defined according to Equation 5.31, 5.32 or 5.34 as appropriate for the test image, and where X is the raster for which the distance to A and B is computed. δ_c^2 will sometimes be denoted by $\delta_{r,c}^2$, $\delta_{l,c}^2$, and $\delta_{a,b,c}^2$ to indicate which measure of the cutoff area was used to compute it. The error measure is now averaged based roughly on the number of pixels contributing to the sum in Equation 5.29 rather than on the image size. We thus obtain an error measure independent of both image size and contour length.

In all experiments presented, we use $c = 5$ as the cutoff distance, as suggested in [Bad92a].

5.4.4 Measuring the Error: Practical Considerations

In our experiments, the true distance to the boundary of the image object is computed using a distance transform in the "true" image, which is then scaled to compute the distance in the smaller raster. In this way, we obtain a measure which closely approximates the distance to the object boundary in $\mathbf{R}^2$. We do this because the boundary position is not well defined for the test image objects on a raster. So that we can evaluate our results, we show the response of the $\delta_{r,c}^2(A, B)$ measure for the Bresenham point set. Let B_r be the Bresenham approximation to a circle of radius r, and let A_r be the true circle, the distance to which we obtain in the finer raster. For

the test images described, we have

$$0.034 \leq \delta^2_{r,c}(A_r, B_r) \leq 0.052$$

for $r \in \{10, 20, ..., 100\}$. In general, $\delta^2_{r,c}(A_r, B_s) \approx |s - r|$. Thus from $\delta^2_{r,c}$ we can see how far apart two boundaries are on the average. The same is true for the hexagon and ellipse variations on the δ^2_c measure.

5.5 Parameter Tuning

In Section 5.3.2, the parameters in Equations 5.28 and 5.24 must be tuned to optimize the method. In this section, we consider the criteria used to optimize the parameters, and describe a set of experiments used to obtain their optimal values for images containing discs of varying radii, with added noise resulting in $SNR = 10$ and $SNR_{dB} = 20$. If the shape of the objects or the conditions related to the SNR in the imaging process are known a priori, these experiments may be repeated to optimize the parameters for a specific class of images, or by substituting user input for the simulated user data, for a particular user.

5.5.1 Optimizing the Direction α_k

Recall from Section 5.3.2 that the angle α used in Equation 5.28 is the primary factor in selecting among the candidate paths for V_k. In this section, we seek the optimal value for ν used to define α_k (from which α is derived) based on the weighted sum in Equation 5.24. As mentioned in Section 5.3.2, ν determines the ease with which the user can change direction. A high value for ν increases sensitivity to user jitteryness, while a low value makes it difficult for the user to turn.

In the experiment to optimize ν, we fix the following values:

1. $s = 6$ is the step size for generating the simulated user data, which we use in practice as the minimum distance to be moved before we apply the algorithm. We use the minimum in this experiment because smaller movements result in a more jittery user path.

2. $\varepsilon = 4$ is the minimum error allowed for the user, as is the case in practice for $s = 6$.

3. $\omega = 0.5$ is the value of the parameter in Equation 5.28.

4. $\sigma_{noise} = 10$ is the standard deviation of the Gaussian noise added to the images.

Now because we want to test the effect of reducing the user's ability to turn, we evaluate paths obtained when tracing boundaries of varying curvature. In Figure 5.13, we show the average value of the δ^2 error measure for discs of varying radii. For both dark images on a bright background ($f_0 = 100$) and vice versa ($f_0 = 200$), we performed the experiments including the simulation of user data on 100 images of each

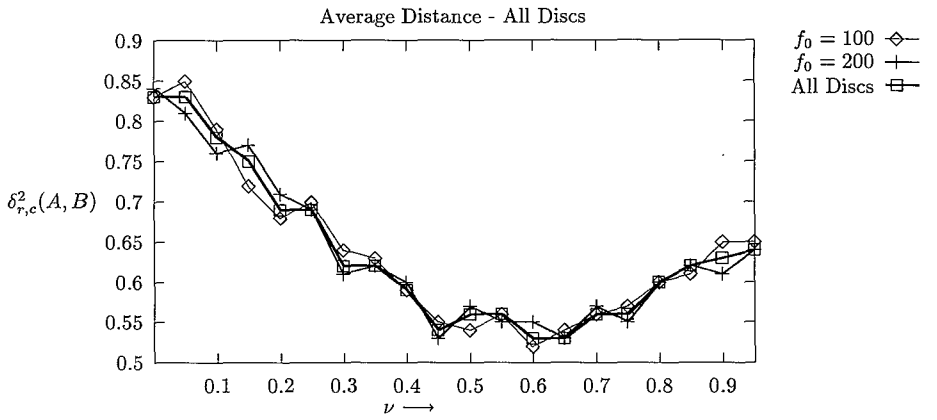


Figure 5.13: The effect on $\delta^2_{r,c}(A,B)$ of varying ν in Equation 5.24. Results are shown for 1000 discs with $f_0 = 100$ and radii $r \in \{10, 20, ..., 100\}$, 1000 discs with $f_0 = 200$, and the average of both. $\sigma_{noise} = 10$.

disc size, each with independent Gaussian noise. Based on the results averaged over all disc sizes, we choose $\nu = 0.6$.

5.5.2 Optimizing the Cost Function $c(x, y)$

We now seek to find the optimal value for ω which controls the relative influence of the user input and image data in the cost function $c(x, y)$ defined in Equation 5.28. In this case, the criteria which we simultaneously want to satisfy are:

A) If the path of the user is near an image object boundary, the direction of which is close to that of α_k, the path we extract should lie on the object boundary.

B) If the path of the user is not close to an image object boundary, the path we select should be the user path (smoothed by the averaging due to using α_k rather than θ_k).

It is clear that high values for ω will result in the satisfaction of the second criteria, while low values encourage the satisfaction of the first. To determine the best value for ω, we seek the value for which the average of the errors for criteria A and B is minimal.

To control the experiment, we fix the values listed in the previous section, substituting the third with

3. $\nu = 0.6$, based on the experiments performed in the previous section.

Let A_r be a binary image which contains the optimal boundary path for a disc of radius r (see Section 5.4.1). Given a randomly generated user data set, suppose we

extract a boundary path $B_r(\omega)$ from the disc image for some ω. Criterion A states that ω should be chosen so that $\delta^2_{r,c}(A_r, B_r(\omega))$ is as small as possible. To be sure this is true in general, for $r \in \{10, 20, ..., 100\}$, we measure the average value of $\delta^2_{r,c}(A_r, B_r(\omega))$ for N random user data sets. The average error is then given by

$$T_d(\omega, N) = \frac{1}{10} \sum_r \frac{1}{N} \sum_{i=1}^{N} \delta^2_{r,c}(A_r, B_r(\omega)),$$

where the $1/10$ results in an averaging over the 10 different radii, and N is the number of data sets for which B_r is generated for each radius.

Now given a random user data set, suppose C_r is a binary image which contains the path generated by applying our method to an image using the optimal value for ν obtained in the previous experiment, and with $\omega = 1$. C_r may be considered the optimal user path for a specific random data set, as it is not influenced in any way by the image data. Let $D_r(\omega)$ be the path obtained if the same user data set is applied to an empty image (with added noise). Then we want a value of ω for which $\delta^2_{r,c}(C_r, D_r(\omega))$ is small. We again want this to hold in the general case, and therefore define

$$T_u(\omega, N) = \frac{1}{10} \sum_r \frac{1}{N} \sum_{i=1}^{N} \delta^2_{r,c}(C_r, D_r(\omega))$$

Because we want the value of ω for which our method performs well both in the presence and absence of a contour, we want the value for which the mean

$$T_m(\omega, N) = \frac{T_d(\omega, N) + T_u(\omega, N)}{2}$$

is minimal. We performed the experiments for $N = 200$, with 100 for dark on bright ($f_0 = 100$) and 100 for the reverse, and thus our results are averaged over 2000 experiments for each value of ω. In Figure 5.14, we plot $T_d(\omega, N)$, $T_u(\omega, N)$, and $T_m(\omega, N)$ as a function of ω. Based on the results we choose $\omega = 0.7$, the value for which $T_m(\omega, N)$ is minimal as the optimal value.

5.6 Evaluation

Having optimized the parameters which influence the cost of a path, we are now in a position to evaluate the performance of our method. We begin by performing our experiments on images containing discs of varying radii and with various levels of added noise. The results shown in Figure 5.15, are averaged over 100 trials for each disc at each noise level. That is, given a disc image of radius r, we add randomly generated Gaussian noise at level σ_{noise}, 100 different times, and each time regenerate the simulated user data for a disc of radius r.

The user step size was fixed to $s = 6$ and allowed error fixed to $\varepsilon = 4$. In practice $r_k = s = 6$ is the minimum move required before we apply the technique. Although this is much smaller than that required theoretically (see Theorem 5.2) our results are

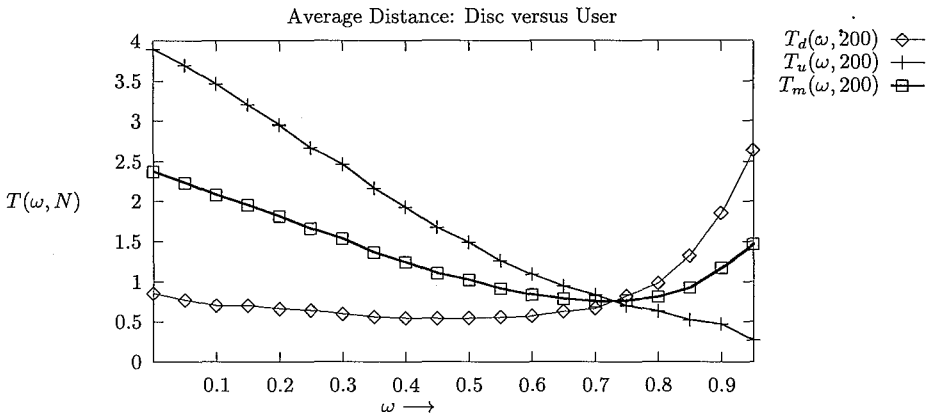


Figure 5.14: The result of varying ω on the ability to locate the contour of a disc $T_d(\omega, N)$, to follow a user $T_u(\omega, N)$, and the average $T_m(\omega, N)$. The results shown are for $N = 200$, $\sigma_{noise} = 10$, and $r \in \{10, 20, ..., 100\}$.

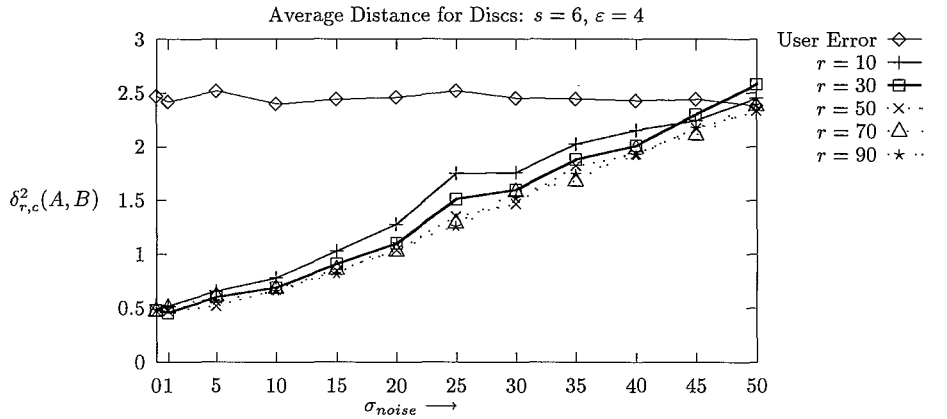


Figure 5.15: The performance of the algorithm applied to disc images with added Gaussian noise with $\sigma_{noise} \in \{0, 5, 10, ..., 50\}$. The average value of $\delta^2_{r,c}(A, B)$ is plotted for a range of radii r. Here we have a user step size of 6 and the allowed error $\varepsilon = 4$. The average error of the input data is also plotted.

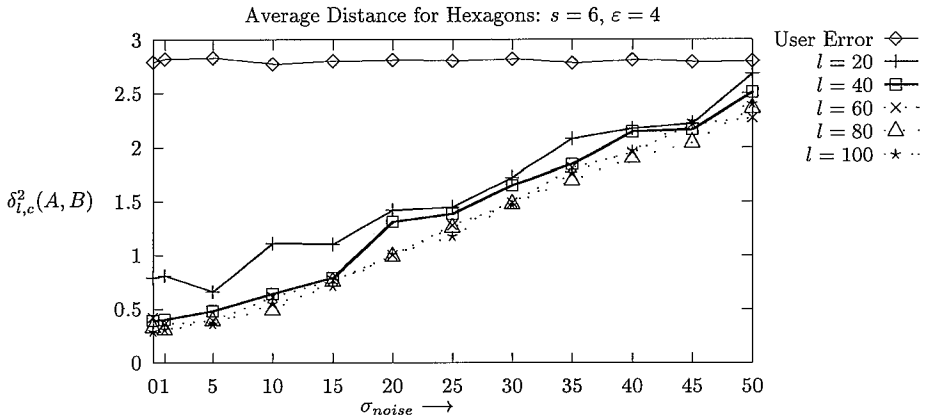


Figure 5.16: The performance of the algorithm applied to images of hexagons with added Gaussian noise with $\sigma_{noise} \in \{0, 5, 10, ..., 50\}$. The average value of $\delta^2_{l,c}(A, B)$ is plotted for a range of lengths l. The user step size is 6 and the allowed error $\varepsilon = 4$. The average error of the input data is also plotted.

very good. This is because in general, the extreme cases handled in Section 5.2.2 will not all arise simultaneously. Smaller values for the step size are advantageous because they allow the user more control, and feedback can be provided more quickly.

The experiments were performed both for dark objects ($f_0 = 100$) on a bright background ($f_1 = 200$), and vice versa, with no significant difference in the results. The average of both experiments is shown in Figure 5.15. The results show a linear decrease in the performance of our algorithm in response to added noise. When the curvature becomes very high $\kappa = 1/10$ ($\delta = 10$), the performance starts to deteriorate, as was predicted (Theorem 5.2). Up to a significant noise level ($SNR > 5$, $SNR_{dB} > 14$), the resulting boundary is within one pixel of the true object boundary, even when the curvature is high.

In Figures 5.16 and 5.17, the results are shown for hexagons with a range of side lengths and for a range of ellipses. The results show that although the parameters were tuned for discs, the method works well for a variety of object shapes.

The actual boundaries obtained with our method in practice are far better than the experiments here indicate. This is because the error in the user angle θ_k, generated by the random data used in our experiments is, in general, far worse than that made by a user. Connecting the dots in Figure 5.11 will illustrate this point. In Figure 5.18, we show boundaries obtained as a user traced some objects in medical images.

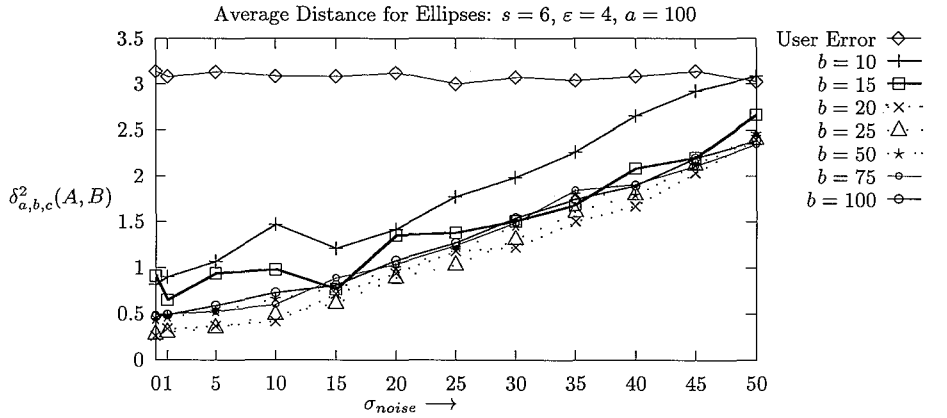


Figure 5.17: The performance of the algorithm applied to ellipse images with added Gaussian noise. The average value of $\delta^2_{a,b,c}(A,B)$ is plotted for $a = 100$, and $b \in \{10, 15, ...100\}$. The user step size is 6 and the allowed error is $\varepsilon = 4$. The average error of the input data is also plotted.

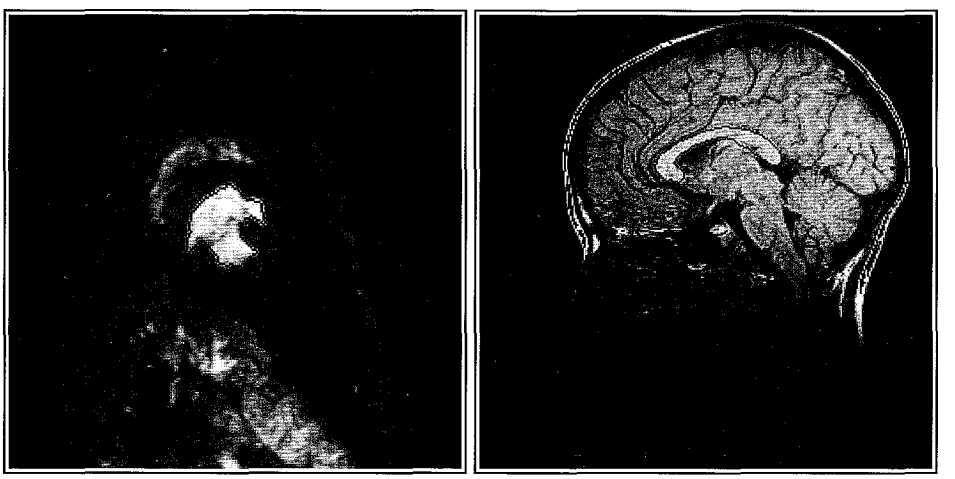


Figure 5.18: Tracing in practice: On the left, the left ventricle in the heart of a dog. On the right the corpus callosum.

5.7 Conclusions

We have presented a methodology with which a path acquired from a user tracing an image object boundary is interpreted and subsequently corrected to follow the correct path of an image object boundary. We present a theoretical foundation for the method developed and show it to work well in practice. Our method may be used both for segmenting unknown images and for constructing image object models for a particular problem domain. The technique is suitable for a specific, but widely applicable image model, and the development of the cost function and the parameter tuning method shows how this may be adjusted for other image models. The experimental methods used may also be employed to optimize the method for a particular user, and thus make the method suitable for incorporation in an adaptive interactive environment for image segmentation.

The method is particularly suitable for the specification of image object boundaries in the presence of various forms of disturbance which frequently interfere with the segmentation process. It is robust in the presence of significant noise in the image, or user jitteryness. This is due to the cost function which suppresses the response to image transitions due to intersecting or nearby parallel object boundaries, and shows a preference for smooth paths.

The dynamic programming method developed to extract the object boundary based on the minimal cost path is substantially faster than similar methods, with no penalty in accuracy. As we had hoped it is fast enough to correct the path of a user while it is drawn, and thus provides the user with the sense of working with a magnetic pen.

Chapter 6

Concluding Remarks

6.1 Conclusions

In this thesis, we introduced an approach to image segmentation called *supervised boundary formation*. With this approach, an image is partitioned based in part on user sketches of image object boundary paths. For two general input models for path specification, namely *connect-the-dots* and *free-hand drawing*, we investigated problems of user interpretation and correction. The investigation leads us to conclude that issues arising in interactive segmentation can be addressed in a formal manner, and that interactive techniques are an effective tool for obtaining a correct partitioning of an unknown image.

User error model

To facilitate the task of user interpretation, we performed experiments to measure user errors in the specification of corners on polygonally shaped objects, as described in Chapter 2. For each corner, we measured the error in distance, corner angle, corner orientation, and corner scale. For all users, the level of the first three errors were uneffected by the tested image variations. We tested four variations, namely images with and without added noise; dark objects on a bright background versus bright objects on a dark background; ramp edge and corner model versus a roof model; and two different shapes. It turned out that the distribution of the user error in corner scale varied depending on the object shape, and somewhat depending on the presence of noise. This was especially true for one user.

The polygonal objects were made up of line segments in the range $\{15, 30, 45, 60, 75\}$, and corners of angle in the set $\{45, 90, 135\}$ degrees. Both the mean and variation of the distance error differed as a function of user but not as a function of line length. The magnitude and the variation of both angle errors (corner angle and orientation) decreased for all users as a function of line length. The corner scale error differed for the various users, and per user for the different objects.

The bias of the corner angle and the bias of the corner scale both proved to fit a Gaussian distribution for all users. Although we did not find a well known statistical

distribution for the errors in distance and corner orientation, both were well behaved for all users in the following sense. For each user, the mean μ and standard deviation σ of the user error was calculated, and the user error was less than $\mu + 2\sigma$ in more than 90% of the cases. This applies to all users for both the distance and orientation measures.

These experiments lead us to conclude that the type and degree of errors made by individual users in the specification of corner points, can be predicted. Based on the results, we were able to develop a user dependent error model for corner specification. This model consists of the maximum expected distance between a user defined point and the corresponding point on the object boundary, the maximum difference in corner orientation, and the distributions of the user corner angle and scale errors.

Limitations The performance on the four Brodatz texture images varied significantly among the users. In some cases, numerous extra points were inserted, and in other cases, a number of corner points were not specified. Because the users were presented with the texture images before the other images, we may thus conclude that if users do not possess a priori knowledge of the expected image contents, they do not perform well for arbitrary texture differences.

Open Questions

Other geometric models – Suppose a user were to specify a sequence of image locations intended as part of a curve rather than as a set of polygonal corner positions. Then a different corner model must be developed, which allows one to define limits on the difference between the user defined curve and the boundary curve. This requires the definition of an appropriate set of error measures for curves.

Texture – The methods described for user modeling in Chapter 2, may prove useful for modeling expert users, such as radiologists, working with known images. Further, testing users with artificial textures may provide additional insight.

Inexact polygon matching

To match the user-defined polygons with the model polygons for the experiments in Chapter 2, we developed an algorithm to perform inexact polygon matching, which produces a best fit of two polygons. The key contribution which distinguishes our approach from other polygonal object matching methods, is the separation of the node pair match evaluation from the global matching decision. This allows us to guarantee that with respect to a given cost function, the match of the polygons produced with our algorithm is a global optimum match.

We provide a general framework for the development of cost functions for applications which tolerate any subset of affine transformations. In addition to the cost function used to match the user and model polygons generated by the experiments in Chapter 2, we develop a cost function suitable for robotics applications in which the

camera position is fixed. In [GAW93], this algorithm provides a reliable mechanism for deciding whether an object in an imaged scene matches one extracted from a database.

The algorithm proved to find 96% of the point pairs for a polygon with a random distortion of itself, when up to 4 (of the 17) points were removed from either or both polygons. The A* matching algorithm is efficient in its exploitation of cyclic characteristics of polygonal data sets. The complexity of the algorithm is comparable with geometric hashing in the worst case [GG92], and two orders of magnitude faster $O(n^4)$ on average. We have shown the basic scheme to be extendible for matching polyhedra in three dimensions.

Model-based corner detection

To correct user errors made in the specification of corner points on polygonally shaped objects, we developed the model-based corner detection method described in Chapter 4 of this thesis. We derived a scheme which enables the evaluation of an image region based on the geometry of a corner in a polygonal shape specified by a user. In particular, we examined the geometry of an image corner in relation to the geometry of a corner specified by a user. For a given user error model, and an image feature measure λ, we examine conditions under which we can extract an image corner model with respect to the measure λ. We show that if the window size required to measure λ and the user error in distance are both sufficiently small in relation to the corner angle and scale, then we can establish the presence of a ramp or roof corner as measured by λ. To illustrate the effectiveness of the method, we show that if λ measures the grey value intensity, we can distinguish four corner types, namely dark and bright ramp and roof corners. Specifically, for corners with angle in the range $[3\pi/8, 3\pi/4]$, we determine the correct model in more than 95% of the cases. We use the model to localize the corner to pixel accuracy.

Limitations A key problem in the model-based corner detection method is the modeling and subsequent localization of corners with small angles ($\alpha \leq \pi/4$). Because the size of the region used to investigate the image function in the cone of the corner decreases as the corner angle decreases, reliable modeling of small corners is hindered. This indicates a different approach to obtaining a segmentation model for small corners must be developed.

Magnetic contour tracing

In Chapter 5, we developed a method for real time correction of a user specified boundary path. The interactive model supported is free-hand drawing, but rather than producing the path actually traced by the user, we extract a best approximation to an object boundary path in the immediate region, with a dynamic programming method. The set of candidate paths and the costs that determine which among them is selected, are based on the location and geometry of the user input, and on an a priori model of the image function near object boundaries. For the boundary model we used, the gradient magnitude of the image function is required to be high along

the resulting boundary path, which is also required to be connected and smooth. The image boundary model must be known a priori to extract the path in real time.

In this chapter, we developed a set of conditions under which a boundary path can be accurately extracted based on user input or any other point set. In particular, we investigated the relationship between the image object boundary path and the user defined path, and derived a formal set of conditions (Theorem 5.2), under which our method is guaranteed to work, assuming our model of the image function on the object and background regions is correct.

For interactive segmentation, the cost function to be minimized, is defined as a weighted combination of a user term based on the direction of user movement and a boundary (gradient) strength term. The influence of the weights was evaluated, and the set which minimized the errors made over the length of a boundary was selected. We used a random error generation method to simulate the user.

The magnetic ink method was shown to produce a good approximation of a boundary path for a range of test images containing discs, ellipses and hexagons. Specifically, for signal to noise ratios as low as $5SNR$, the path produced was within one pixel of the true boundary path, on average. This was true for a far wider combination of input and boundary paths than satisfied the theoretical conditions, which makes the method widely applicable in practice.

6.2 Discussion

Extensions For supervised boundary formation to become an effective tool in the segmentation of unknown images, a variety of input/correction models are required. Ideally, the path specification input tools found in standard drawing packages should be available to the human expert setting out to segment an image. For each such input method, a range of correction options should be supported.

In addition to correction of corner locations described in Chapter 4, modeling of image boundary curves and subsequent localization of the boundaries, based on connect-the-dots user input, must be addressed. Such modeling requires a user error model be derived for curve specification. Meanwhile, if a model of the image function near the object boundary can be determined, various forms of correcting the user defined boundary path must be investigated. For freehand drawing, it may be useful to derive a model of the image function in the object and background regions before correction can take place. The real time magnetic ink correction introduced in Chapter 5 might then be replaced with some form of post-specification correction.

Segmentation of Three Dimensional Images The approach to interactive segmentation advocated in this thesis cannot be directly applied to the segmentation of three dimensional images. This is because the direct manipulation paradigm is based on a *what-you-see-is-what-you-get* interaction model. That is, the user is assumed to perform manipulations in a fully representative view of the image.

Viewing of three dimensional images in two dimensions requires either a user be presented with some two dimensional subset of the image or a two dimensional visu-

alization of the three dimensional image. The latter is problematic because visualization techniques are based on a predefined segmentation model, so that what the user views is not the image data, but a manipulated view in which the certain characteristics have been used to distinguish the object from its background in the view (e.g. [LC87, DCH88, Lev88]). If the visualization is successful, then the image can be segmented automatically, and if not, then it will not help in the segmentation because it is based on the wrong model.

Views containing some subset of the image data, such as a single image plane or the maximum or minimum value in some direction, are also problematic because the user has an incomplete view of the image. In this case, the task of visualization is left to the user.

Still, interactive techniques can sometimes play a useful role in the segmentation process when sufficient a priori knowledge is available both of the segmentation model, and the views which should be presented to the user for manipulation. Examples of effective interaction techniques in the segmentation of specific sets of three dimensional images can be found in [FLP89, LP90, HSvdV+77].

Finally, if a representative plane can be extracted from an image for viewing, one might apply magnetic contour tracing or model-based corner detection to find the significant boundaries in that plane. The results might then be propagated to other planes using model-based deformable contour methods such as those described in [SD89, WSSD93].

Real time image function modeling With respect to human computer interaction principles, the method developed in Chapter 5 is extremely attractive because it provides immediate feedback to the user. It requires, however, that the properties of the image function in the region about the image object boundary be known a priori. Though it can be employed to segment difficult images such as those in Figure 5.18 when an appropriate segmentation model is known, a model of the image function which can be used to locate the boundary cannot be derived as the user draws. This limitation is not only due to the complexity of the method required to model the image function, but due to an insufficient geometric model (a short line segment). A hypothesis of the shape and size of a boundary section, comparable to that used to model the corners in Chapter 4 can only be extracted after larger sections of the boundary have been specified.

Learning from an Expert Perhaps the most relevant question arising from the work in this thesis is how much one can learn about a given image, based on a simple user sketch. In Chapter 4, we presented a method for examination of an image region about a section of a user sketch. If a sufficiently large region can be investigated near a path section, then the segmentation parameters, including an image function model can be obtained from the user input.

Human experts can be relied upon to provide an approximation of the geometry of an object boundary. Asking an expert to specify the characteristics of the image function in the object and background regions in a form which can be used to quantify

the difference between the two is unrealistic, unless the model is trivial (for example, dark object on bright background). If this information can be deduced from the geometry of the user defined path, then supervised boundary formation will provide a powerful tool in the development of segmentation models for new applications.

6.3 Concluding Remarks

In both Chapters 4 and 5, we developed a theoretical framework in which the relationship between the user defined path and the true path of the image object boundary can be inspected. For both input/correction models, we defined specific conditions, for which we can guarantee the object boundary path could be found given the user input path.

In both cases, the methods developed prove to work well for a far larger set of user/boundary path pairs than was theoretically guaranteed. This is because to *guarantee* the boundary path would be extractable based on the user defined path, we had to assume the worst case. In practice, of course the worst case is an exception to the rule, and the boundary path can be found even when the theoretical conditions are not met. Still for critical applications, it is essential to know under what conditions correct results can be guaranteed, and the arguments presented in Chapters 4 and 5 can be used for this purpose.

In summary, the results in this thesis support the notion that supervised boundary formation is an effective approach to the problem of segmenting unknown images. We have shown the human interface issues involved with object boundary specification can be dealt with in a systematic fashion. This removes much of the uncertainty involved when a human expert plays a direct role in the segmentation process. The results for the corner detection method introduced in Chapter 4 also suggest that substantial knowledge about the object boundary, both in terms of its form and the behavior of the image function in the immediate region, which a user is unable to express directly, can be extracted from a user specified path. Ideally, this knowledge may be exploited to construct object models for the automatic segmentation of similar images.

Appendix A

GRIP - A GRaphics library for Image Processing

A.1 Introduction

Vision problems do not, in general, lend themselves to easy automation [BB82]. While good model based systems have been developed for a variety of specific applications, no general methods have been developed which can be applied to an arbitrary analysis problem. Thus, in producing new applications, and in dealing with images which cannot yet be analyzed with fully automatic methods, it is often useful to allow an expert to direct some stages of the analysis interactively.

Direct manipulation has been shown to be a particularly effective interaction style in a number of user interface studies [Shn87]. The key characteristic of a direct manipulation interaction tool is the feeling on the part of the user that she is operating directly on the object of interest, rather than requesting via some form of dialog that a desired operation be performed. A good example is the act of driving a car using a steering wheel and gas pedal, as opposed to giving verbal directions in the role of a passenger. Because image analysis is a highly visual application domain, direct manipulation is a particularly suitable style of interaction. Image objects to be evaluated can be easily indicated with a pointing device rather by some means of verbal description. In practice, this concept has been shown to work well in Athena [vVYtK+89], a Macintosh based interactive karyotyping system. In that system, an expert is offered the opportunity to interactively indicate that two objects should be joined or that a single object should be split in two, as part of the segmentation process.

To support interactive methods in image processing and analysis, a graphics library is required which can cooperate with an image analysis system. For example, if one wants to split an image object, a path must be specified which indicates how the object should be divided. The user specified path must be superimposed on the image display as it is drawn so that the user can decide whether the resulting division is the one intended. If the user is satisfied, the path must be used in the segmentation process to produce two image objects from the single one, upon which the user drawing was superimposed. To achieve this functionality, communication between the interactive

graphics and the image processing system components is essential. Cooperative use of input devices such as a mouse and keyboard, and output display devices must be supported. Further, information about the data sets produced by each system component must be shared, so that a graphical system data component, such as a path specified by the user can be used to manipulate a image analysis system component such as the object to be split.

Image processing differs from other application domains in the sense that most data that a user wants to manipulate is a subset of the displayed image, and while visually available, it is not represented geometrically or textually. Thus, tools are required which allow the user to describe and manipulate the image subset of interest. Direct manipulation techniques for specification of an arbitrarily shaped image regions may therefore play a key role in many image analysis applications.

Currently, no systems exist which support the development of direct manipulation user interfaces for image analysis. The Athena system was built from scratch, and cannot be extended in any way. The Image 1.52 package [oH93] on the Macintosh provides direct manipulation tools for experimental purposes, however, it lacks support for application development. Many systems, which support experimental image processing as well as application development exist [oAP88, tKvBS+90], but do not provide support for direct manipulation. A number of general purpose user interface toolkits [You89, Com85], contain useful graphical components such as menus and sliders which are useful for image processing applications. However, these systems do not provide support for sharing a display space as required for the graphical manipulation of visible image data.

To address these shortcomings, we designed GRIP, a graphics library which supports the construction of direct manipulation tools for image processing and analysis. Such tools are useful in an image analysis system, and essential for developing interactive image analysis applications. By combining GRIP with a general purpose user interface toolkit such as that described in [You89], a high level toolkit which supports interactive image analysis applications can be constructed.

In the next section, we consider the role that direct manipulation tools can play in the various components of an image analysis problem. Key design issues which must be addressed if such tools are to be constructed, are considered in Section A.3. This is followed by a functional overview of GRIP, which provides the necessary graphics support to build such tools. Because GRIP provides functionality similar to that of the standard graphics system GKS [HDGS83], we list the similarities and differences of the two systems in Section A.5.

A.2 Direct Manipulation and Image Analysis

GRIP is intended to provide tools for solving image analysis problems which cannot (yet) be solved with fully automatic methods. In particular, GRIP can be used to construct direct manipulation tools, which cannot be built using general purpose interactive toolkits such as [You89]. In this section, we outline the components of a

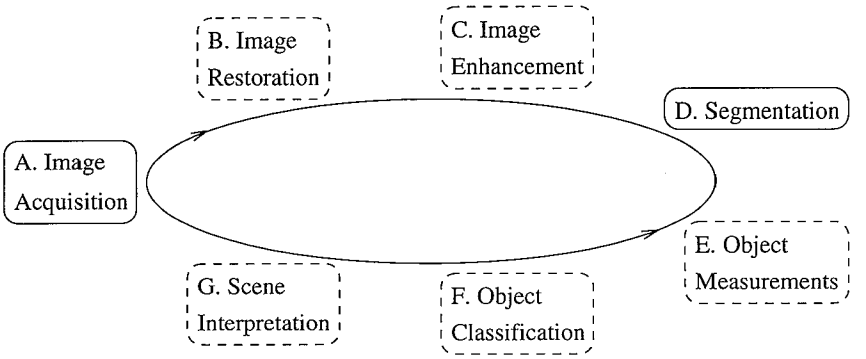


Figure A.1: The components of a general image analysis system. Depending on the application at hand, one or more steps with a striped outline may be skipped. Image acquisition and segmentation are fundamental steps in nearly every image analysis problem. Traditional data driven image analysis corresponds with clockwise movement through the steps starting with image acquisition.

general image analysis scheme, and consider the role direct manipulation interactive techniques can play in extracting information from a scene.

Figure A.1 shows the major components of a general image analysis system. In traditional *bottom up* or *data driven* image analysis, the steps are performed one for one in a clockwise manner, starting with image acquisition and stopping when the intended information has been acquired. More recently, the benefits of *top down* or *model driven* analysis schemes have been recognized. In this case high level expectations about the image contents may be used to direct the low level processing. In these systems the steps need not be followed in a strictly clockwise or counterclockwise fashion. [GAW93] contains a nice example of a model-based object recognition system.

Involving an expert user in the analysis process is similar to using a model-based scheme. Presumably, the user knows what information is to be extracted from the image, and what part of the image is relevant. There are two key roles a user can play in the analysis process.

Decision making: A user can decide what steps should be taken when. For example, if there is insufficient contrast in the image to visually separate an object from its background, the user may select an enhancement procedure, or request a recalibration of the camera and a fresh image acquisition. Selecting parameters for a given operation also falls under the decisive powers of a user.

Analysis operations: A user can perform or help the system perform individual steps in the analysis of an image. For example, the user may recalibrate the camera prior to image acquisition, or trace the boundary of an image region in which measurements should be performed.

The menus, sliders, dialog boxes and other interactive components found in general purpose toolkits provide sufficient support for the first set of operations, as can be seen in systems such as SCIL_Image [oA91] for image analysis. We are therefore concerned with the tools required for the user to interactively aid the system in specific analysis steps.

Now, consider the components of an image analysis system depicted in Figure A.1. Both image acquisition and restoration are global operations in which the role of the user, beyond decision making, is limited. Image processing systems such as Adobe Photoshop [Ado90] in which the user can manipulate the image data using various painting tools have become increasingly popular in recent years, and fall in the category of image enhancement methods. Because these tools manipulate rather than evaluate the image data, however, they are not suitable for image analysis systems. In fact, application of such tools makes objective analysis of the image impossible due to a loss of information in the enhancement process.

Segmentation of an image into meaningful parts is the most difficult task to automate in image analysis [Sme91]. Simultaneously, it is an essential element in almost all image analysis applications.[1] The precision of all subsequent measurements on image objects depends directly on the accuracy of the segmentation procedure. Segmentation techniques are, in general, geared toward identifying the region of an image associated with an object, or by identifying the path of the object region's boundary. Given either description, the other can be computed directly. Both approaches, when applied without a priori knowledge of an object's characteristics are severely limited in their applicability to new analysis problems. In general, a segmentation method must be based on a model of how the image object differs from its background. Models which have been developed to date (see for instance [RA77, HS85]) are only applicable to a limited set of image analysis problems.

If an expert is provided with drawing tools such as those found in general purpose drawing packages (cf [Cla92]), then many images for which no appropriate segmentation models have been developed can be segmented interactively, allowing further object analysis to take place. Optimally, the user sketches should be taken as rough outlines and corrected (see for examples Chapters 4 and 5). Alternatively, results obtained with automatic methods might be corrected. Splitting an object in two based on a user defined path, as in [vVYtK+89], is a useful example. Similarly interactive techniques can be used to connect or split edges to produce complete object boundaries if automatic edge detection techniques fail to do so.

Some object measurements which are difficult to obtain even when segmentation has been successfully completed, might be extracted interactively. For example if the distance between two image objects is of interest, the user can indicate the objects under consideration with simple pointing and clicking techniques. If an image object is to be matched with a graphical database model for identification, the user may select the appropriate model from a database. The actual fitting might be assisted if the

[1] Analysis of images known to contain a single texture provide a rare example of when segmentation is not one of the analysis steps.

user can drag a copy of the model to the object region of the image and interactively modify its geometry (scaling, rotation, etc) until it matches the object in the image.

Specifying a region of interest with the operations available in most image analysis systems is awkward and inaccurate at best, requiring the user to estimate and type in the coordinates of a rectangle surrounding the region. It is often an important step in reducing the amount of superfluous data extracted in the analysis of an image. Direct manipulation techniques such as those found in drawing packages, for specifying regions can therefore be very useful in many image analysis problems. In image database applications, the ability to specify arbitrary geometric patterns is essential for image database searches [GS92].

A.3 Design Considerations

An environment which enables the development of the interaction techniques described in the previous section must support data communication and shared control of devices among the graphics and image processing system components. At the most basic level, all applications which display both graphical output and an image require that information be exchanged about modifications to the display contents so that screen updates can be handled correctly. Tools for correction of user defined boundaries such as those in Chapters 4 and 5, clearly require significant interchange of data. The user defined graphical sketch is used to drive the image processing, and the results (a 2D path) must in turn be presented to the user. Likewise, the tools provided in Athena [vVYtK+89], drive the low level image operations based on graphical user input.

The key elements in GRIP which support the necessary cooperation between the graphical and image processing components are incorporated in the *workstation* structure. As in standard graphics systems, the term workstation is used to refer to the data structure which describes the physical and abstract input and output devices. The workstation keeps track of the required physical display details such as its dimensions and depth, and input device characteristics such as the number of buttons on the mouse. For updating and interactive manipulation, a list of displayed graphical objects is also maintained on a workstation. This data structure can also be used to support of printers and storage files, neither of which has input devices. The workstation concept described here is common to many graphics packages including the GKS [HDGS83] and PHIGS [Gra88] standards.

Because our primary concern is the support of interactive image analysis, we introduce the concept of a *shared* workstation. By *shared*, we mean the structure supports cooperation with applications and user interface tools for output to and updates of a shared display space. Mechanisms are provided in the system so an application can update the display space before or after GRIP update operations are executed.

Another essential aspect of GRIP for cooperative work with other systems is the *external* input model. In sharp contrast with systems such as GKS and PHIGS, GRIP responds to notifications from another system that input has occurred. After handling the input, control is returned. In standard graphics systems, full control of all input and output devices is assumed. This is one of the key problems in using those systems

in cooperation with a window system or an application which may assume the same control.

In the remainder of this section, we consider the key aspects of GRIP which enable cooperative use of the input and output mechanisms. We first define a number of terms which arise in the discussion.

Terminology

Much of the terminology used in this appendix is common in computer graphics literature, and definitions can be found in standard graphics texts such as [FvDFH90]. Terminology which is essential for understanding the description of GRIP is briefly defined here, and in more depth in Section A.4.

The appearance of graphical output on a workstation is controlled in part by a set of geometric *transformations*. The transformation mechanism used in GRIP is analogous to that used in GKS and PHIGS. A user specified rectangular area in a user defined, world coordinate coordinate system (WC) is mapped to the display coordinate system (DC). The mapping is done via the normalized device coordinate system (NDC) defined by the square $\{(0,0),(1,1)\}$. See Section A.4.4.

Graphical objects are formed by assigning graphical *primitives* such as the polyline, polymarker and text primitives to a *graphical object* or *group*. A group is similar to a segment in GKS in the sense that primitives in a group can be identified, manipulated and deleted as a set.

A *primitive attribute set* contains information which controls the appearance of primitives which use it. Some examples of primitive attributes are color, line thickness, line style (e.g dotted, dashed), marker style (e.g. circle, star), and text font.

A set of *group attributes* is associated with each graphical object. These define the group transformation, priority, and primitive attribute set for the object. The *group transformation* determines the size, location, and orientation of all primitives in the group. The *priority* determines the visibility of primitives in the group in relation to primitives in other groups. If the display area of primitives in two or more groups overlap, the priority determines which primitives will be "on top" and thus selected first in pick operations.

A full description of these system components is found in Section A.4.

A.3.1 The GRIP Input Model

In order to coordinate activities with underlying packages, and to work in cooperation with a user interface toolkit or management system, GRIP uses an *external* input model. By this we mean that the application programmer controls the input devices using the capabilities and tools of the window system, or the user interface toolkit (which might be part of an image analysis package). By means of a call-back mechanism GRIP is notified when specific events occur. The programmer defines functions for those events which should be handled if some event occurs, and informs the application via a function naming mechanism. Presumably, the system which controls the input will call the appropriate function under a generic name when the event occurs.

GRIP supports the *locator* and *pick* input classes, and assumes the *choice, keyboard, stroke* and *valuator* input classes are supported by the user interface toolkit used, or by the application programmer. The locator input device returns the world coordinate position of a device location, and the pick input device returns an object identifier for the nearest object to a device position. Which object is "nearest" depends on the position, the object priorities (see Section A.4.2, and on parameters which may be set by the programmer. For example, a programmer can specify that a filled area be detected only if a location is within its boundaries, and that a marker have a large neighborhood in which it is detected. A marker may then be picked even if a filled area is actually closer. The object priority determines the order in which objects of the same distance will be inspected. For interactive applications, the device position usually corresponds to user pointer location, but a programmer may of course pass any position.

Whether input is handled in event, request or sample mode is up to the programmer and the toolkit being used to create the user interface. This is in sharp contrast to the nonextensible internal input model used in GKS and PHIGS. It is not possible to make use of modern user interface tools in applications based on those packages, or to extend the input model beyond its original design [HM91]. This is unfortunate as increasingly sophisticated dialog tools with buttons, sliders and text manipulation support become widely available [Com85, You89].

For the locator input class, echo functions are provided which allow the programmer to provide feedback to the user in a number of common ways. For indicating a change in functionality, a variety of cursors are available, and to give feedback as a path is interactively defined, the rubber-band line and rectangle functions are available. Mechanisms are available for the programmer to use externally defined echo functions, so application related feedback can be supported. For the pick device class, highlighting of a selected graphical primitive or group can be performed by changing the primitive attribute set associated with the group (see Section A.4.3). This allows the programmer full control over the feedback mechanism for pick related functionality. For example, an object picked for deletion might be highlighted differently than one picked for scaling.

The model described here is essential for a graphics package to work effectively with other systems. Rather than providing a fully defined input model, mechanisms are provided for cooperation with an arbitrary input model. Interactive tools have been created using the GRIP input tools in cooperation with the X event model [SGN88], the X Toolkit and the X Widget Set [You89], and with the SCIL_Image package [oA91]. In all cases, the external model used in GRIP has proven to work harmoniously with the input model of the system used. A detailed description in terms of the input related functions is provided in Section A.4.5.

A.3.2 The NDC Grid

Two operations which depend on the ability to identify the graphical objects which are displayed in the immediate neighborhood of a screen location are required for an interactive system. The first, *hit detection*, is used to identify the closest displayed

object in the vicinity of a screen location. This is necessary for pointing and dragging techniques, or any other form of interactive object manipulation. To support these interactive techniques, the object to be manipulated must be identified. The second operation, *damage repair*, is used to regenerate the contents of a display area, when damage has occurred. This may be necessary when an object is drawn, moved, or erased, or if the priority of an object is modified. This requires the immediate identification of all displayed objects to which damage may have been inflicted. For direct manipulation techniques, it is essential that hit detection and damage repair be performed immediately, to maintain a sense of control on the part of the user.

For rectangular objects, the bitBlt procedure described by Pike in [Pik83], has proven an effective means of damage repair. For arbitrarily shaped graphical objects, however, the situation is far more complex. Many graphical systems, including GKS, postpone damage inducing operations until a complete redraw of the screen contents is requested. This makes the fast display of small changes such as those caused by deleting a small object impossible, and is therefore not suitable for highly interactive environments. Another approach introduced by Bramer in [BS81], is to erase an object by redrawing it in exclusive OR mode. This causes parts of objects which intersect it to be redrawn as well, and thus may do more damage than it repairs. Even if no other objects are displayed, this operation will leave the display of the underlying image modified, and can therefore not be used when the display area is to be shared.

Warner introduced another approach in which the smallest enclosing rectangle for each graphical object displayed on the workstation is computed [WK79]. If an object is erased, the entire rectangle is cleared, and any graphical objects whose enclosing rectangles intersect that of the erased object are redrawn using it as a clipping region to prevent further damage. This method, while attractive, has the disadvantage that many objects which never intersect the erased object may have to be redrawn. In the worst case, the erased object is a diagonal line which extends from one corner to another of the display space. If it is erased, the entire picture must be redrawn. If there is an image in the display area, this is a costly and visible operation.

In the tiling method introduced by Slater [SDS88], the display space is divided into an $N \times N$ grid of tiles, each containing $w \times h$ display pixels. When a primitive is drawn, some calculations are performed to determine the set of tiles it intersects. This tile set is then used for the same purposes as the enclosing rectangle described above. Although elegant, the tiling method is limited by the fact that the grid is defined in display coordinates. Its major drawback in a windowing environment is that a window resize requires the entire tile grid to be destroyed and rebuilt, which includes recalculation of the tile intersections for all primitives.

To address this problem, we introduce a tiling grid on the normalized device coordinate (NDC) space defined by the rectangle $\{(0,0),(1,1)\}$. This allows us to define an $N \times M$ grid in terms of a world coordinate system, which for image analysis applications might be expressed in terms of image dimensions. To support both hit detection and damage repair, we maintain a list of the graphical primitives which intersect each grid element or tile. This requires that when a primitive is drawn, the tiles it intersects

be computed. The primitive identifier must then be stored in each intersected tile. When deleted or geometrically modified, it must be removed from each tile's list.

Hit Detection

When a request is made that the nearest graphical object to a given location be determined, a list is made of all primitives that intersect the grid element to which the device location maps, as well as those primitives which intersect the eight connected neighboring grid elements. Selection among the list of primitives depends on a number of factors including viewport and object priority as discussed in Section A.4.

Damage Repair

In GRIP, the contents of the display device are updated whenever viewed data is modified. Immediate updates are imperative for direct manipulation tools, as the user must have instant feedback on the consequences of interactive input. A number of data modifications in GRIP might inflict *damage*, and thus require *repair*. Drawing, erasing or moving a graphical object requires the area effected by the modification be redrawn. Any modification to a primitive attribute set requires that all primitives using that set be redrawn. Most modifications to a groups' attribute set require a redisplay of the group. Finally, if changes are made to an underlying image, graphical objects in the area modified must be redrawn.

A key problem for GRIP is that the display of an underlying image must be kept intact when graphical display contents are updated. Because copying image data to a display is a costly operation which depends on the dimensions of the region to be copied, reducing the size of the update region is essential for efficient damage repair. To achieve acceptable performance, we use the NDC grid to limit the area of modification to the set of tiles effected by the damage inducing operation.

As an example, consider what happens when an object is erased. Associated with each primitive in the group, is a list of tiles intersected by the primitive which is calculated when it is first drawn. The list of tiles for the primitive group to be erased is assembled. For each tile in the list, the object name is removed from the list of graphical objects for that tile, and a list is made of all other objects intersecting that tile. A mask is created for updating the underlying image based on the tiles through which the object passed. This is similar to drawing in the background color to erase a primitive. The objects which intersected the grid elements are then redrawn in order of priority using the set of grid elements as a clip mask.

Corresponding grid based methods are used for other operations which require display updates. Using the grid for these operations reduces the area to be modified to that actually effected by the change, and therefore allows them to be executed at an acceptable speed for interactive purposes. In the context of image analysis, we define the grid in terms of image coordinates, and require the divisions to fall between pixels. Calculation of the image region associated with a tile set is thus efficient and unambiguous.

A.4 GRIP – A Functional Overview

The graphics library GRIP supports the development of applications for which two dimensional interactive graphics functionality must be combined with image analysis functionality, or with some other application functionality which requires access to the input and/or output devices. Because GRIP has been successfully implemented as part of the SCIL_Image package for image analysis [oA91], we describe parts of the system using its incorporation in SCIL_Image as an example. The library is, however, designed to share input and output devices with an arbitrary application.

Graphical output appears in a window opened and controlled by GRIP or in a window opened by an application with which it shares its display space. In SCIL_Image, a GRIP display structure is initialized with useful defaults in each window containing an image. Therefore graphical output can be sent to any or all image windows with the use of the GRIP functions. Update of the graphical output when window exposes and resizes occur, is handled automatically (in the routine which updates the image display SCIL_Image calls a GRIP redraw function). Further, when a new image is generated in a display window containing GRIP output, this output will appear automatically "above" the new image contents. The image sharing display space with GRIP is not modified, only the display is.

A.4.1 Workstations

GRIP display objects are made visible on a graphics *workstation*. For interactive image analysis applications, GRIP supports both a simple window and a *shared* display window as workstations. The input and output devices of a simple workstation are completely controlled by GRIP. A shared workstation allows another application to share control over the workstation contents.

Before output can appear on the display, a workstation must be opened with either *GrOpenWs()*, or *GrOpenShWs()*. These functions initiate the necessary data structures to manage graphical output on the workstation. For each image display window created in SCIL_Image, a GRIP workstation identifier is reserved and *GrOpenShWs()* is called to allow output to be superimposed on the display of the image.

GRIP determines which of the open workstations should display a graphical object based on a workstation activation scheme. All workstations that are active when a graphical object is initially created will display the object. Workstations are activated and deactivated with *GrActWs()* and *GrDeActWs()*. Further, *GrDeActAllWs()* deactivates all workstations. These functions provide a mechanism for controlling the direction of output for applications such as SCIL_Image, which use several workstations simultaneously. For instance, to insure that output appears in exactly one workstation, *GrDeActAllWs()* can be called just prior to *GrActWs()*.

In addition to whether a workstation is active, the visibility of a given graphical object on a workstation depends both on the locations specified for the graphical primitives which make up the object, and the various transformations that these locations traverse before being displayed (see Section A.4.4).

A.4.2 Graphical Output

A group (or graphical object) in GRIP is a collection of graphical output primitives which can be identified as a set for manipulation, modification and deletion. A unique group identifier can be obtained with the function *GrGetGid()* for associating a number of output primitives and subsequently manipulating them as a set. A group can be created implicitly by referring to it in a primitive drawing function, or with a call to *GrSetGroupAttr()* which sets its attributes. Graphical primitives may be added to or deleted from a group whenever some GRIP workstation is active. A group is deleted with *GrDelGroup()*.

There are three graphical output primitives supported in the current implementation of GRIP.

GrPolyline() defines $n-1$ connected lines between n points in the order specified.

GrPolymark() defines n markers at n points specified.

GrText() defines a character string at the specified location with the specified height.

Graphical primitives are created in a group. Each of the above functions for creating primitives requires that a primitive and a group identifier be specified. These identifiers can be used to modify the set of primitive attributes with *GrModPrimAttr()*, or to delete the primitive from its group with *GrDelPrim()*. Alternatively, NONE can be specified as the primitive identifier, in which case the primitive can no longer be identified for individual modification or deletion. No two primitives in the same group may have the same primitive identifier. Visual characteristics of a primitive, such as line width and marker style, depend on the primitive attribute set with which it is created (see Section A.4.3). If a group is deleted, all primitives in the group will be deleted.

Each group of primitives (or graphical object) in GRIP has the following set of attributes associated with it which control its visibility and its visual characteristics.

Transformation: This is a matrix of affine transformation parameters that are used to translate, rotate and scale all primitives in a group prior to display. By default, the transformation is described by the identity matrix, and all primitives are drawn exactly as specified. A number of routines are available for manipulation of the group transformation. It is easy to define any scaling, rotation or translation for all primitives in a group (see Section A.4.4).

Priority: The value of the group priority as compared with that of other groups which share the immediate display space (e.g. overlap) determines which group is "on top", or visible to the user. By default, the priority of a newly created group is higher than that of all existing groups to assure it is visible when created. *GrSetGroupAttr()* can be used to set the priority of a group explicitly. Alternatively, the function *GrSwitchPri()* can be used to exchange the priorities of overlapping groups.

Primitive attribute set: This is the set of attributes which control the appearance of output primitives which use default attributes. By default, these attributes are inherited from the output workstation. The default primitive attribute set is NONE. This means the primitive attribute set associated with each output workstation on which the group is drawn will be used to determine the visual characteristics for those primitives in which NONE has been specified as the primitive attribute set (see Section A.4.3).

A group is visible on those workstations which are active at the time of its creation. This is at the time the group identifier is first used in a drawing or manipulation function and not when the identifier is reserved for use. A group can be copied to other workstations after its creation with *GrCopyGroupWs()*. This is the only function in GRIP which can be used to produce output on an inactive workstation.

A.4.3 Primitive Attributes

A primitive attribute set contains information which controls the appearance of primitives which use it. Some examples of primitive attributes are color, line thickness, line style (e.g. dotted, dashed), marker style (e.g. circle, star), and text font. An attribute set can be assigned to a primitive explicitly, or a primitive can inherit the attribute set from its group. If no primitive attribute set is associated with the group, the primitive will inherit its attributes from the set on each workstation on which its group resides. This is explained in more detail below.

An attribute set is created with *GrGetAid()*, which returns a unique identifier for the set. It can be deleted with *GrDelAid()*. The following functions are provided to modify the parameters in a primitive attribute set.

GrSetColor() sets the color in which to draw the primitive.

GrSetLineStyle() determines whether a line will be solid, dashed, or dotted.

GrSetLineAttr() can be used to set the line join and cap styles as well as the style and width.

GrSetMarkAttr() is used to set the style and size of a marker.

GrSetTextAttr() is used to set the font and direction of the text string.

Each primitive drawing function in GRIP takes an attribute set identifier as one of its arguments. If NONE is specified as the attribute set, then the primitive attribute set associated with the group in which the primitive has been defined will be used. If this in turn is NONE (the default), the attribute set for each workstation on which the primitive appears will be used. A primitive will have the visual characteristics defined in the attribute set at the most specific level. Accordingly, the visual characteristics of a primitive are determined by the primitive attribute set hierarchy in Figure A.2.

Each workstation is created with a generic set of attributes with the same default settings as a set created with *GrGetAid()*. This set exists for the life of the workstation,

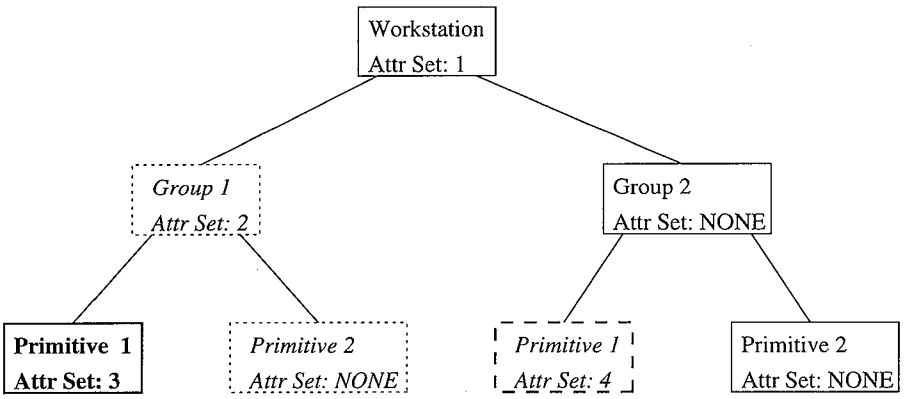


Figure A.2: The primitive attribute inheritance scheme. Primitive 1 in Group 1 and Primitive 1 in Group 2 use private attribute sets 3 (bold font and fat lines) and 4 (italic font and fat striped lines) respectively. Primitive 2 in Group 1 inherits attribute set 2 (italic font and dotted lines) from Group 1. Finally because no attribute set is specified for Primitive 2 in Group 2, and none is set in Group 2, the attribute set 1 (roman font and solid line) are used to draw that primitive.

and cannot be modified in any way. When a workstation is opened, this is the default set for primitives drawn on the workstation. Another set can be specified to be the default by calling *GrSetWsDefAttr()*. This set can be modified with the functions listed above to modify the default visual characteristics of a workstation.

A.4.4 Transformations

Before a graphical primitive is displayed, the set of display coordinates which correspond with the user defined coordinates in the primitive definition must be computed. The coordinate sets are related by a set of transformations. First, the affine transformation associated with the group of primitives must be performed resulting in a possible rotation, translation and scaling of the primitive with respect to the coordinates in the primitive definition. Second, the user or world coordinate system in which the user defines the primitives must be transformed to a normalized coordinate system which is independent of both the application and the selected output devices. Finally, the primitive coordinates in the normalized coordinate system must be transformed to the output devices being used. Thus, there are three independent transformations each primitive undergoes between its specification and its display. The transformations, along with the functions related to manipulating them, are discussed below. The model used in GRIP is similar to that used in standard graphics systems as described in [FvDFH90].

The Normalization Transformation

A normalization transformation maps a rectangular *window* of the world to a rectangular region in a normalized coordinate system. By default, the square defined by the coordinate pair $\{(0,0),(1,1)\}$ is mapped to itself. The rectangle of the world coordinate system is completely determined by the user. With the function *GrSetWindow()* the user can define a window in application (world) coordinates. For a plotting application, the coordinates of the window might depend on the domain and the range of the function being plotted. For an image analysis application, it may depend on the size and the section of the image being investigated. The world coordinate window is mapped to a rectangular subsection or *viewport* of the normalized coordinate system $\{(0,0),(1,1)\}$ with *GrSetViewport()*. A number of normalization transformations or window/viewport pairs can exist simultaneously. *GrSelTrans()* selects the normalization transformation to be used for drawing.

SCIL_Image and the Normalization Transformation When SCIL_Image creates an image window, a default transformation is defined in which the dimensions of the window in the world coordinate system corresponds to the image dimensions, and in which the viewport corresponds to some part of the NDC (normalized device coordinate) square $\{(0,0),(1,1)\}$. By selecting this transformation whenever graphical output is to appear in the workstation in which the image is displayed, primitives defined in terms of image coordinates are superimposed on the displayed image as expected (assuming simple group and workstation transformations).

If the image size is changed, the transformation associated with the image is modified so the world coordinate window coincides with the new image size. This is managed by creating a unique normalization transformation for each image being displayed. For easy manipulation, the workstation in which the image is displayed and the normalization transformation associated with the image are assigned the same identifier.

A convenience function, *DrawIn()*, is provided with SCIL_Image to simplify the management of transformations and the workstation activation scheme for the majority of applications. Although a very simple function, it prevents users wanting to perform simple drawing in images from having to be aware of the normalization transformation the workstation activation scheme. The source of *DrawIn()* is listed in Figure A.3.

For drawing in non-image windows or for using GRIP in other systems, similar functions can be written in terms of the application context.

The Group Transformation

The group transformation allows the user of the library to scale, translate or rotate a group of primitives in any way desired. Primitives are stored in groups in the coordinates in which they are specified. Prior to being displayed, they are transformed according to the affine transformation matrix in the group attribute set, and then according to the normalization transformation associated with the group. This is the last transformation selected with *GrSelTrans()* prior to creation of the group.

```
#include "image.h"
#include "grip.h"

DrawIn(ip)
IMAGE *ip;
{
        int wsid;
        if((wsid = GetWsid(ip)) >= 0) {
                GrDeActAllWs();
                GrActWs(wsid);
                GrSelTrans(wsid);
        }
        return(wsid);
}
```

Figure A.3: The source of *DrawIn()* which allows easy handling of transformations when using GRIP with SCIL_Image.

There are two ways to modify the affine transformation associated with a group. Any of the functions *GrRotateGroup()*, *GrTranslateGroup()*, *GrScaleGroup()*, and *GrReflectGroup()*, which apply a simple transformation to a group can be used. Each of these functions applies the transformation indicated by its name, to the transformation already associated with the group. For example, if *GrRotateGroup()* is twice requested to rotate a group 45°, the group transformation will then include a rotation of 90°. *GrReflectGroup()* reflects all primitives in a group about a specified line.

The group transformation can also be modified with the function *GrSetGroupAttr()*. This function is used to apply a predefined combined transformation. This can be useful if an object is to be translated and rotated without a display update between the two transformations. Each of the transformation functions, *GrRotate()*, *GrTranslate()*, *GrScale()*, and *GrReflect()* is used to define a simple transformation. The results can be combined with the function *GrCompose()*.

In some applications, it is necessary to maintain a set of group transformations. This is particularly useful for animation applications, in which the same set of transformations may be applied and reversed numerous times. For these purposes, the group transformation is implemented as a stack which can be manipulated with the functions *GrPushTrans()* and *GrPopTrans()*.

The Workstation Transformation

Finally, the primitive location must be mapped to the display. This mapping is determined by the size of the display space and by the workstation transformation. By default, this transforms the NDC square $\{(0,0),(1,1)\}$ to the entire device coordinate (DC) space. The area or window in NDC to be mapped can be modified with *GrSetWsWindow()*. The viewport of the display to which it will be mapped can be modified with *GrSetWsViewport()*. Although this function is used to specify a rect-

angle in device coordinates, the coordinates are specified in NDC, and transformed automatically, because the device coordinates may not be known to the programmer.

Transformations in Practice

For simple image overlays, and interactive image manipulations such as those described in this thesis, the default transformations and those maintained for SCIL_Image are sufficient. Using the function *DrawIn()*, one need only select the output image display window. For other sorts of interaction, such as zooming tools, the added control of the transformation parameters is essential.

A.4.5 Input

As discussed in Section A.3.1, the GRIP input model is *external*, which means input events can be accessed and handled according to the needs of an application programmer. A call-back mechanism is provided for accessing events, and a number of functions are provided for handling them. Mechanisms to access internal data, such as the world coordinate position of a given device coordinate pair, or what object is closest to a given location are supplied. A number of tools are also provided for manipulating feedback according to a programmer's wishes. The programmer is free to handle input independently of or in combination with the GRIP functions. Because GRIP does not control input, but only provides functions for handling it, it can be used in combination with interactive systems which do control input, such as the X Toolkit and the SCIL_Image package for image analysis.

In this section, we discuss the tools available in GRIP for input handling. We begin with a discussion of how events are accessed within SCIL_Image, to indicate how direct manipulation tools for image analysis might be developed. We then describe input control functions, which determine which information internal to GRIP is extracted with information retrieving functions. Finally, tools supplied for providing feedback to the user are described.

Accessing Events in SCIL_Image

GRIP receives all input events associated with image windows from SCIL_Image after they have been handled. The call-back mechanism which allows applications to handle events of interest, works as follows. A number of function pointers are available which are initially set to NULL. If the programmer wants to handle a particular event, the function pointer can be set to a function which should be called whenever the event occurs.

The available call-back functions are *GrB-***n**-*Press()*, *GrB-***n**-*Release()*, *GrKeyPress()*, and *GrMotion()*. The press and release functions are supplied for up to five mouse buttons. The functions are by default dummies, implemented as NULL function pointers in the *C* programming language [KR78]. In SCIL_Image, the appropriate GRIP function is called after the corresponding event has been handled. Using the function pointer mechanism, the programmer can reassign the function for a partic-

ular event to one which handles the event (e.g. mouse button press) in a desired fashion. The input handling for a particular action is changed by reassigning the function pointer, or it can be shut off by resetting the pointer to NULL. The source code example in Figure A.4 indicates how this works in practice. The same event handling mechanism can be used in applications outside of SCIL_Image, with the function *GrHandleEvent()*.

Input Control Functions

The input control functions in GRIP allow the application programmer to determine which data is returned by the information retrieval functions *GrLocate()* and *GrPick()*. Both functions request information based on a device location. Which information is returned depends on the viewport input priorities of the normalization transformations. That with the highest viewport input priority will be used in conjunction with the workstation transformation to translate device coordinates to world coordinates. The groups created while this transformation was active, will be the first inspected by the function *GrPick()*.

Suppose, for example that in SCIL_Image, one wants the location in image coordinates of a particular device location in the window associated with image A. The first step is to set the normalization transformation associated with image A to have the highest viewport input priority. In Section A.4.4, we explained that a normalization transformation with the same identifier as the workstation, is associated with each image window. We can access the identifier for image A with the function *GetWsid()*. To assure this will be the transformation used to translate the device coordinates to world coordinates (in this case image coordinates), the value returned by *GetWsid()* can be passed to the input control function *GrTopVpInputPri()* The function *GrLocate()* is then used to perform the actual translation from device to world coordinates. If one wants to pick an object in image A, the same steps would be taken prior to a call to *GrPick()*. Thus for highly interactive applications, a function *GetFrom()* analogous to *DrawIn()* (see Figure A.3) would be useful in SCIL_Image.

Finally, the input control function *GrSetVpInputPri()* can be used to set the viewport input priority of a normalization transformation to be higher or lower than that of another. This allows one to control the relative viewport input priorities of transformations with overlapping viewports.

Information Functions

The function *GrLocate()* produces the world coordinate position of a device location. The translation to world coordinates is based on the normalization transformation with the highest viewport input priority. *GrLocate()* is essential for interactive drawing. For example, when the user points to a particular device location, the location of the mouse pointer can be translated to world coordinates, and a primitive can then be defined in terms of its location.

The function *GrPick()* is used to determine which, if any, primitive is close to a particular device location. As described in Section A.3.2, those primitives drawn

```
#include <stdio.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include "grip.h"       /* where function pointers are known */

int hello();
int bye();

my_func()       /* set up responses to user button presses */
{
        GrB1Press = hello;
        GrB2Press = bye;
}

hello(event)        /* handle event */
XEvent *event;
{
        printf("Hello\n");
}

bye(event)      /* handle event - shut off event handling */
XEvent *event;
{
        printf("My lips are sealed!\n");
        GrB1Press = NULL;
        GrB2Press = NULL;
}
```

Figure A.4: An example of the event access mechanism in GRIP. After *hello()* is called, pressing the first mouse button will generate the text "hello" on the standard output. If the second mouse button is pressed, a different message is printed, and further button presses will be ignored.

in the immediate neighborhood about the device location are considered potential hits. These primitives are sorted, first according to the viewport input priority of the normalization transformation active when the primitive's group was created, and then according to the relative priority of the primitive's group. The sorted primitive list is inspected, and the first to satisfy the hit criteria will be selected. The hit criteria depend on the primitive type.

To inspect graphical primitives, *GrGetPlineData()*, *GrGetPmarkData()*, and *GrGetTextData()*, are provided. Each of these functions returns a pointer to a data structure containing the data used to draw the primitive, and the attribute identifier which determines its visual characteristics. These functions must be called for the correct primitive type, which an be obtained for a particular primitive identifier with *GrGetPrimType()*.

Feedback Functions

GrChangeWsCur() can be used to modify the form of the mouse cursor. This helps indicate that a particular action is being performed. *GrEchoLoc()* echos the user pointer position in terms of a base location. Two types of echo are provided, RUBBER_LINE, and RUBBER_RECT, which are useful for interactive drawing. This feedback mechanism can be used in combination with application specific feedback functions.

No generic functions are available for echoing a primitive or group obtained with *GrPick()*. Rather, a different primitive attribute set can be assigned to the primitive, or group. In this way the feedback mechanism is controlled by the application programmer.

A.5 GRIP versus GKS

Given that GRIP is a two dimensional graphics package, it is useful to know how it differs from a standard graphics package such as GKS. Programmers familiar with GKS, PHIGS and other packages should find it easy to use GRIP, as it shares a number of features found in these packages. The key similarities and differences in GKS and GRIP are the following.

Similarities

- Both systems have a multiple workstation model in which output is directed to the workstations which are active when it is generated.
- Primitive groups in GRIP are similar to segments in GKS in the sense that a group contains a number of primitives which can together be manipulated and deleted.
- The transformation model is identical in both systems.

Differences

- GRIP can share its display space with another package such as SCIL_Image. The lack of this capability in other systems was the motivating factor in the decision to develop GRIP.
- Primitive groups can be created in GRIP at any time simply by referring to them in a function which creates a primitive. The identifier must, however, be reserved prior to use. In GKS, a segment must be created explicitly.
- Primitives can be added to or deleted from a group at any time. In other words, primitive groups can be edited. In GKS, a segment can only have primitives added to it while it is open. It must be closed explicitly. More than one segment may not be open simultaneously.
- The primitive attribute inheritance model described in Section A.4.3 for the manipulation of primitive attributes differs strongly from the GKS model.
- In GRIP, modifications to primitive attributes and changes induced by transformations result in an immediate update of the display contents. GKS supports a programmable model for manipulating screen updates.
- GRIP has an external input model, which means that the programmer can handle input according to the context within which it is used, be it SCIL_Image, an X Windows widget set, or another system. The input model in GKS is internal, meaning that input is handled according to a fixed input model, and cannot be extended in any fashion.

A.6 Implementation

GRIP has been implemented under the X Window System [SGN88] in the C programming language [KR78], and runs on a variety of UNIX workstations including the Sun-3, the Sun Sparc workstation series, the SGI workstation series, the IBM RS6000, the Stardent 3000, the HP9000, and a variety of Apollo workstations. It has been incorporated in the SCIL_Image package [oA91], and is employed both for experimental image analysis and for application development.

The communication between SCIL_Image and GRIP works as follows. When SCIL_Image opens a window for image display, it calls GRIP, and passes pointers to image and display structures using a connection data structure provided for shared applications. In particular, it passes the application name, the X window identifier and X window display pointer, and pointers to the Scil_Image image structure and to the X Window System image data structure (XImage). The application name is used for resource handling, the X window and display pointer for directing graphical output, and the XImage pointer for performing selective update operations.

SCIL_Image performs X Window event handling. When an expose event is received, or when the XImage is modified due to the execution of an image processing operation,

SCIL_Image calls the appropriate redrawing function in the GRIP library. Window resize events also cause SCIL_Image to call a special function to update the workstation transformation. All other events are passed to GRIP or to the application.

Finally, GRIP has been designed to be portable. The system components which are UNIX and X-Window dependent make up only 15% of the source code and are separated from the remainder of the sources. Moreover, the shared workstation and cooperative input models in GRIP, while designed to work well with SCIL_Image have no knowledge of the context in which they are used, and can be easily incorporated in other systems which require cooperative use of the input and output devices.

A.7 Conclusions

GRIP provides the basic graphics functionality necessary to the development of direct manipulation user interface tools for image analysis. It was used to construct the polygonal drawing and correction tools described in Chapters 2 and 4 of this thesis, and for the tracing tool described in Chapter 5. It has also been used in various analysis applications to display object boundaries, medial axes, and other geometric object entities (see for example [PRS+94]).

Because we designed GRIP with direct manipulation tools in mind, we support an intentionally simple display-update model which is effective for this application domain. For applications such as animation or robot simulation, however, in which numerous modifications follow one another immediately, it is far more efficient to update the display after a series of modifications rather than after each individual modification. Extending GRIP to allow periodic updates under programmer control would make it useful for other applications in which communication with an image analysis package is required.

The default update scheme combined with the attribute inheritance scheme makes GRIP far easier to program than standard packages such as GKS and PHIGS. Further, the external input model and the shared workstation model make GRIP fit in well with currently available user interface toolkits. This makes it possible to develop highly interactive applications for image analysis and other domains which require access to the display and input devices.

Bibliography

[ACH+91] E.M. Arkin, L.P. Chew, D.P. Huttenlocher, K. Kedem, and J.S.B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):209–216, March 1991.

[Ado90] Adobe Systems Incorporated, 1585 Charleston Road, Mountain View, California. *Adobe Photoshop*, 1990.

[Ant93] P.P.J. Antonissen. Model-gestuurde object-detectie(in Dutch). Master's thesis, Faculty of Math and Computer Science, University of Amsterdam, Amsterdam, The Netherlands, February 1993.

[Att54] F. Attneave. Some informational aspects of visual perception. *Psycological Review*, 61:183–193, 1954.

[AWJ90] A.A. Amini, T.E. Weymouth, and R.C. Jain. Using Dynamic Programming for Solving Variational Problems in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):855–867, September 1990.

[Bad92a] A.J. Baddeley. An error metric for binary images. In W. Förstner and S. Ruwiedel, editors, *Robust Computer Vision*, pages 59–78, Karlsruhe, 1992. Wichmann.

[Bad92b] A.J. Baddeley. Errors in binary images and an L^p version of the Hausdorff metric. *Nieuw Archief voor Wiskunde*, 10:157–183, 1992.

[BB82] D.H. Ballard and C.M. Brown. *Computer Vision.* Prentice Hall, Englewood Cliffs, NJ, 1982.

[BD90] J. Buurman and R.P.W. Duin. Object recognition using inexact matching of 3d graphs. In *Progress in Image Analysis and Processing*, pages 415–419, Singapore, 1990. World Scientific.

[BD92] B. Bascle and R. Deriche. Features extraction using parametric snakes. In *Proc. of the 11th IAPR International Conference on Pattern Recognition*, volume 3, Image Speech and Signal Analysis, pages 659–662, Los Alamitos, CA, 1992. IEEE Computer Society Press.

[Bie90] E.A. Bier. Snap-Dragging in Three Dimensions. In *Symposium on Interactive 3D Graphics, ACM Computer Graphics*, volume 24, 2, pages 193–204, Snowbird, Utah, March 1990.

[Bor81] A.H. Borning. The Programming Aspects of ThingLab, A Constraint-Oriented Simulation Library. *ACM Transactions on Programming Languages*, 3(4):353–387, October 1981.

[Bor86] A.H. Borning. Defining Constraints Graphically. In *ACM SIGCHI Proceedings*, pages 137–143, April 1986.

[Bre65] J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[Bre77] J.E. Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2):100–106, February 1977.

[Bro66] P. Brodatz. *Texture: A Photographic Album for Artists and Designers.* Dover, New York, 1966.

[BS81] B. Bramer and D.C. Sutcliffe. Application of GINO-F to Use Display File Techniques on Raster Scan Displays. In *EuroGraphics-1981*, pages 271–280. North Holland Publishing Company, 1981.

[BS86] E.A. Bier and M.C. Stone. Snap-Dragging. In *ACM Computer Graphics*, volume 20, 4, pages 233–240, Dallas, August 1986.

[Can86] J.F. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intellgence*, 8:679–698, 1986.

[Cla92] Claris Corporation, Mountain View, CA. *MacDraw Pro Manual*, 1992.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[Com85] Apple Computer. *Inside Macintosh.* Addison-Wesley, Reading, Mass, 1985.

[Cor70] T.N. Cornsweet. *Visual Perception.* Academic Press, New York, 1970.

[Cou92] J. Coutaz. Critical issues: User modelling. In J. Larson and C. Unger, editors, *Engineering for Human-Computer Interaction*, pages 419–423. Elsevier, 1992.

[DCH88] R.A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *ACM Computer Graphics*, volume 22, pages 65–74. (Proceedings SIGGRAPH), 1988.

[DT88] L. Dorst and K. Trovato. Optimal path planning by cost wave propagation in metric configuration space. In *SPIE Mobile Robots III*, volume 1007, pages 186–197, 1988.

[FC77] W. Frei and C.C. Chen. Fast boundary detection: A generalisation and a new algorithm. *IEEE Transactions on Computers*, 26:988–998, 1977.

[FD77] H. Freeman and L.S. Davis. A corner finding algorithm for chain code curves. *IEEE Transactions on Computers*, 26:297, 1977.

[FE73] M.A. Fischler and R.A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, 22(1):67–92, 1973.

[FLP89] H. Fuchs, M. Levoy, and S.M. Pizer. Interactive visualization of 3d medical data. *IEEE Computer*, 22(8):46–51, August 1989.

[FvDFH90] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *The Fundamentals of Computer Graphics.* Addison-Wesley, Reading, Mass, 1990.

[FW94] M.A. Fischler and H.C. Wolf. Locating perceptually salient points on planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):113–129, February 1994.

[GAW93] F.C.A. Groen, P.P.J. Antonissen, and G.A. Weller. Model based robot vision. In *Instrumentation and Measurement Technology*, pages 584–588, Irvine, California, 1993. IEEE.

[Ger88] J.J. Gerbrands. *Segmentation of Noisy Images.* PhD thesis, Delft University of Technology, Faculty of Electrical Engineering, Delft, The Netherlands, November 1988.

[GG92] D.M. Gavrila and F.C.A. Groen. 3d object recognition from 2d images using geometric hashing. *Pattern Recognition Letters*, 13:263–278, 1992.

[Gra88] ANSI (Computer Graphics). Programmer's hierarchical interactive graphics system (phigs) functional description. Technical Report ANSI X3.144-1988, American National Standards Institute, 1988.

[GS66] D.M. Green and J.A. Swets. *Signal Detection Theory and Phycophysics.* Peninsula Publishing, 1988 edition, 1966.

[GS92] T. Gevers and A.W.M. Smeulders. Enigma: An image retrieval system. In *Proc. of the 11th IAPR International Conference on Pattern Recognition*, volume 2, Pattern Recognition Methodology and Systems, pages 697–700, Los Alamitos, CA, 1992. IEEE Computer Society Press.

[GW92] R.C. Gonzalez and R.E. Woods. *Digital Image Processing.* Addison Wesley, Reading, Mass, 1992.

[Har78] R.M. Haralick. Statistical and structural approaches to texture. In *Proceedings 4th IJCPR*, pages 45–60, 1978.

[HDGS83] F.R.A. Hopgood, D.A. Duce, J.R. Gallop, and D.C. Sutcliffe. *Introduction to the Graphical Kernel System.* Academic Press, London, 1983.

[HHN86] E.L. Hutchins, J.D. Hollan, and D.A. Norman. Direct Manipulation Interfaces. In D.A. Norman and S. W. Draper, editors, *User Centered System Design*, pages 87–124. Laurence Erlbaum Associates, Hillsdale, NJ, 1986.

[Hil69] C. J. Hilditch. Linear skeletons from square cupboards. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence 4*, volume 4, pages 404–420, Edinburgh, 1969. University Press Edinburgh.

[HM91] J. Hardenbergh and J. Michener. Integrating PHIGS and User Interface Systems. *Computer Graphics Forum*, 10(1), 1991.

[HS85] R.M. Haralick and L.G. Shapiro. Survey image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29:100–132, 1985.

[HSvdV+77] A.B. Houtsmuller, A.W.M. Smeulders, H.T.M. van der Voort, J.L. Oud, and N. Nanninga. The homing cursor: A tool for three-dimensional chromosome analysis. *Cytometry*, 14:501–509, 1977.

[Hud90] S.E. Hudson. Adaptive Semantic Snapping - A Technique for Semantic Feedback at the Lexical Level. In *ACM SIGCHI Proceedings*, pages 65–70, April 1990.

[KDMSH92] T. Kühme, H. Dieterich, U. Malinowski, and M. Schneider-Hufschmidt. Approaches to adaptivity in user interface technology: Survey and taxonomy. In J. Larson and C. Unger, editors, *Engineering for Human-Computer Interaction*, pages 419–423. Elsevier, 1992.

[Kir71] R. Kirsh. Computer determination of the constituent structure of biological images. *Comput. Biomed. Res.*, 4:315–328, 1971.

[KR78] B.W. Kernighan and D.M. Ritchie. *The C Programming Language.* Prentice Hall, Englewood Cliffs, NJ, 1978.

[KR82] L. Kitchen and A. Rosenfeld. Gray-level corner detection. *Pattern Recognition Letters*, 1(2):95–102, 1982.

[KWT88] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, pages 321–331, 1988.

[LC87] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *IEEE Computer Graphics*, 21(3):163–169, July 1987.

[Lev88] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):27–33, May 1988.

[LP90] L.M. Lifshitz and S.M. Pizer. A multiresolution hierarchical approach to image segmentation based on intensity extrema. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):529–540, June 1990.

[LSWM86] X. Li, C. Shanmugamani, T. Wu, and R. Madhavan. Correlation measures for corner detection. In *IEEE CVPR*, pages 643–646, 1986.

[LT90] S. Liu and W. Tsai. Moment preserving corner detection. *Pattern Recognition*, 23(5):441–459, 1990.

[LW88] Y. Lamdan and H.J. Wolfson. Geometric Hashing: A general and efficient model-based recognition scheme. In *Second International Conference on Computer Vision*, pages 238–249, Tampa, Florida, 1988. IEEE.

[MH80] D. Marr and E. Hildreth. Theory of edge detection. In *Proceedings: Royal Society London*, pages 187–217, 1980.

[MNR90] R. Mehrotra, S. Nichani, and N. Ranganathan. Corner detection. *Pattern Recognition*, 23(11):1223–1233, 1990.

[MT81] J.E. Marsden and A.J. Tromba. *Vector Calculus.* W.H. Freeman and Company, New York, second edition, 1981.

[Nil80] N.J. Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufman, 1980.

[oA91] University of Amsterdam. SCIL_Image Manual, 1991.

[oAP88] TNO Institute of Applied Physics. TCL-IMAGE User's Manual, November, 1988.

[OG93] C.M. Orange and F.C.A. Groen. Model based corner detection. In *Computer Vision and Pattern Recognition*, pages 690–691, New York, New York, 1993. IEEE.

[OG94] C.M. Orange and F.C.A. Groen. Magnetic contour tracing. In *Workshop on Visualization and Machine Vision*, pages 33–44, Seattle, WA, 1994. IEEE.

[OGA94] C.M. Orange, F.C.A. Groen, and P.P.J. Antonissen. An A* algorithm for inexact polygon matching. In E. Backer, editor, *Computing Science in the Netherlands, CSN'94*, November 1994.

[oH93] National Institute of Health. Image 1.52, 1993.

[Pik83] R. Pike. Graphics in Overlapping Bitmap Layers. *ACM Transactions on Graphics*, 2:135–160, April 1983.

[PM82] T. Peli and D. Malah. A study of edge detection algorithms. *Computer Graphics and Image Processing*, 20:1–21, 1982.

[Pra77] W.K. Pratt. *Digital Image Processing.* John Wiley and Sons, 1977.

[Pra91] W.K. Pratt. *Digital Image Processing, 2nd Edition.* Wiley, 1991.

[Pre70] J.M.S. Prewitt. Object enhancement and extraction. In B. S. Lipkin and A. Rosenfeld, editors, *Picture Processing and Psychopictorics*, pages 75–149. Academic Press, New York, 1970.

[PRS+94] J. Piper, D. Rutovitz, D. Sudar, A. Kallioniemi, O. Kallioniemi, F.M. Waldman, J.W. Gray, and D. Pinkel. Computer image analysis of comparative genomic hybridization. *submitted*, 1994.

[RA77] E.M. Riseman and M.A. Arbib. Survey computational techniques in the visual segmentation of static scenes. *Computer Graphics and Image Processing*, 6:221–276, 1977.

[RJ73] A. Rosenfeld and E. Johnston. Angle detection on digital curves. *IEEE Transactions on Computers*, 22:875–878, 1973.

[RK76] A. Rosenfeld and A.C. Kak. *Digital Picture Processing.* Academic Press, New York, 1976.

[Rob65] L.G. Roberts. Machine perception of three dimensional solids. In J.T Tippet et al., editor, *Optical and Electro-Optical Information Processing*, pages 159–197. MIT Press, Campbridge, Massachusetts, 1965.

[Rob76] G.S. Robinson. Detection and coding of edges using directional masks. Technical Report 660, University of Southern California, Image Processing Institute, 1976.

[RSB89] K. Rangarajan, M. Shah, and D. Van Brackle. Optimal corner detector. *Computer Vision, Graphics, and Image Processing*, 48:230–245, 1989.

[SD89] L.H. Staib and J.S. Duncan. Parametrically deformable contour models. In *Conference on Computer Vision and Pattern Recognition*, pages 98–103. IEEE, June 1989.

[SDS88] M. Slater, A.J. Davidson, and M.B. Smith. Liberation from Rectangles: A Tiling Method for Dynamic Modification of Objects on Raster Displays. In *EuroGraphics-1988*, pages 381–392, 1988.

[Ser82] J. Serra. *Image Analysis and Mathematical Morphology.* Academic Press, London, 1982.

[SGN88] R.W. Scheifler, J. Gettys, and R. Newman. *X Window System.* Digital Press, 1988.

[Shn83] B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Computer*, 16(8):57–69, August 1983.

[Shn87] B. Shneiderman. *Designing the User Interface.* Addison-Wesley, Reading, Mass, 1987.

[Sme91] A.W.M. Smeulders. An Introduction to Image Processing and Computer Vision. University of Amsterdam, Amsterdam, The Netherlands, 1991.

[Smi39] N.V. Smirnov. On the estimation of the discrepancy between empirical curves of distribution for two independent samples. *Bulletin Math. University of Moscow*, 2:3–14, 1939.

[SS90] A. Singh and M. Shneier. Grey Level Corner Detection: A Generalization and a Robust Real Time Implementation. *Computer Vision, Graphics, and Image Processing*, 51:54–69, 1990.

[Str50] D.J. Struik. *Lectures on Classical Differential Geometry.* Addison-Wesley, Reading, Mass, 1950.

[Sut63] I.E. Sutherland. Sketchpad: A man-machine graphical communication system. In *SJCC*, Baltimore, MD, 1963. Spartan Books.

[Sut85] S. Sutanthavibul. *Xfig – Facility for Interactive Generation of Figures under X11.* Massachusetts Institute of Technology, Cambridge, MA., 1985.

[tHRFKV92] B.M. ter Haar Romeny, L.M.J. Florack, J.J. Koenderink, and M.A. Viergever. Invariant third order properties of isophotes: T-junction detection. In P. Johansen and S. Olsen, editors, *Theory and Applications of Image Analysis*, volume 2 of *Series in Machine Perception and Artificial Intelligence*, pages 30–37. World Scientific, Singapore, 1992.

[tKvBS+90] T.K. ten Kate, R. van Balen, A.W.M. Smeulders, F.C.A. Groen, and G.A. den Boer. SCILAIM: A Multi-Level Interactive Image Processing Environment. *Pattern Recognition Letters*, 11:429–441, 1990.

[vBtKK+93] R. van Balen, T. ten Kate, D. Koelma, . Mosterd B, and A.W.M. Smeulders. Scilimage: A multi-layered environment for use and development of image processing software. In H.E. Christensen and J.L. Crowley, editors, *Experimental Environments for Computer Vision and Image Processing.* World Scientific Press, Singapore, 1993.

[vM64] R. von Mises. *Mathematical Theory of Probability and Statistics.* Academic Press, New York, 1964.

[vVYB89] L.J. van Vliet, I.T. Young, and A.L.D. Bekkers. A nonlinear Laplace operator as edge detector in noisy images. *Computer Vision, Graphics and Image Processing*, 45:167–195, 1989.

[vVYtK+89] L.J. van Vliet, I.T. Young, T.K. ten Kate, B.H. Mayall, F.C.A. Groen, and R. Roos. Athena, A Macintosh Based Interactive Karyotyping System. In C. Lundsteen and J. Piper, editors, *Automation of Cytogenetics*, pages 47–66. Springer Verlag, Berlin, 1989.

[Wat87] R.J. Watt. An outline of the primal sketch in human vision. *Pattern Recognition Letters*, 5(2):139–150, February 1987.

[WD84] K. Wall and P. Danielsson. A Fast Sequential Method for Polygonal Approximation of Digitized Curves. *Computer Vision, Graphics, and Image Processing*, 28:220–227, 1984.

[WK79] J. Warner and N. Keifhaber. The DIGRAF Implementation of the Proposed GSPC Standard. In *EuroGraphics-1979*, Balogna, Italy, 1979.

[WS92] D.J. Williams and M. Shah. A Fast Algorithm for Active Contours and Curvature Estimation. *CVGIP: Image Understanding*, 55(1):14–26, January 1992.

[WSSD93] M. Worring, A.W.M. Smeulders, L.H. Staib, and J.S. Duncan. Parameterized feasable boundaries in gradient vector fields. In H.H. Barrett and A.F. Gmitro, editors, *Proceedings of Information Processing in Medical Imaging*, volume 687 of *Lecture notes in computer science*, pages 48 – 61, 1993.

[You89] D.A. Young. *X Window Systems Programming and Applications with Xt.* Prentice Hall, Engelwood Cliffs, NJ, 1989.

[ZH83] A. Zuniga and R. Haralick. Corner detection using the facet model. In *Computer Vision and Pattern Recognition*, pages 30–37. IEEE, 1983.

Summary

The segmentation of an image into meaningful parts is a key step in nearly every image analysis problem. Accurate segmentation is crucial for object identification and feature analysis. In spite of the apparent ease with which approximate segmentation is performed with the human eye, it remains a central problem in computer vision. In general, successful segmentation requires a good model of the image function, particularly in the neighborhood of object boundaries. In this thesis, we introduce an approach to segmentation called *supervised boundary formation* for the segmentation of images for which we have insufficient models. With this approach, an image is partitioned based in part on user sketches of image object boundary paths. Problems which arise in the effort to segment images with this approach are investigated for two interactive path specification models, namely *connect-the-dots* and *freehand drawing*. These two models were selected because they have been effective for interactive path specification in drawing packages. Because a user path cannot be assumed to be an accurate description of an object boundary path, supervised boundary formation is an effective method of segmentation only if it provides a means for correcting a user input path. Such correction, of course, requires user input be correctly interpreted.

Aspects of user interpretation which are critical for image object boundary formation are formally addressed for both the connect-the-dots and freehand drawing input models. Based on numerous experiments, we show supervised boundary formation to be an effective approach to the segmentation of images for which a sufficient segmentation model is unavailable.

User error model To correct a user sketch of an image object boundary, it is essential to understand the type and degree of error a user makes in producing it. In Chapter 2, we describe experiments used to measure user errors in the specification of corners on polygonally shaped objects. A number of users were presented with images containing arbitrary polygonal objects and asked to specify points corresponding to corners on the object boundaries. Lines were drawn automatically between the subsequent points specified by the user ("connect-the-dots"). For each corner on the resulting polygonal path, we measured the error in distance, corner angle, corner orientation, and corner scale. These experiments lead us to conclude that the type and degree of errors made by individual users in the specification of corner points, can be predicted. Among the tested users, however, performance differed significantly. A user dependent error model for corner specification is therefore derived based on the results. This model consists of the maximum expected distance between a user defined

point and the corresponding point on the object boundary, and the bias and deviation of the user corner angle, orientation and scale in comparison with the model corner in the image. In Chapter 4, this model is employed to develop a user adaptable technique for corner correction.

Inexact polygon matching To perform the corner error measurements described in Chapter 2, points in a user defined polygon must be matched with those in a model polygon. An isomorphic relationship between the user defined polygon and the object model polygon cannot be assumed, as users do not always specify all corner points on the test object, and sometimes specify extra ones. To address this problem, we introduce an algorithm for inexact polygon matching of nonisomorphic polygons. A *goodness-of-fit* measure for comparing polygon components (points or line segments) is quantified in a cost function which is applied to every component pair derived from two polygons. Subsequently, we determine an optimal fit of the polygons with an A* algorithm that exploits the cyclic characteristics of polygons and allows points to be skipped in either polygon.

A general framework for the development of a cost function used to compare possible matches is described. Within this framework, we developed a cost function used to match the user and model polygons generated by the experiments in Chapter 2, and a cost function suitable for robotics applications in which the camera position is fixed. The algorithm was used to correctly match each of the user input polygons in the experiments in Chapter 2 with the associated model polygon. In a robotics application, it has been successfully applied to decide whether an object in an imaged scene matches one extracted from a database. A complexity analysis shows it to be two orders of magnitude faster than existing methods for object matching. The key contribution which distinguishes our approach from other polygonal object matching methods, however, is the separation of the node pair match evaluation from the global matching decision. This allows us to guarantee that, with respect to a given cost function, the match of the polygons produced with our algorithm is a global optimum match.

Model-based corner detection In Chapter 4, we turn our attention to the correction of polygonal object boundary paths, specified by a user using a "connect-the-dots" drawing tool. To correct user errors made in the specification of corner points on polygonally shaped objects, we develop a model-based corner detection method. Our scheme enables the evaluation of an image region based on the geometry of a corner in a polygonal shape specified by a user. Based on the user error model derived in Chapter 2, we examine constraints on the geometry of corner on the image object in relation to the geometry of a corner specified by a user. For a given user error model and an image feature measure λ, we examine conditions under which we can extract an image corner model with respect to the measure λ. We show that if the window size required to measure λ and the user error in distance are both sufficiently small in relation to the corner angle and scale, then we can establish the presence of a ramp or roof corner as measured by λ. In our experiments, we show that if λ measures the

grey value intensity, we determine the correct corner model in 95% of the cases. We use the model to localize the corner in the object boundary.

Magnetic contour tracing In Chapter 5, we develop a method for real time correction of a user specified boundary path. The interactive model supported is "free-hand drawing", but rather than producing the path actually traced by the user, we extract a best approximation to an object boundary path in the immediate region. Under the assumption that the user path satisfies specific conditions in relation to the path of the object boundary, we derive a set of candidate digital paths guaranteed to contain a best digital approximation to the image object boundary. If our model of the image function along the image object boundary is correct, the dynamic programming algorithm described in Chapter 5 will produce a best approximation to the path of the object boundary.

The definition of a cost function determines which among a set of candidate paths is selected. For interactive segmentation, the cost function is defined as a weighted combination of a user term based on the direction of user movement and a boundary (gradient) strength term. The influence of the weights is evaluated experimentally, and the set which minimizes the errors made over the length of a boundary is used. The magnetic ink method produces a good approximation of a boundary path for a range of test images containing discs, ellipses and hexagons. Moreover, it works very well for a far wider combination of input and boundary paths than the theoretical arguments would lead us to believe.

Conclusion The results in this thesis show that the uncertainty about the segmentation results involved when a human expert plays a direct role in the segmentation process, can be eliminated based on user modeling and formal analysis. By approaching the human interface issues involved with image object boundary specification in a systematic fashion, we have shown supervised boundary formation to be an effective approach to segmentation.

Samenvatting

Het opdelen van een beeld in zinvolle gebieden is een essentiële stap bij beeldanalyse. Nauwkeurige segmentatie is cruciaal voor het identificeren van objecten in een beeld en voor het meten van gegevens als omtrek en vorm van objecten. Hoewel het voor een mens vaak eenvoudig is om een beeld te segmenteren, blijft het automatiseren van dit proces één van de moeilijkste problemen in de beeldanalyse. Om een beeld automatisch te kunnen segmenteren, is een goed model van de beeldfunctie in de omgeving van objectgrenzen vereist. Voor een willekeurig beeld is echter in het algemeen geen geschikt segmentatiemodel voorhanden. In dit proefschrift introduceren we een nieuwe aanpak, "supervised boundary formation", voor het segmenteren van beelden waarvoor onvoldoende modellen beschikbaar zijn. Bij deze aanpak gebruiken we een door een mens gemaakte schets van de objectgrenzen in een beeld om een segmentatiemodel te bouwen.

De problemen die zich bij onze methode voordoen hebben we voor twee interactieve tekenmodellen onderzocht, namelijk "connect-the-dots" en "freehand drawing". Deze twee modellen zijn gekozen omdat ze in tekenpakketten handig zijn gebleken bij het schetsen van paden. Omdat niet zonder meer mag worden aangenomen dat een gebruikersschets de werkelijke objectgrens precies weergeeft, is "supervised boundary formation" alleen doeltreffend als het automatisch verbeteren van een gebruikersschets deel van de methode uitmaakt. Het aanpassen van zo'n schets vereist uiteraard een correcte interpretatie van de gebruikersinvoer.

Voor beide tekenmodellen onderzoeken we daarom op formele wijze aspecten van de gebruikersinvoer die van belang zijn voor beeldsegmentatie. Aan de hand van talrijke experimenten tonen we aan dat "supervised boundary formation" een effectieve methode is voor het segmenteren van die beelden waarover te weinig informatie beschikbaar is om de segmentatie volledig automatisch te verrichten.

Een gebruikers-foutenmodel Om een door de gebruiker gegeven schets van de grens van een object in een beeld te kunnen corrigeren, moeten we weten wat voor soort fouten een gebruiker maakt bij het tekenen en hoe groot die fouten kunnen zijn. In Hoofdstuk 2 van dit proefschrift beschrijven we experimenten om de fouten te meten die iemand maakt bij het aangeven van hoekpunten van polygonale objecten. Een aantal personen is gevraagd om de hoekpunten van polygonale objecten in testbeelden aan te geven. Opeenvolgende punten in de tekening worden automatisch door lijnstukken verbonden ("connect-the-dots"). Voor iedere hoek in de aldus verkregen schets hebben we vervolgens de geometrische eigenschappen positie, aantal graden,

orientatie en schaal vergeleken met de eigenschappen van de corresponderende hoek van het testobject. Uit deze experimenten bleek dat de fouten die gemaakt worden bij het aangeven van hoekpunten per persoon voorspelbaar zijn. Derhalve hebben we een gebruikersspecifiek foutenmodel kunnen afleiden. In dit model wordt de maximaal verwachte afstand tussen een punt dat een gebruiker aangeeft en het overeenkomstige punt op de feitelijke objectgrens beschreven, en zijn de bias en de standaarddeviatie voor de afwijkingen in het aantal graden van de hoek, de orientatie en de schaal opgenomen. In Hoofdstuk 4 maken we van dit model gebruik om een aan een gebruiker aan te passen methode voor hoekcorrectie te ontwikkelen.

Polygoon matchen Om bovengenoemde hoekafwijkingen te kunnen meten moet eerst iedere hoek die een gebruiker in een polygoon aangeeft met een hoek van het testobject geïdentificeerd worden. Tijdens het tekenen zou een gebruiker per ongeluk één of meerdere hoekpunten van het object kunnen overslaan of extra punten kunnen aangeven. Men mag dus niet aannemen dat de gebruikerspolygoon isomorf is met de polygoon in het testbeeld. Om dit probleem aan te pakken introduceren we in Hoofdstuk 3 een methode voor het *matchen* van twee polygonen. Eerst wordt een kostenfunktie beschreven die een waarde toekent aan de mate van verschil tussen componenten (punten of lijnstukken) van polygonen. Met deze kostenfunktie wordt iedere component in de ene polygoon vergeleken met iedere component in de andere. Een A* algoritme berekent uit de resulterende matrix de reeks van componentparen die een match van de twee polygonen specificeert en optimaal is met betrekking tot de gegeven kostenfunctie. Dit algoritme benut de cyclische kenmerken van een polygoon en is daardoor zeer efficient.

Een algemeen raamwerk voor het ontwikkelen van kostenfuncties voor willekeurige applicaties wordt beschreven. Binnen dit raamwerk hebben we een kostenfunctie ontworpen om de door gebruikers geschetste polygonen en de testpolygonen uit Hoofdstuk 2 met elkaar te vergelijken. We zijn er vervolgens in geslaagd om met ons A* algoritme iedere geschetste polygoon correct te *matchen* met het bijbehorende testobject. Verder hebben we binnen hetzelfde raamwerk een kostenfunctie voor lijnstukken ontworpen voor een robotica toepassing. Het is gelukt om met onze methode objecten die in de ruimte voorkomen te *matchen* met objecten uit een databestand.

Een complexiteitsanalyse toont aan dat onze methode twee orden van grootte sneller is dan bestaande methoden. Belangrijker is echter dat onze methode, in tegenstelling tot andere, het met een kostenfunctie vergelijken van componenten scheidt van het berekenen van een geschikte reeks componentparen. Door deze scheiding kunnen we garanderen dat de gevonden reeks optimaal is met betrekking tot de gebruikte kostenfunctie.

Model gebaseerde hoekdetectie In Hoofdstuk 4 houden we ons bezig met het corrigeren van een door een gebruiker aangegeven hoekpunt in een polygonale schets die met de "connect-the-dots" methode is gemaakt. Hiertoe proberen we eerst een model van het beeld in het gebied van het aangegeven punt te bouwen. Aan de hand van de geschetste hoek in de omgeving van het punt wordt een geometrisch

model van de hoek in het beeldobject gebouwd. Voor een gegeven functie λ die een bepaald kenmerk, bijvoorbeeld grijswaarde of textuur, van een punt in een beeld meet, onderzoeken we onder welke voorwaarden we een model van de hoek kunnen afleiden. We tonen aan dat het mogelijk is om een op λ gebaseerd hoekmodel te bouwen als zowel het gebied dat nodig is om λ te meten als de te verwachten gebruikersfout klein is in verhouding tot het aantal graden en de schaal van de door de gebruiker aangegeven hoek. Het door de gebruiker geschetste polygonale pad deelt het gebied rondom het aangegeven hoekpunt in tweeën. Als de waarden van de functie λ voor de twee gebieden duidelijk verschillend zijn, zijn we in staat het soort hoek (*ramp* of *roof*) te bepalen en vervolgens de positie van het eigenlijke hoekpunt te lokaliseren.

Het magnetisch schetsen van contouren In Hoofdstuk 5 van dit proefschrift wordt een methode ontwikkeld voor dynamische correctie van een gebruikersschets. In dit geval maken we gebruik van het "freehand drawing" tekenmodel, maar in plaats van het door de gebruiker geschetste pad weer te geven, wordt de schets aangepast aan de nabijgelegen beeldgrens. Het lijkt alsof de pen van de gebruiker door een magneet naar de grens van het object wordt getrokken. Indien de verhouding tussen de schets van de gebruiker en het pad van de objectgrens aan bepaalde eisen voldoet, kunnen we een verzameling digitale paden afleiden die gegarandeerd de beste benadering van de objectgrens bevat. Het algoritme dat we in Hoofdstuk 5 beschrijven kan een optimale digitale benadering van een objectgrens vinden als we een correct model van de beeldfunctie voor de omgeving van die grens hebben.

We selecteren de optimale benadering van een grens uit de verzameling van mogelijke digitale paden met behulp van een kostenfunctie. De kostenfunctie voor interactieve beeldsegmentatie wordt bepaald door de tekenrichting en de verwachte eigenschappen van de beeldfunctie in de omgeving van de objectgrens. Hoe zwaar elk van deze factoren mee moet wegen hebben we experimenteel vastgesteld. De magnetische schetsmethode leidt tot een goede digitale benadering van objectgrenzen van cirkels, ellipsen en hexagonen in testbeelden. Bovendien levert de methode ook uitstekende resultaten bij een groot aantal beelden die niet voldoen aan de theoretische eisen.

Conclusie In dit proefschrift tonen we aan dat het nemen van gebruikersinvoer als uitgangspunt voor beeldsegmentatie tot goede resultaten leidt, mits men zich baseert op een formele analyse van gebruikersschetsen. Als gebruikerskarakteristieken systematisch worden gemodelleerd kan "supervised boundary formation" een doeltreffende aanpak voor beeldsegmentatie zijn.

Acknowledgements

First and foremost, I want to thank my research advisor Frans Groen for his consistent and excellent guidance. His interest and enthusiasm were always stimulating. His careful reading and constructive criticism of numerous drafts of this thesis resulted in a much clearer text. Most of all, his patient questions ("snap ik niet") required me to develop and present my work in an increasingly formal manner. It is a great honor to have worked with him.

I am grateful to Ted Young for providing me with the opportunity to perform this work and a stimulating atmosphere in which to do so. Not only was I able to benefit from the excellent atmosphere in the Pattern Recognition Group at the TU Delft, but was given the chance to spend time in the Computer Systems Group at the University of Amsterdam. This was ideal given the nature of my research and the location of my advisors. Ted's useful comments and suggestions on experimental methods allowed me to reach stronger conclusions about the practical applicability of my techniques. Due to his careful review of this thesis, I was able to remove almost all of the modifiers that were dangling.

What a treat to have Leo Dorst appear one April. He always seemed happy to answer simple and complicated questions, and was even kind enough to act as if all were the latter. His willingness to review my work, and his art for identifying interesting issues were most beneficial. His great sense of humor cheered many a day. Finally, Leo's careful reading of this thesis allowed me to remove many many double words.

Arnold Smeulders made room for me in the image processing group at the UvA, then known as the breiclub. The informal group discussions provided an excellent forum for the exchange of ideas. I am especially indebted to Arnold for his encouragement and constructive interest in early phases of this research.

I am honored that in addition to those mentioned above, dr. ir. J.J. Gerbrands, prof. dr. ir. F.W. Jansen, and prof. dr. ir. M.A. Viergever were willing to review my thesis as members of the reading committee.

As a master's student at the UvA, René Vreeswijk worked on a contour tracing technique, which helped me solidify ideas on the material in Chapter 5.

Patrick Antonissen is thanked for sacrificing some weekends to explain a predecessor to the polygon matching method, which he developed as a masters student.

I'm grateful to the following people who, on short notice, willingly participated in the experiments described in Chapter 2: Richard van Balen, Kai Compagner, Casper Dik, Patricia Griffin, Bianca Jongkind, Marijke Kaat, Dennis Koelma, Frank van der Linden, Benno Mosterd, Patrick van der Smagt, Susan Üsküdarlı, and Frans Vos.

Secretaries are often forgotten, but if they forgot us, just what would we do? At the UvA, Laura Lotty, Ina van der Velde, Monique Kleinendorst, and Virginie Meijer saw to it that my articles raced away in the nick of time, and that lots of now forgotten details were seen to. Annelies Frijters called me regularly from the TU Delft in the final stages of preparing my thesis and the arrangements that go with it. Her thoughtful reminders allowed me to sleep peacefully.

Without a good system, work of this nature cannot take place. At the UvA, I'd especially like to thank Gert Poletiek for his immediate help with anything I asked, and Casper Dik for happily handling all kinds of problems at odd hours. In Delft, I enjoyed working with Wouter Smaal in our efforts to keep the network running there. I'm particularly grateful for his effort, especially in the last two years, to relieve me of system tasks so I could work on my thesis.

The Pattern Recognition Group at the TU Delft was an especially stimulating atmosphere due to the quality of its scientific staff. It was a pleasure and an honor to work with Erik Bouts, Hans Buurman, Bob Duin, Pieter Jonker, Martin Kraaiveld, Jim Mullikin, Hans Netten, Wouter Schmidt, Karel Strasters, Fons Verbeek, Piet Verbeek, Ben Verwer, Lucas van Vliet, Albert Vossepoel, and Henri Vrooman. I particularly want to thank Lucas for helping me many times in both scientific and practical ways.

The less formal atmosphere at the UvA, with discussions in the hallway carried out by Richard van Balen, Rein van den Boomgaard, Leo Dorst, Theo Gevers, Dennis Koelma, Benno Mosterd, and Marcel Worring, was stimulating and enjoyable.

Without Rein's ready advice on the proper use of latex, gnuplot, postscript, etc., this thesis would look a lot different. Discussions with Richard on software issues related to image analysis were always lively and fun. Dennis ("I don't write bugs, I read 'em") and Benno helped me remove annoying little features from my programs.

Benno and I shared an office for several years in great harmony. He always knew when to let me work and when to make a joke. He put up with my intermittent messes and dirty dishes, cheered me up and gave me peppermints when I needed them most.

In the development of this thesis, it has been beneficial to have had contact with a number of scientists in neighboring research institutes. Most especially, I want to thank Adrian Baddeley, for discussions on error metrics, Ton ten Kate, for cooperative efforts on user interfaces for image analysis, and Peter Nacken, for helping me notice the applicability of some techniques in different contexts.

The Department of Pure Juggling at the Circus voor Wiskunde & Informatica has been an entertaining distraction. The theoretical sessions, with the pinball machine used for demonstrations were often even more enlightening than the laboratories.

Many friends have been encouraging, understanding and fun. In particular, Yvette Oomen, Adrian Baddeley, Marijke Kaat, Susan Üsküdarlı, Julie Hewitt (albeit remote), Leo Dorst and Phyllis Crabill have brightened my days.

Last, but most certainly not least, Emma ("zal ik even voor je bellen?") van der Meulen has been helpful in every way. If I mention that she cooked me tasty meals, helped with page layout and figures, and rewrote my Dutch summary so Dutch people could read it too, then at least I made a small start. Emma is a friend of the finest kind. To say more would be to say less.

Curriculum Vitae

Carol Orange was born on December 7, 1958 in Salem, Oregon. She acquired her Washington High School diploma in June, 1976 based on her work at Clinton Street School and Willamette Learning Center in Portland, Oregon. In 1980, after gaining some work experience, she began her university studies at Reed College. She concluded her thesis entitled "The Schnirelmann Result and the $\alpha+\beta$ Theorem" in 1984 at Reed, and obtained the Bachelor of Arts degree in Mathematics accompanied by a letter of commendation for academic excellence.

From 1984 to 1989, she worked as a UNIX systems programmer first at Lucasfilm, Ltd. in San Rafeal, California, and later at the Center for Mathematics and Computer Science in Amsterdam (CWI). In 1989, she began her PhD work in the Pattern Recognition Group of the Applied Physics Department at the Delft University of Technology. From 1989 to 1994, part of her research was performed in the Computer Systems Group of the Department of Mathematics and Computer Science at the University of Amsterdam.

Stellingen

behorende bij het proefschrift

Supervised Boundary Formation

Carol Orange

31 oktober 1994

1. Zowel het soort fouten als de grootte van de fouten die een gebruiker zal maken bij het interactief schetsen van de grens van een object in een beeld kan worden voorspeld.

2. Een door een gebruiker gemaakte schets van de grens van een object, kan als basis dienen voor het ontwikkelen van een model van de beeldfunctie in de omgeving van de grens. Dit model kan vervolgens voor beeldsegmentatie worden gebruikt.

3. Als de eigenschappen van een gebruiker met betrekking tot het schetsen van objectgrenzen systematisch worden gemodelleerd, kan de onzekerheid over de resultaten van interactieve beeldsegmentatie worden weggenomen.

4. "Supervised boundary formation" zoals in dit proefschift wordt voorgesteld kan niet veralgemeniseerd worden tot "supervised *surface* formation" voor drie dimensionale beeldsegmentatie, omdat een drie dimensionale dataset niet nauwkeurig kan worden weergegeven in twee dimensies. "Supervised boundary formation" kan echter als uitgangspunt dienen voor segmentatie van een driedimensionaal beeld als het op één of meerdere beelvlakken toegepast wordt.

5. Zonder a priori kennis van de beeldfunctie in het overgangsgebied tussen object en achtergrond, kan een schets van een gebruiker niet tijdens het tekenen worden gecorrigeerd.

6. Als een punt gerepresenteerd wordt door $a = \{x(a) = r_a \cos\theta_a, y(a) = r_a \sin\theta_a\}$, dan definieert

$$d(a,b) = \begin{cases} |\theta_a - \theta_b| \cdot \min\{r_a, r_b\} + |r_a - r_b| & \text{als } |\theta_a - \theta_b| \leq 2 \\ r_a + r_b & \text{anders} \end{cases}$$

een metriek, en als het Centraal Station als de oorsprong wordt genomen, benadert deze functie de "city block distance" in het centrum van Amsterdam veel beter dan de functie

$$d(a,b) = |x(a) - x(b)| + |y(a) - y(b)|,$$

die geschikt is om de "city block distance" in Portland, Oregon te benaderen.

7. Jongleren vereist maar een zeer beperkte hand-oog coordinatie, noch draagt het bij aan een significante verbetering daarvan.

8. Als een reiziger onderweg niet slaapt:

   - neemt bij een reis naar het westen de sterkte van de jetlag logaritmisch toe als functie van het tijdverschil.
   - neemt bij een reis naar het oosten de sterkte van de jetlag exponentieel toe als functie van het tijdverschil.

9. Iemand die kan leren schaatsen kan ook leren fietsen op een éénwieler.

10. Het generaliseren over mensen in andere landen dat af en toe opduikt in de Nederlandse Journalistiek en in Nederlands conversatie, is in strijd met het beeld van ruimdenkendheid en wereldwijsheid wat diegenen die zo generaliseren vaak van zich zelf hebben.

1. The type and extent of errors that a user will make in the interactive specification of an image object boundary path can be predicted.

2. A user defined sketch of an image object boundary can be used to extract a model of the image function in the neighborhood of the boundary which is sufficient for subsequent segmentation.

3. By approaching the user modeling issues in image object boundary formation in a systematic fashion, the uncertainty about the results of interactive segmentation can be eliminated.

4. Supervised boundary formation as presented in this thesis cannot be generalized to supervised *surface* formation for the segmentation of three dimensional images because a three dimensional data set cannot be accurately represented in two dimensions. However, if applied to one or more image slices, supervised boundary formation can serve as a basis for the segmentation of a three dimensional image.

5. Without a priori knowledge of the image function in the object to background transition region, dynamic correction of a user sketch of an object boundary as it is drawn is not possible.

6. If a point is denoted by $a = \{x(a) = r_a \cos\theta_a, y(a) = r_a \sin\theta_a\}$, then

$$d(a,b) = \begin{cases} |\theta_a - \theta_b| \cdot \min(r_a, r_b) + |r_a - r_b| & \text{if } |\theta_a - \theta_b| \leq 2 \\ r_a + r_b & \text{otherwise} \end{cases}$$

   defines a metric, and if Central Station is defined as the origin, it provides a much better approximation to the city block distance in the Amsterdam city center than the well known

$$d(a,b) = |x(a) - x(b)| + |y(a) - y(b)|,$$

   which is suited to approximate the city block distance in Portland, Oregon.

7. Juggling neither requires a significant level of hand-eye coordination, nor does it improve one's coordination significantly.

8. If an air traveler does not sleep enroute, then:

   - when traveling west, the severity of jet lag increases logarithmically as a function of the number of hours in the time change.
   - when traveling east, the severity of jet lag increases exponentially as a function of the number of hours in the time change.

9. Anyone who can learn to ice skate can learn to ride a unicycle.

10. The generalizations about people from other countries which sometimes creep into Dutch journalism and conversation are in conflict with the open-minded and worldly view those who formulate the generalizations often have of themselves.