Improved Segmentation of Salient Closed Contours

by

Kanglin Xu

M.S., Mechanical Engineering,
Shanghai University of Science and Technology, 1990

M.S., Mechanical Engineering, University of New Mexico, 2000

THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Computer Science

The University of New Mexico

Albuquerque, New Mexico

August, 2002

Dedication

I wish to dedicate this thesis to my wife, Min, who is expecting our first child.
I also wish to present this thesis to our baby
as a special gift for his first birthday.

Acknowledgments

First, I would like to thank my advisor, Professor Williams, who has given me a lot of help, including both academic direction and financial support, since I began this project. Without his direction and help, it would have been impossible for me to finish this thesis. Second, thank to my former advisor, Professor Shahinpoor, who provided me with the first year's financial support as well as an office, I could complete my studies in the Department of Computer Science. In this thesis I used many resources and I wish to thank their authors, developers or owners, including Professor Stefanovic, who took pictures for my experiments.

Improved Segmentation of Salient Closed Contours

by

Kanglin Xu

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Computer Science

The University of New Mexico

Albuquerque, New Mexico

August, 2002

Improved Segmentation of Salient Closed Contours

by

Kanglin Xu

M.S., Mechanical Engineering,

Shanghai University of Science and Technology, 1990

M.S., Mechanical Engineering, University of New Mexico, 2000

M.S., Computer Science, University of New Mexico, 2002

Abstract

The segmentation we describe is based on a distribution of contours traced by particle moving in directions undergoing Brownian motion. Using a saliency measure based on the global property of contour closure, we describe an improved method for the segmentation problem. We first integrate an optimization algorithm into the segmentation algorithm to find one of the three important parameters: the particle's speed, γ. We also analyze the properties of the affinity matrix, whose elements are the edge-to-edge transition probabilities, from a mathematical point of view and give formal proofs. In addition, we correct some problems in the existing algorithm. Our experiments show that the particle's speed, γ, found by the optimization algorithm, is reasonable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many objects in the physical world are bounded by smooth closed contours. Human visual systems are able to perceive the bounding contours of objects even in the absence of contrast due to occlusion. In computer vision, we attempt to emulate the way the human visual system behaves. Given an input image, we can segment different objects in it based on a computational model. The purpose of segmentation is to partition a set of edges into subsets so that the edges in different subsets bound different objects in the scene. In this thesis, we call this the *segmentation problem*.

In prior work, Thornber and Williams[16] use generalized particle trajectories to derive an analytic expression characterizing the probability distribution of boundary-completion shapes. This expression provides the basic foundation for the solution of the segmentation problem described in this thesis. A problem related to the segmentation problem is the saliency problem. The goal of the saliency problem is to assign a value to each edge which is correlated with whether that edge belongs to a shape or is background noise. There are many saliency measures described in the literature. Williams and Thornber[17] implement and compare these saliency methods and describe a new method based on the relative number of closed random walks which visit an edge. We believe that we can solve the segmentation problem using the saliency method to isolate the edges of the single most salient object from the background noise or from edges belonging to less salient objects. We could use a fixed threshold to separate the desired edges. However, for most problems a single threshold is not adequate. For example, it is difficult to obtain a good solution if two object contours have similar saliencies.

Many existing approaches have used graph-based search techniques to find closed contours. The entries of the affinity matrix, from which the graph is built, are the probabilities that a smooth contour passes through a pair of edges. The affinity values are based on the Gestalt principles of proximity and smooth-continuation and derive from the local properties of position and orientation of the two edges. Mahamud, Thornber and Williams[9] show that the effectiveness of these approaches is limited. In addition to the local properties, proximity and smooth-continuation, Mahamud *et al.*[9] use an approach based on global saliency computation. They use the local affinity measure to compute a global saliency measure. Concretely, they first compute the affinity matrix, $\mathbf{P}$, and the eigenvector associated with its maximum eigenvalue, and then compute a link saliency matrix, $\mathbf{C}$. Only after the computation of the global saliency measure, do they use a depth-first-search to find the strongly connected components. It should be noted here that instead of a fixed threshold, Mahamud *et al.*[8] use a dynamic threshold to isolate the most salient edges.

In this thesis, we improve on Mahamud *et al.*'s approach[9] to the segmentation problem. We modify their method in the following ways. (1) We use an algorithm to find one parameter, the particle's speed, automatically. (2) The properties of the affinity matrix, $\mathbf{P}$, and saliency matrix, $\mathbf{C}$, are very important because they directly affect the computational efficiency and can reduce the amount of computation greatly. We analyze these properties in detail and give formal proofs. (3) Mahamud *et al.* use the *biggest jump* to isolate the most salient edges. In practice, we are unable to find a simple cutoff. It follows that segmentation based on the idea of the biggest jump is not feasible. Instead of the biggest jump, we use the magnitude of the k-largest link saliency for $k = 2$, or $k = 3$, as a fixed threshold to isolate the most salient edges. (4) Finally, Mahamud *et al.*[8] use a set intersection operation to identify closed contours. But in some cases, this approach leads to problems. We solve this problem.

The remainder of this thesis is organized as follows: Chapter 2 presents the basic algorithm for solving the segmentation problem. Chapter 3 introduces the modified algorithm and the method to find γ, the particle's speed. Chapter 4 gives experimental results. Chapter 5 discusses future work, and Chapter 6 offers concluding remarks.

Chapter 2

An Algorithm for Segmentation

In this chapter, we describe in detail the algorithm for solving the segmentation problem. First, we define the saliency measure used and present its properties. Then we describe how to integrate it into a graph-based search to perform segmentation.

2.1 Saliency Measure

The following formulation is due to Williams and Thornber[18]. Suppose we have an input consisting of N edges. $\vec{x}_i$ is the location and θ_i is the orientation of edge i. Now we add another N edges. For each edge i, we add edge $\bar{\imath}$ and let its location be the same as that of i but its orientation be equal to *i's* orientation plus π, that is, $\vec{x}_i = \vec{x}_{\bar{\imath}}$ and $\theta_i = \theta_{\bar{\imath}} + \pi$. This step is necessary because it ensures that a contour arriving in a given direction must exit in the same direction. Basically, it ensures that tangent continuity is satisfied. Without this mechanism, it is possible to obtain a contour containing *cusps* (i.e., reversals in direction). Williams and Thornber[18] give a more detailed explanation.

Given two directed edges, i and j, Thornber and Williams[16] derive an expression for the probability, $P(j|i)$, that a particle moving with constant speed in a direction given by a Brownian motion will travel from edge i to edge j and visit $n-1$ intermediate edges. The equation for $P(j|i)$ is found in Appendix A. From Williams and Thornber[18], we know that $P(j|i)$ is dependent on three parameters in addition to the locations and orientations of edges i and j: (1) the half-life τ, (2) the variance T of the directional change of the particle, and (3) the speed of the particle, γ, and the locations and orientations of edges i and j. The half-life, τ, models the principle proximity while the variance, T, models smooth-continuation. These three parameters determine the distribution of shapes.

2.1.1 Edge Saliency

In this thesis, the saliency of edge i is defined to be the relative number of closed contours that pass through this edge. As in Williams and Thornber[18], we observe that any closed contour of infinite length can be decomposed into an infinite number of closed contours which visit edge i exactly once. Imagine that the infinite length contour which passes through edge i multiple times consists of an infinite number of closed contours which pass through edge i exactly once. If we assume that the closed contour begins and ends at edge, i, we can write the definition of saliency as follows

$$c_i = \lim_{t\to\infty} \frac{(\mathbf{P}^t)_{ii}}{\sum_j (\mathbf{P}^t)_{jj}}$$

where $\mathbf{P}$ is the affinity matrix whose elements are the edge-to-edge transition probabilities given by $P(j|i)$. Williams and Thornber[18] have shown that the edge saliency based on the above definition can be evaluated as

$$c_i = \frac{x_i \bar{x}_i}{\sum_j x_j \bar{x}_j}$$

where $\mathbf{x}$ is the right eigenvector corresponding to the largest eigenvalue of the affinity matrix, $\mathbf{P}$, and $\bar{\mathbf{x}}$ is the corresponding left eigenvector. If we do not consider the constant factor, $\sum_j x_j\bar{x}_j$, or normalize the eigenvector such that $\sum_j x_j\bar{x}_j = 1$, we have $c_i = x_i\bar{x}_i$.

2.1.2 Properties of the Affinity Matrix

The algorithm for solving the segmentation problem requires a large number of floating point operations, especially for large real images. Fortunately, we can use the properties of the affinity matrix to reduce the amount of computation greatly. Although previous papers[9, 18, 19] mention this idea, none of them give a detailed analysis and an explicit approach. Because this analysis is essential to efficient calculation, we include it here.

Lemma 1 *If edge i and edge $\bar{\imath}$ have an identical location but opposite directions, the affinity matrix $\mathbf{P}$ which is created using the equations described by Williams and Thornber[18] has the following properties:*

1. $P_{ij} = P_{\bar{\jmath}\bar{\imath}}$. *This is termed reversal symmetry.*

2. *In the above equation, we can substitute i for $\bar{\imath}$ everywhere, and $\bar{\imath}$ for i everywhere.*

3. $P_{\bar{\imath}i} = P_{i\bar{\imath}}$ *is constant, and* $P_{ii} = P_{\bar{\imath}\bar{\imath}}$ *is constant (for all $i = 1...n$)*

Proof. As mentioned before, the affinity, P_{ji}, is dependent on three parameters as well as the locations and orientations of edges i and j. If we focus on these three parameters, from the expression given by Williams and Thornber[18] we know that the affinity, P_{ji}, is determined by three factors related to the locations and orientations of edges i and j. Let these three factors be a, b, c:

$$
\begin{aligned}
a &= \frac{2+\cos(\theta_j-\theta_i)}{3} \\
b &= \frac{x_{ji}(\cos\theta_j+\cos\theta_i)+y_{ji}(\sin\theta_j+\sin\theta_i)}{\gamma} \\
c &= \frac{(x_{ji}^2+y_{ji}^2)}{\gamma^2} \\
x_{ji} &= x_j-x_i \\
y_{ji} &= y_j-y_i.
\end{aligned}
$$

Given the above equations, the proof is very straightforward. We just substitute $x_i = x_{\bar{\imath}}$, $y_i = y_{\bar{\imath}}$, and $\theta_{\bar{\imath}} = \theta_i + \pi$ into the above three equations. Since the only difference between edge, i, and edge, $\bar{\imath}$, is its orientation, and because the function for calculating c is even, we need to consider just the two factors a and b:

$$
\begin{aligned}
a_{ij} &= \frac{2+\cos(\theta_i-\theta_j)}{3} \\
b_{ij} &= \frac{x_{ij}(\cos\theta_i+\cos\theta_j)+y_{ij}(\sin\theta_i+\sin\theta_j)}{\gamma}
\end{aligned}
$$

Similarly, we have $a_{\overline{ji}}$ and $b_{\overline{ji}}$ for $P_{\overline{ji}}$ as follows:

$$
\begin{aligned}
a_{\overline{ji}} &= \frac{2+\cos(\theta_j+\pi-\theta_i-\pi)}{3} = \frac{2+\cos(\theta_i-\theta_j)}{3} \\
b_{\overline{ji}} &= \frac{x_{ji}[\cos(\theta_j+\pi)+\cos(\theta_i+\pi)]+y_{ji}[\sin(\theta_j+\pi)+\sin(\theta_i+\pi)]}{\gamma} \\
&= \frac{x_{ij}(\cos\theta_i+\cos\theta_j)+y_{ij}(\sin\theta_i+\sin\theta_j)}{\gamma}
\end{aligned}
$$

Finally, we have $P_{ij} = P_{\overline{ji}}$. Note that the *reversal symmetry* property means that the probability that any particle travels along a curve starting from edge, j, and ending in edge, i, is the same as that of a particle traveling from edge, $\bar{\imath}$, to edge, $\bar{\jmath}$.

Similarly, we can obtain $P_{\bar{i}j} = P_{\bar{j}i}$ and $P_{i\bar{j}} = P_{j\bar{i}}$. Therefore, Property *2* is also correct. Finally, since $\theta_i = \theta_{\bar{i}} + \pi$, a_{ii} and b_{ii} are constant. Therefore, Property *3* is proved.

Consider an input pattern consisting of N edges. We construct a set of $2N$ states, S. For each edge, i, in the input pattern, there are two states, i and $\bar{i}$, in S. If we arrange the set, S, so that the order of the first half of the $2N$ elements is the same as the input order (i.e., edge *i's*) and the second N of the *2N* elements are edges with opposite directions to each element in the first half (i.e., edge $\bar{i}'s$), we obtain the following lemma:

Lemma 2 *Given an input vector $A[1..2N]$ as described above, if we use the following loops*

$$\begin{aligned}&\mathit{for}(i = 1; i \leq 2N; i++)\\&\quad \mathit{for}(j = 1; j \leq 2N; j++)\\&\qquad P(i|j) = \mathit{function}(A[i], A[j])\end{aligned}$$

to create the affinity matrix, we obtain $\mathbf{P}$ *as follows:*

$$\mathbf{P} = \begin{bmatrix} \{P_{ij}\} & \{P_{i\bar{j}}\} \\ \{P_{\bar{i}j}\} & \{P_{\bar{i}\bar{j}}\} \end{bmatrix}$$

where

$$\{P_{ij}\} = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N1} & P_{N2} & \cdots & P_{NN} \end{bmatrix}$$

$$\{P_{i\bar{j}}\} = \begin{bmatrix} P_{1\bar{1}} & P_{1\bar{2}} & \cdots & P_{1\bar{N}} \\ P_{2\bar{1}} & P_{2\bar{2}} & \cdots & P_{2\bar{N}} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N\bar{1}} & P_{N\bar{2}} & \cdots & P_{N\bar{N}} \end{bmatrix}$$

$$\{P_{\bar{i}j}\} = \begin{bmatrix} P_{\bar{1}1} & P_{\bar{1}2} & \cdots & P_{\bar{1}N} \\ P_{\bar{2}1} & P_{\bar{2}2} & \cdots & P_{\bar{2}N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{\bar{N}1} & P_{\bar{N}2} & \cdots & P_{\bar{N}N} \end{bmatrix}$$

$$\{P_{\bar{i}\bar{j}}\} = \begin{bmatrix} P_{\bar{1}\bar{1}} & P_{\bar{1}\bar{2}} & \cdots & P_{\bar{1}\bar{N}} \\ P_{\bar{2}\bar{1}} & P_{\bar{2}\bar{2}} & \cdots & P_{\bar{2}\bar{N}} \\ \vdots & \vdots & \ddots & \vdots \\ P_{\bar{N}\bar{1}} & P_{\bar{N}\bar{2}} & \cdots & P_{\bar{N}\bar{N}} \end{bmatrix}$$

Proof: The given loops implement the product of a $2N \times 1$ matrix and a $1 \times 2N$ matrix, i.e., an outer product. Assume the input vector to be

$$\left(\begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_N \\ \hline x_{\bar{1}} \\ x_{\bar{2}} \\ \vdots \\ x_{\bar{N}} \end{array} \right).$$

Then the above loops perform

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ \hline x_{\bar{1}} \\ x_{\bar{2}} \\ \vdots \\ x_{\bar{N}} \end{pmatrix} \left(\begin{array}{cccc|cccc} x_1 & x_2 & \cdots & x_N & x_{\bar{1}} & x_{\bar{2}} & \cdots & x_{\bar{N}} \end{array} \right)$$

to create a matrix of the form:

$$\mathbf{P} = \begin{bmatrix} \{P_{ij}\} & \{P_{i\bar{j}}\} \\ \{P_{\bar{i}j}\} & \{P_{\bar{i}\bar{j}}\} \end{bmatrix}.$$

Theorem 1 *Suppose we use the method described in Lemma 2 to construct the affinity matrix* $\mathbf{P}$ *and* $\mathbf{x}$ *is an eigenvector of* $\mathbf{P}$ *associated with eigenvalue,* λ*, where*

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \\ \hline x_{N+1} \\ x_{N+2} \\ \vdots \\ x_{2N} \end{pmatrix}.$$

Then the eigenvector of $\mathbf{P}^T$ *associated with the same eigenvalue* λ *is*

$$\begin{pmatrix} x_{N+1} \\ x_{N+2} \\ \vdots \\ x_{2N} \\ \hline x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}.$$

Proof. Since we use the same method as Lemma 2 to construct the affinity matrix, it follows that

$$\mathbf{P} = \begin{bmatrix} \{P_{ij}\} & \{P_{i\bar{j}}\} \\ \{P_{\bar{i}j}\} & \{P_{\bar{i}\bar{j}}\} \end{bmatrix}$$

where $\{P_{ij}\}$, $\{P_{i\bar{j}}\}$, $\{P_{\bar{i}j}\}$ and $\{P_{\bar{i}\bar{j}}\}$ are four $N \times N$ matrix whose elements are P_{ij}, $P_{i\bar{j}}$, $P_{\bar{i}j}$ and $P_{\bar{i}\bar{j}}$ respectively. Let $\mathbf{x}$ be an eigenvector of the matrix, $\mathbf{P}$,

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix}$$

where $\mathbf{x_1}$ consists of the first N elements of the vector $\mathbf{x}$ and $\mathbf{x_2}$ consists of the second N elements of the vector $\mathbf{x}$, and

$$\mathbf{R} = \begin{bmatrix} \mathbf{0}_N & \mathbf{I}_N \\ \mathbf{I}_N & \mathbf{0}_N \end{bmatrix}$$

where $\mathbf{0}_N$ is the $N \times N$ zero matrix and $\mathbf{I}_N$ is the $N \times N$ identity.
We have $\mathbf{R}^T = \mathbf{R}$ and

$$\mathbf{R}^2 = \begin{bmatrix} \mathbf{0}_N & \mathbf{I}_N \\ \mathbf{I}_N & \mathbf{0}_N \end{bmatrix} \begin{bmatrix} \mathbf{0}_N & \mathbf{I}_N \\ \mathbf{I}_N & \mathbf{0}_N \end{bmatrix} = \mathbf{I}_{2N}$$

Therefore, $\mathbf{R}^{-1} = \mathbf{R} = \mathbf{R}^T$. If we left-multiply $\mathbf{P}$ by $\mathbf{R}$, we obtain

$$\mathbf{RP} = \begin{bmatrix} \mathbf{0}_N & \mathbf{I}_N \\ \mathbf{I}_N & \mathbf{0}_N \end{bmatrix} \begin{bmatrix} \{P_{ij}\} & \{P_{i\bar{j}}\} \\ \{P_{\bar{i}j}\} & \{P_{\bar{i}\bar{j}}\} \end{bmatrix} = \begin{bmatrix} \{P_{\bar{i}j}\} & \{P_{\bar{i}\bar{j}}\} \\ \{P_{ij}\} & \{P_{i\bar{j}}\} \end{bmatrix}$$

Compare the above solution with

$$\mathbf{P} = \begin{bmatrix} \{P_{ij}\} & \{P_{i\bar{j}}\} \\ \{P_{\bar{i}j}\} & \{P_{\bar{i}\bar{j}}\} \end{bmatrix}.$$

We have

$$(\mathbf{RP})_{ij} = P_{\bar{i}j}$$

and similarly, if we right-multiply $\mathbf{P}$ by $\mathbf{R}$ we obtain

$$(\mathbf{PR})_{ij} = P_{i\bar{j}}.$$

We will use these two equations for the following proof. From reversal symmetry $P_{\overline{ji}} = P_{ij}$ and $(\mathbf{PR})_{ij} = P_{i\bar{j}}$, we have

$$(\mathbf{P}^T\mathbf{R})_{\overline{ji}} = (\mathbf{P}^T\mathbf{R})_{ij} = (\mathbf{P}^T)_{i\bar{j}},$$

and from $(\mathbf{P}^T\mathbf{R})_{\overline{ji}} = (\mathbf{P}^T)_{i\bar{j}}$ and $(\mathbf{RP})_{ij} = P_{\bar{i}j}$, we have

$$(\mathbf{RP}^T\mathbf{R})_{\overline{ji}} = (\mathbf{RP}^T)_{i\bar{j}} = (\mathbf{P}^T)_{\bar{i}\bar{j}} = P_{\overline{ji}}.$$

Therefore,

$$\mathbf{RP}^T\mathbf{R} = \mathbf{P}.$$

From $\mathbf{Px} = \lambda\mathbf{x}$,

$$\lambda\mathbf{x} = \mathbf{RP}^T\mathbf{Rx}.$$

Since $\mathbf{R}^2 = \mathbf{I}$,

$$\lambda\mathbf{R}^2\mathbf{x} = \mathbf{RP}^T\mathbf{Rx}.$$

Therefore,

$$\lambda \mathbf{R}\mathbf{x} = \mathbf{P}^T \mathbf{R}\mathbf{x}.$$

Finally, since

$$\mathbf{R}\mathbf{x} = \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_1 \end{pmatrix},$$

we finish our proof.

Note that it is necessary to use $\bar{\imath}$ notation to represent some elements of the eigenvectors in order to define link saliency in the next section. Actually, we have used these notations before. We use $\mathbf{x}$ to represent the eigenvector of the matrix, $\mathbf{P}$, and $\bar{\mathbf{x}}$ to represent the eigenvector of the matrix, $\mathbf{P}^T$. In addition, we use i to represent the index which is between 1 and N, and $\bar{\imath}$ to represent the index which is between N and $2N$. For example, $\bar{x}_{\bar{\imath}}$ is the $(i+N)$-th element of the eigenvector of the $\mathbf{P}^T$ matrix. With this notation, we obtain $\bar{s}_i = s_{\bar{\imath}}$ and $\bar{s}_{\bar{\imath}} = s_i$ directly from the above theorem.

Now we can simplify the edge saliency definition. Recall that

$$c_i = \frac{x_i \bar{x}_i}{\sum_j x_j \bar{x}_j}$$

which involves both the right and left eigenvectors. With the relation between $\mathbf{x}$ and $\bar{\mathbf{x}}$, we can rewrite the edge saliency definition as

$$c_i = \frac{x_i x_{(i \pm N) \bmod N}}{\sum_j x_j x_{(j \pm N) \bmod N}}$$

which is only dependent on the right eigenvector with largest positive real eigenvalue of affinity matrix, $\mathbf{P}$.

Actually, we can reduce computation further. For the $\mathbf{P}$ matrix, there are only $2N^2 - 2N + 2$ distinct elements, rather than $4N^2$. This can be demonstrated quite easily. From the properties of the affinity matrix, $\mathbf{P}$, we know that submatrices $\{P_{i\bar{j}}\}$ and $\{P_{\bar{i}j}\}$ are symmetric (see Appendix B). We also know that the elements in the upper triangle of $\{P_{ij}\}$ are equal to those in the lower triangle of $\{P_{\bar{i}\bar{j}}\}$, and that the elements in the lower triangle of $\{P_{ij}\}$ are equal to those in the upper triangle of $\{P_{\bar{i}\bar{j}}\}$(see Appendix B). Therefore, if we do not consider the diagonal elements of the four matrices, then the total number of distinct elements is $2N^2 - 2N$. In addition, we also can see that the diagonal elements of matrix $\mathbf{P}$ are constant and so are the diagonal elements of the submatrices $\{P_{i\bar{j}}\}$ and $\{P_{\bar{i}j}\}$. These results are the consequences of Lemma 1. Finally, we have $2N^2 - 2N + 2$ distinct elements.

2.1.3 Link Saliency

As mentioned before, Mahamud *et al.*[9] use an approach based on a global saliency measure, called *link saliency*, which is related to the relative likelihood that closed contours pass through edges j and i successively. They first compute the affinity matrix, P, and the eigenvector associated with its maximum eigenvalue, and then compute the *link saliency* matrix, C. The *link saliency* matrix is used to create a directed graph. The segmentation problem is reduced to the problem of finding strongly-connected components of this graph. In this section we will discuss the properties of the *link saliency* matrix, $\mathbf{C}$.

Suppose we begin at edge i, immediately visit edge j, and then eventually return to i. We have the following mathematical expression:

$$C_{ij} = \lim_{n\to\infty} \frac{P^{n-1}(j|i)P(i|j)}{\sum_k P^{(n)}(k|k)}.$$

Williams and Thornber[18] solved this expression

$$C_{ij} = \frac{\bar{s}_i P_{ij} s_j}{\lambda \bar{\mathbf{s}}^T \mathbf{s}}.$$

If we use the approach mentioned in Section *2.1.2* and we normalize the eigenvectors so that $\sum_i s_i \bar{s}_i = 1$, we can simplify the above expression as follows:

$$C_{ij} = \frac{s_{(i+N) \bmod N} P_{ij} s_j}{\lambda}.$$

which is only dependent on the right eigenvector.

As with the **P** matrix, the **C** matrix has many interesting properties.

Theorem 2 *The* **C** *matrix created using the above equation has the following properties:*

1. *Conservation property: The number of closed contours entering edge i equals the number of closed contours leaving edge i.*
2. *Reversal symmetry property: $C_{ij} = C_{\bar{j}\bar{i}}$*
3. *In the above equation, we can substitute i for $\bar{\imath}$ everywhere, and $\bar{\imath}$ for i everywhere.*

Proof: For convenience, we suppose $\sum_i s_i \bar{s}_i = 1$. This assumption involves no loss of generality because any eigenvector can be normalized so that it has unit length. Property *1* has been shown by Williams and Thornber[18]. We prove Properties 2 and 3. Recall that $P_{ij} = P_{\bar{j}\bar{i}}$, $\bar{s}_i = s_{\bar{\imath}}$ and $\bar{s}_{\bar{\imath}} = s_i$. Thus, it is easy to verify that

$$C_{ij} = \frac{\bar{s}_i P_{ij} s_j}{\lambda} = \frac{s_{\bar{\imath}} P_{\bar{j}\bar{i}} \bar{s}_{\bar{j}}}{\lambda} = \frac{\bar{s}_{\bar{j}} P_{\bar{j}\bar{i}} s_{\bar{\imath}}}{\lambda} = C_{\bar{j}\bar{i}}.$$

In $P_{ij} = P_{\bar{j}\bar{i}}$, we can substitute i for $\bar{\imath}$ everywhere, and $\bar{\imath}$ for i everywhere. Thus, we can obtain $P_{i\bar{j}} = P_{j\bar{\imath}}$ and $P_{\bar{\imath}j} = P_{\bar{j}i}$ directly from $P_{ij} = P_{\bar{j}\bar{i}}$. Therefore,

$$C_{i\bar{j}} = \frac{\bar{s}_i P_{i\bar{j}} s_{\bar{j}}}{\lambda} = \frac{s_{\bar{\imath}} P_{j\bar{\imath}} \bar{s}_j}{\lambda} = \frac{\bar{s}_j P_{j\bar{\imath}} s_{\bar{\imath}}}{\lambda} = C_{j\bar{\imath}}.$$

Similarly, we can show that $C_{\bar{\imath}j} = C_{\bar{j}i}$, and thus Property 3 is proven.

2.2 Segmentation

As mentioned before, the goal of segmentation is to partition a set of edges into subsets so that the edges in different subsets bound different objects in the scene. One easy way to do this is to choose a fixed threshold, and to divide the set of edges into two subsets based on this threshold. If an edge has a saliency above the threshold, we put it into the subset, where the edges belong to the single most salient object. Otherwise, we put it into the subset including the edges of less salient objects and background noise. In this way, the most salient object can be isolated. Unfortunately, this method does not work in general. Mahamud *et al.*[9] show that in images in which two or more salient contours have comparable saliencies, no single threshold can be expected to isolate one contour from the others. For example, if two pears are identical, it is difficult to segment out the contour of one pear from that of another using a single threshold. To handle this situation, Mahamud *et al.*[9] modify the original segmentation algorithm. In the following section, we describe the modified algorithm of Mahamud *et al.*[9].

Consider the set of contours which link edges in the scene. We can construct a directed graph whose vertices correspond to the edges in the scene and create a directed link from edge, i to edge, j, if i and j are successive edges in some contour. Mahamud *et al.*[9] have shown that such a construction partitions the graph into a set of *strongly-connected* components with no links between any two components. Therefore, the segmentation problem can be reduced to the problem of finding a set of isolated strongly-connected components of a directed graph. From graph theory, we know that two vertices, u and v, of a directed graph $G = (V, A)$, are related if and only if $u = v$ or there exists a directed cycle passing through both vertices. This relation partitions the graph into subgraphs, $G_i = (V_i, A_i), i = 1, 2, ..., k$, where the V_i are the equivalence classes determined by the relation and the A_i are those edges of A that have both endpoints in V_i. Each of these subgraphs is called a *strongly* *connected component* of G. Stated simply, two vertices are in the same strongly connected component if and only if there exists a directed path from each vertex to the other. Note that one component may contain multiple contours since more than one contour may pass through the same edge.

Recall that the *link saliency* is the relative number of closed contours which visit edge i and j in succession. It provides a measure of whether two edges are salient, i.e., the likelihood that a contour passes through the pair of edges. Using this information, we can create a directed graph. Generic graph search techniques such as depth-first-search or modified depth-first-search can be used to find the set of strongly-connected components.

We can find all the strongly-connected components at once. Unfortunately, not all of the components found in this way provide reliable segmentations. The reason is that the dominant contours tend to suppress the saliencies of all of the other contours, and therefore its saliencies suppress the saliencies of less dominant contours. The other way is to extract out the contours one by one. We first extract out the most dominant contour which passes through the most salient edge. The strongly-connected component containing this most dominant contour is the set of edges reachable from the most salient edge. This set of edges can be easily found using depth-first-search of the directed graph. After extracting out the dominant component, we suppress it to find the next dominant component. To suppress the current component, we deflate the affinities of all links among the edges in the strongly-connected component. For example, if edges, i and j are edges in the component, we deflate the link $i \rightarrow j$ by setting $P_{ij} = 0$ and $P_{\overline{j}\overline{i}} = 0$. We iterate the process of depth-first search followed by deflation until all components are isolated.

Due to noise, strongly-connected components which include edges of the current most salient object may also contain edges which belong to less salient objects or to the background. We can eliminate these undesirable edges using an important property of strongly-connected components, namely that, *the set of edges in a strongly-connected component containing a given edge is the intersection of the set of edges reachable from the given edge and the set of edges reachable from the same edge when all links have their directions reversed.* Using the reversal-symmetry property, we can implement this operation as follows[9]: (1) Find A, the set of edges reachable from the most salient edge, i; (2) Find $\bar{A}$, the set of edges reachable from the edge, $\bar{\imath}$; and (3) Find the intersection of the two sets, $A \cap \bar{A}$. Note that this intersection operation may lead to the wrong solution in some cases. This will be explained in the next chapter.

The last problem Mahamud *et al.*[9] solved is how to select thresholds. Instead of a constant threshold, they used an adaptive threshold for the set of C_{ij}'s coming out of each edge, j. They found the threshold dynamically and used it to isolate the desired salient object. The following steps are used:

1. For edge, i, reachable from edge k, sort the set in order of increasing C_{ik}.

2. Find the place in the sorted list, where the largest jump, $|C_{ik} - C_{i+1,k}|$, between successive C_{ik}'s occurs.

3. Add the edge, k, to the reachable set A, i.e., $A \leftarrow \{k\}$.

4. For each C_{ik} after this jump, call these steps recursively and union the resulting sets with this set.

Note that when we implemented this method, we were unable to find a single threshold for the magnitude of the jump. Therefore, the segmentation which results is sometimes not as good as we would like. We will modify Mahamud *et al.*'s algorithm in the next chapter to fix these problems.

Chapter 3

An Improved Algorithm

In the last chapter, we described Mahamud *et al.*[8]'s approach to the segmentation problem. In addition, we analyzed and proved several properties of the affinity matrix, P, and *link saliency* matrix, C, independently. In this chapter, we will describe an approach to find the particle's speed, γ, and several additional improvements to the algorithm of Mahamud *et al*[8].

3.1 A Method to Find the Particle's Speed

We first will give the objective function used in our algorithm. We then will explain why we selected the Brent[2] algorithm for finding a minimum of a function of one variable.

3.1.1 The Objective Function

As mentioned in Chapter 2, the affinity, P_{ji}, from edge, i, to edge, j, is dependent on a particle's half-life, τ, the variance, T, of the directional change of the particle, and the particle's speed, γ. Therefore, λ, the largest eigenvalue of the affinity matrix, $\mathbf{P}$, is a function of τ, T, and γ. It is also a function of the position and direction of the set of edges. In this work, we will try to find one of these parameters, γ, automatically using an optimization process.

Before building the objective function for finding the particle's speed, we need to understand the physical meaning of the eigenvalue in our approach. From linear algebra, we know that the eigenvalue of a matrix, $\mathbf{P}$, are solutions to the equation $\det(\mathbf{P} - \lambda\mathbf{I}) = 0$. Now suppose that Γ is a closed path which passes through m directed edges and the probability that a particle following this path will reach directed edge, $\Gamma_{(i+1 \bmod m)}$, given that it is located at directed edge, Γ_i, equals $P(\Gamma_{(i+1 \bmod m)}|\Gamma_i)$. Williams and Thornber *et al.*[19] have shown that the eigenvalue is given by the following equation:[1]

$$\lambda(\Gamma) = \Big(\prod_{i=1}^{m} P(\Gamma_{(i+1 \bmod m)}|\Gamma_i)\Big)^{1/m}.$$

This is called the *geometric mean* of the transition probabilities in the closed path and gives the relation between the eigenvalue and contour saliency. The magnitude of the largest positive real eigenvalue of $\mathbf{P}$ is related to the saliency of the most salient closed contour. If we perform the logarithm operation and then change the sign on both sides of the above equation, we have

$$-\ln\lambda(\Gamma) = -\sum_{i=1}^{m} \ln P(\Gamma_{(i+1 \bmod m)}|\Gamma_i)/m$$

which is called the *average transition energy*. Therefore, maximizing the geometric mean of the transition probabilities is equivalent to minimizing the average transition energy. We apply the concept of average transition energy to defining an objective function. Through minimizing the average transition energy, we can find the suitable particle's speed, γ, for a specific segmentation problem.

[1] Assuming that the probability of a particle traveling from directed edge Γ_i to Γ_j when Γ_j does not immediately follow Γ_i on the closed path is negligible.

Basically, we have two options. The first is to use a single speed, γ, for extracting all of the objects in the given image. The other is to use different speeds for extracting different objects. For the second option, we first use the optimization algorithm to find γ_1 for the first object including the most salient edges in the given input image by minimizing the function, $E(\gamma) = -\ln \lambda(\gamma)$. We then need to update the input image and recompute the affinity matrix, $\mathbf{P}$. After updating the image and computing the new affinity matrix, we use the same method to find γ_2 for the second object including the most salient edges in the updated input image. In this way, we can find all the speeds. The objective function for this method is simple. However, this method has low efficiency because we have to recompute the affinity matrix after every object, and calculate and store the updated image. In addition, we cannot compare our improved algorithm with Mahamud's algorithm if we use different γ for different objects because Mahamud's algorithm uses a single γ for all of the objects in the given image. For these reasons, we use a single speed, γ. We define the objective function as follows:

$$E(\gamma) = -\sum_{i=1}^{n} \ln \lambda_i(\gamma)$$

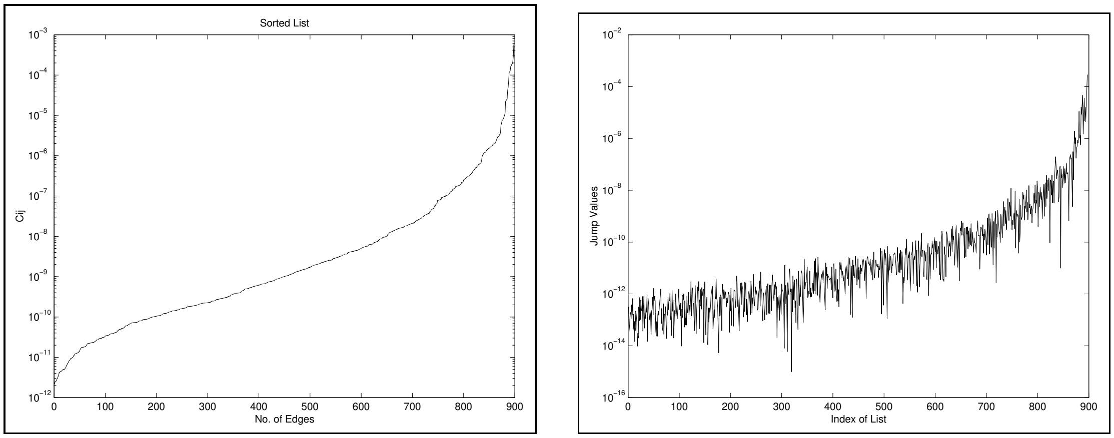
where λ_i is the largest eigenvalue corresponding to the most salient contour at the i-th iteration. By minimizing the objective function $E(\gamma)$, we suppress the saliencies of all other contours and background noise when we isolate the most salient contour at each iteration. Our experimental results in the next chapter show that this objective function can help us to find a suitable value for γ.

3.1.2 Optimization Algorithm

Because the relation between the particle's speed, γ, and the largest eigenvalue, λ is complicated, we would like to select an optimization algorithm which does not need analytic derivatives. Brent[2]'s algorithm is one of them. From [2], we know that the Fibonacci and golden section search methods have guaranteed linear convergence while successive linear parabolic interpolation processes do not always converge but under certain conditions converge superlinearly. Brent[2]'s algorithm combines both of them, and thus has the advantages of both methods, that is, convergence in a reasonable number of steps is guaranteed for any function, and on well-behaved functions convergence is superlinear, with order at least 1.3247. Therefore, we will use Brent[2]'s algorithm.

3.2 Threshold for Segmentation

Ideally we could use a fixed threshold to separate the first object from the second and remaining objects. But for most problems, a single threshold is not sufficient. To solve these problems, we need multiple thresholds. For example, given an image with three objects in it, we normally would need three thresholds and use one at a time for segmentation. Using multiple thresholds we can obtain better solutions. However, finding these thresholds is difficult. Mahamud *et al.*[8] developed an approach for finding the thresholds automatically. They integrate their approach into a graph search. At each iteration of the depth-first-search, they choose an adaptive threshold for the set of C_{ij}'s coming out of each edge, j. They then sort all of the C_{ij}'s coming out of edge, j, in increasing order. In the sorted list, they find the place where the jump between successive C_{ij}'s is largest and partition the list into two sublists at this point. The links corresponding to the C_{ij}'s occurring after this point are assumed to lie on the most salient closed contours and hence are selected. Those before this jump are discarded. We have observed that the biggest jump often happens between the last two elements of the list, or even if the biggest jump happens somewhere else it may not be the best place to partition the list. Figure 3.1 illustrates this situation. The increasing ordered list at the 12-th iteration for segmenting the double-pear



(a) Sorted C_{ij} list

(b) Jumps vs. Indices

Figure 3.1: Sorted List and Jumps.

image (see Figure 4.1) is shown in Figure 3.1(a). The x-axis is the index in the list and the y-axis is the magnitude of the C_{ij}'s coming out of edge, j, in increasing order. From the figure, we can see that there is no obvious cut-off in the curve. Figure 3.1(b) shows the magnitude of the jumps between successive elements in the ordered list and their locations. The jumps monotonically increase as the index increases and the biggest jump is located between the two largest elements. Note that the jumps in the figure are based on the differences between successive elements. We also tried to use the ratio between successive elements to measure the jump. The biggest jump in this case happens between the two lowest elements instead of the largest two. The reason is that the lowest element is too small as shown in Figure 3.1(a). Because the denominator of the ratio is almost zero, the ratio between the first two element is very large. In addition, finding the precise ratio becomes difficult if the denominator is too small because of the limits of machine precision.

In this thesis, we combine the approach of Mahamud *et al.* with that of fixed thresholds. We implement them within graph depth-first-search. For convenience, we also sort all the C_{ij}'s coming out of edge, j, in increasing order.[2] Instead of using the biggest jump to isolate the most salient edge, we threshold using the magnitude of the k-largest for $k = 1, 2, 3, 4$, or 5. Concretely, we partition the sorted list into two sub-lists. The first consists of all elements with magnitude less than the k-largest and the second consists of all the remaining elements including the k-largest. We discard the elements in the first sublist, and select those in the second sublist to represent links of the first salient closed contour. We repeat this step recursively using a fixed k until all of the salient closed contours in the image are isolated. Our experimental results show that k is usually three when there is less background noise, and is usually two when there is more background noise. Searching among a small number of integers is much easier than trying to find a suitable real valued threshold, especially multiple thresholds, because we have no prior knowledge of the threshold for a given problem. Using our approach, we usually obtain a better solution than the Mahamud's algorithm. This is shown in our experimental results in Chapter 4.

3.3 Correction of Mahamud's Algorithm

As mentioned before, due to background noise some of the C_{ij}'s might wrongly indicate that the strongly-connected component containing the most salient edge is connected to one or more of the other strongly-connected components. To eliminate the edges of other components, Mahamud *et al.*[8] use the intersection of the set of edges reachable from edge, i, and the set of edges reachable from edge, $\bar{\imath}$. When we implement Mahamud *et al.*'s algorithm and the improved algorithm, we find that simple set intersection leads to problems. In some cases, this intersection will lose many desired links. In the extreme, the result may be the empty set. Consider the following example as shown in Figure 3.2.

[2] it is not necessary to sort the entire list if we just want to find the k largest elements in the list.

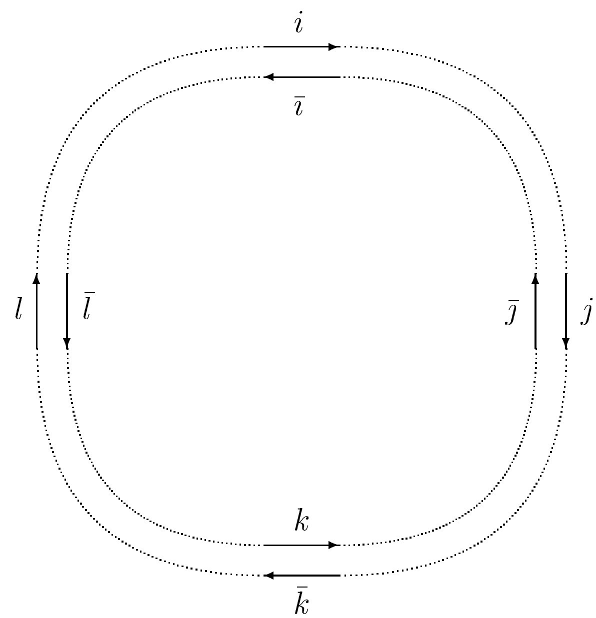


Figure 3.2: An Example.

Let A_i be the set of edges reachable from a given edge, i, and $A_{\bar{\imath}}$ be the set of edges reachable from $\bar{\imath}$. Then $A_i = \{i, j, \bar{k}, l\}$ and $A_{\bar{\imath}} = \{\bar{\imath}, \bar{\jmath}, k, \bar{l}\}$. The intersection of sets, A_i and $A_{\bar{\imath}}$ is empty. Obviously, this solution is wrong. Instead, we use

$$(A_i \cap \bar{A}_{\bar{\imath}}) \cup (\overline{A_i \cap \bar{A}_{\bar{\imath}}})$$

which is

$$(A_i \cap \bar{A}_{\bar{\imath}}) \cup (\bar{A}_i \cap A_{\bar{\imath}})$$

where $\bar{A}_i$ and $\bar{A}_{\bar{\imath}}$ are the complements of A_i and $A_{\bar{\imath}}$ respectively. Use of the above scheme can avoid losing the desired links. Now the solution for the given example is $\{i, j, k, l, \bar{\imath}, \bar{\jmath}, \bar{k}, \bar{l}\}$.

3.4 The Improved Algorithm

In the previous chapter, we introduced Mahamud *et al.*[8]'s algorithm. In this chapter, we discuss the optimization approach and the objective function we used, describe a new method for the threshold, and correct a defect in their algorithm. The improved algorithm for the segmentation problem is:

(a) main program

Input: Set of edges (x_i, y_i, θ_i) and parameters (T, τ)

Output: Segmented contours, and particle speed, γ.

Begin

1. Initialize

2. Segment

2.1 Create $\mathbf{P}$ matrix given parameters, T, γ, τ, x_i, y_i and θ_i.

2.2 Solve $\mathbf{Ps} = \lambda\mathbf{s}$ to find $\mathbf{s}$, $\bar{\mathbf{s}}$, and λ.

2.3 Compute edge saliencies $C_i = s_i\bar{s}_i$ and the most salient edge j.

2.4 Call DFS function to find A_j, the set of edges reachable from j.

2.5 Call DFS function to find $A_{\bar{\jmath}}$, the set of edges reachable from $\bar{\jmath}$.

2.6 Find $\bar{A}_j$ and $\bar{A}_{\bar{\jmath}}$.

2.7 Compute $(A_i \cap \bar{A}_{\bar{\imath}}) \cup (\bar{A}_i \cap A_{\bar{\imath}})$.

2.8 Update $\mathbf{P}$ matrix.

2.9 Repeat Steps *2.1* - *2.8* until all contours are isolated.

3. Compute the objective function, $E(\gamma)$.

4. Repeat Steps *1* - *3* until the termination condition is met.

5. Output the segmentation solutions and particle speed, γ.

End

(b) Depth-first-search (DFS) function

Input: Graph, the most salient edge j and integer $k = 1, 2, 3, 4,$ or 5

Output: Set of edges reachable from j

Begin

1. If j has not visited before then

1.1 Mark j visited.

1.2 Compute link saliencies $C_{ij} = \bar{s}_i P_{ij} s_j / \lambda$ for all i.

1.3 Sort the C_{ij}'s in increasing order.[3]

1.4 Put the edge, j, into set A.

1.5 Put the k-largest elements of C_{ij} onto the stack.

1.6 For each element, v, in the stack do

1.6.1 Call DFS recursively to find A', the set of edges reachable from v.

1.6.2 Compute $A = A \cup A'$.

2. Return A.

End

Before and after the above algorithm, we apply a preprocessing step and a post-processing step. In the preprocessing step, we use the Canny edge detector to find a set of edges. In the postprocessing step, we convert the numerical solution to a graphical solution and display it.

[3]The sorting is $O(n \log n)$. We could instead simply find the k largest elements, which is $O(n)$. In any case, the time is dominated by finding the eigenvalues, which is at least $O(n^2)$. The use of sorting here is purely for convenience in descriptions.

Chapter 4

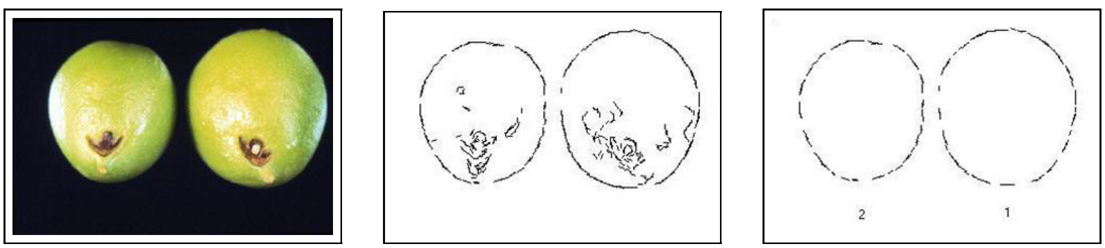
Experimental Results

To verify the improved algorithm, we conducted experiments with five real images. The first image (Figure 4.1(a)) was downloaded from a web page. The second image (Figure 4.2(a)) was obtained from Parker[12]. The last three images (Figures 4.3(a), 4.4(a), and 4.8(a)) were taken by Professors Stefanovic and Williams. All code was written in C or C++. The Canny edge detector code was copied from Parker's textbook, modified to output edge orientations. The codes for segmentation and optimization were implemented in GNU C++. These codes consist of six classes, which are *CubicEqn, MaxEigen, IntSet, List, ParaIn* and *Components*, and eight functions including *createP, getMaxCi, getResults, minin, runSeg, seg, updateP* and *main*. Additional code for graphical display was written in Microsoft Visual C++ 5.0. The number of edges returned by the Canny detector was found to be prohibitively large. To reduce the running time, we subsampled the input edges.

First we used the optimization algorithm to compute the particle's speed, γ. The results and iteration number are shown in Table 4.1. The range of γ which we minimized over was $[.01, 1.5]$ except for the *Coin* image, where the range was $[.01, 1.3]$. The segmentations for the five input images are shown in Figure 4.1(c),

image	number of iter.	particle speed γ	input range
Double Pears	15	0.87	.01-1.5
Coins	10	1.29	.01-1.3
Fruit Group 1	9	1.48	.01-1.5
Fruit Group 2	19	0.31	.01-1.5
Fruit Group 3	16	1.39	.01-1.5

Table 4.1: Particle's speed, γ, obtained by the optimization algorithm

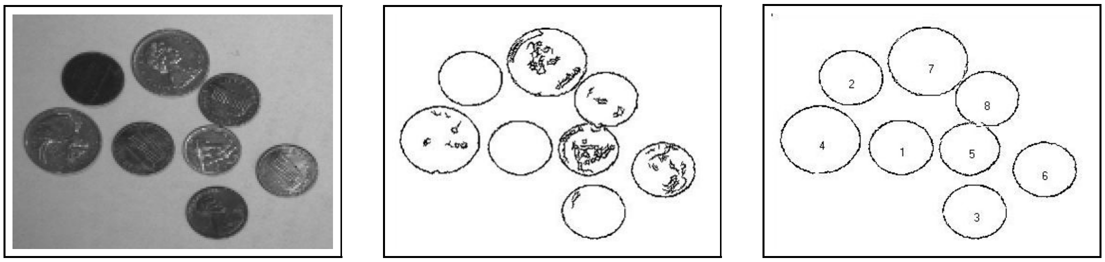


(a) (b) (c)

Figure 4.1: *Double Pears.* (a) Original image. (b) Edge input obtained by Canny detector. (c) Segmented contours numbered in the order in which they are extracted.

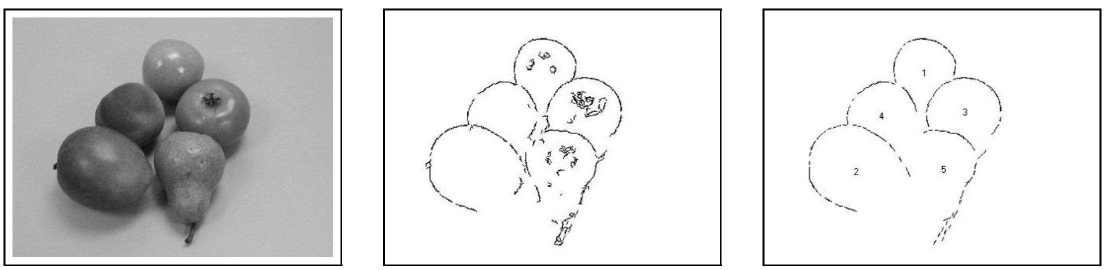
4.2(c), 4.3(c), 4.5(a), 4.6(a), 4.7(a), 4.9(a) and 4.10(a). For all images, the improved algorithm extracts out the salient contours. This is true even for *Fruit Group 1*, *Fruit Group 2* and *Fruit Group 3* where the contours are not completely closed, or partially overlap, which makes separation of the two objects difficult. In Figure 4.8, due to the poor contrast between the top-left object and its background, the Canny detector cannot find most of the edges. In addition, the orientations of input edges are not accurately estimated. Despite these facts, the algorithm segments out two fruits as shown in Figure 4.9(a) and 4.10(a). These results show that our approach can find the particle's speed, γ, automatically.

As we mentioned before, it is sometimes the case that Mahamud *et al.*'s approach, which is based on the biggest jump, can not find the correct solution because the



(a) (b) (c)

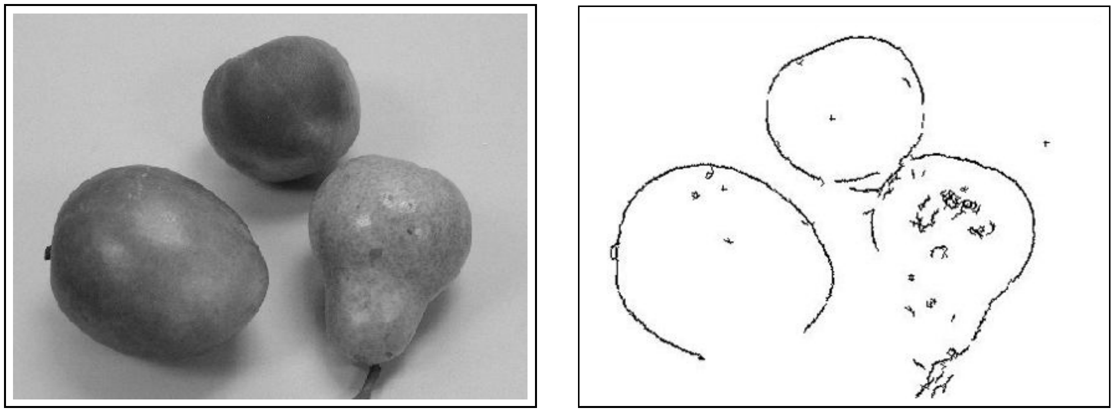
Figure 4.2: *Coins.* (a) Original image. (b) Edge input obtained by Canny detector. (c) Segmented contours numbered in the order in which they are extracted.



(a) (b) (c)

Figure 4.3: *Fruit Group 1.* (a) Original image. (b) Edge input obtained by Canny detector. (c) Segmented contours numbered in the order in which they are extracted.

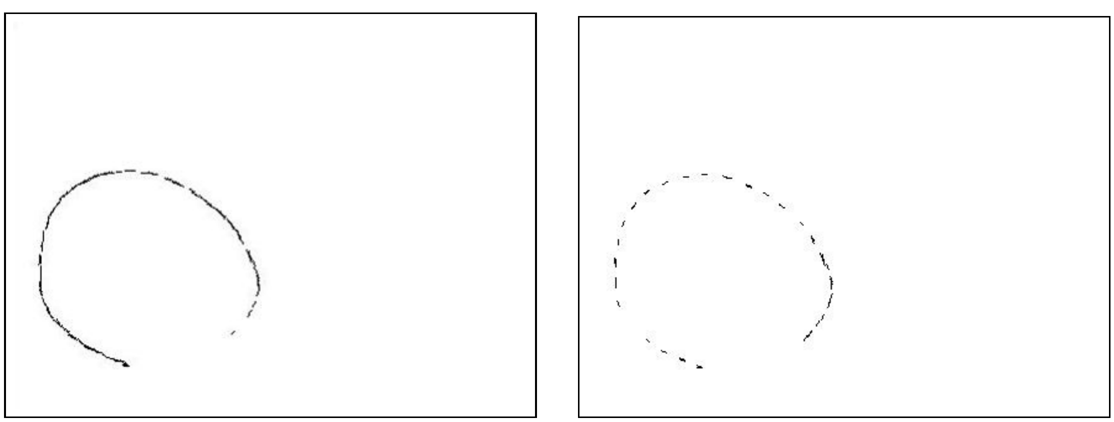
biggest jump usually happens between the last two elements. This will cause many desirable edges to be lost. Comparing subfigure (b) with subfigure (a) in Figures 4.5 and 4.6, we can see that Mahamud's solutions are less complete. Sometimes, the solution is so sparse that it does not look like an object anymore. We can also see this situation in Figure 4.9(b). A potentially more serious problem is that the omitted edges can interfere with the process of extracting the remaining closed contours. Unlike edges belonging to the background, the omitted edges have large pairwise probabilities. In Figure 4.7(b) we see that, after its third iteration, Mahamud's algorithm extracts the omitted edges of the first object rather than those of the



(a) (b)

Figure 4.4: *Fruit Group 2.* (a) Original image. (b) Edge input obtained by Canny detector.

third object, *i.e.*, the pear, as shown in Figure 4.7(a). Again, in Figure 4.10(b) after the second iteration, Mahamud's algorithm extracts the edges omitted from the first object instead of those of the second object, which the improved algorithm



(a) (b)

Figure 4.5: The first object isolated from *Fruit Group 2.* (a) Using the improved algorithm. (b) Using the original algorithm.

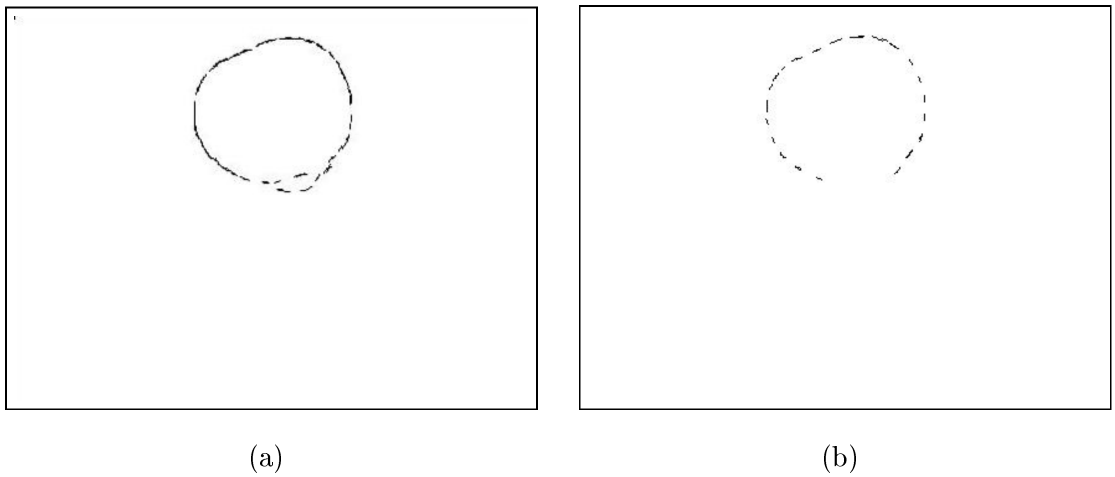


Figure 4.6: The second object isolated from *Fruit Group 2*. (a) Using the improved algorithm. (b) Using the original algorithm.

successfully extracts as shown in Figure 4.10(a). Note that Mahamud's algorithm successfully segments the boundary contours in the *Double Pears, Coins*, and *Fruit Group 1* image. Unfortunately, it still loses many desired edges. However, the

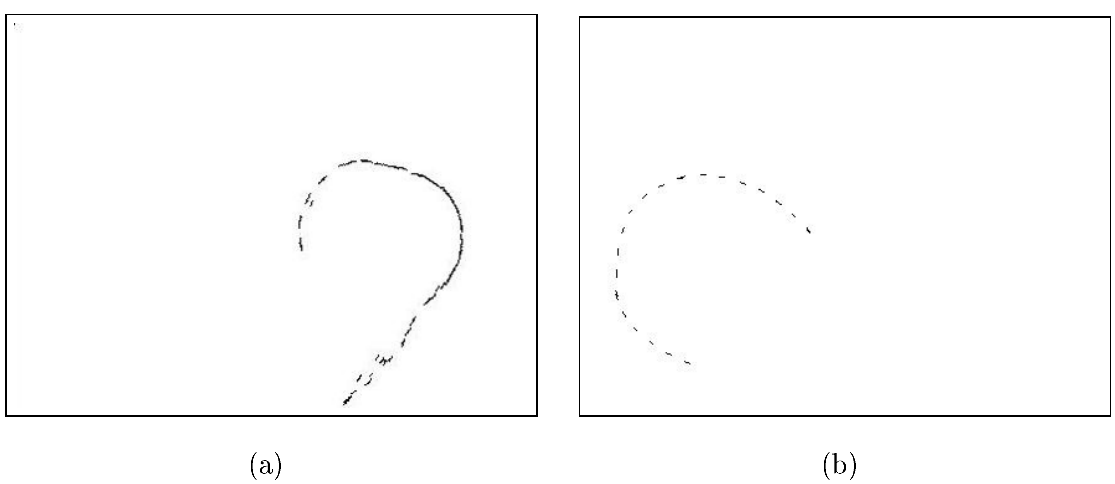
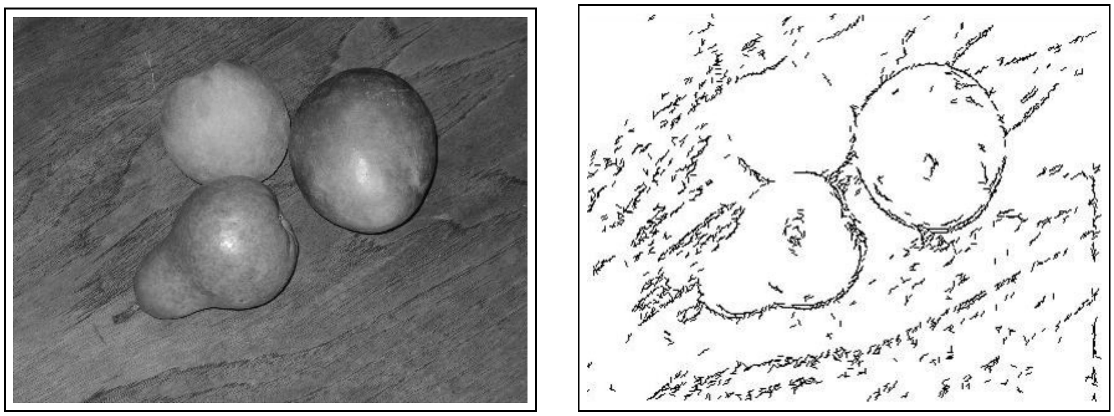


Figure 4.7: The third object isolated from *Fruit Group 2*. (a) Using the improved algorithm. (b) Using the original algorithm.

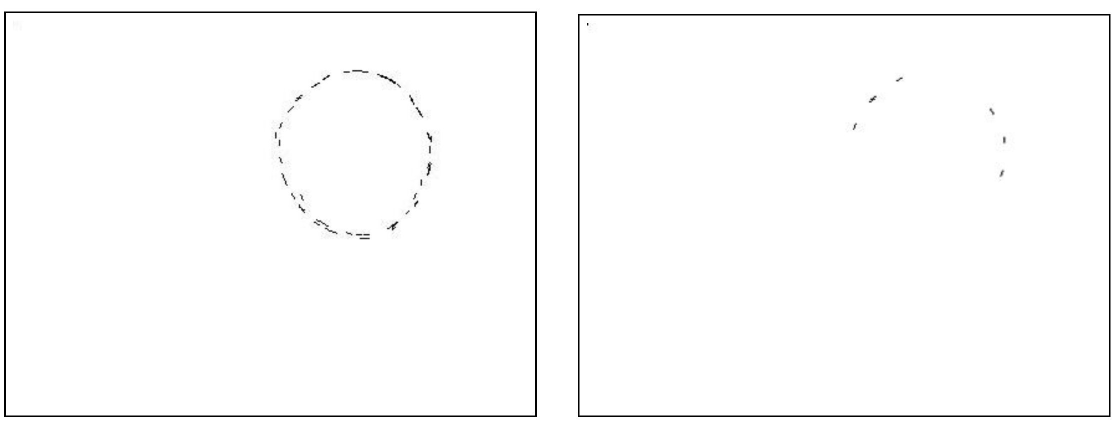


(a) (b)

Figure 4.8: *Fruit Group 3.* (a) Original image. (b) Edge input obtained by Canny detector.

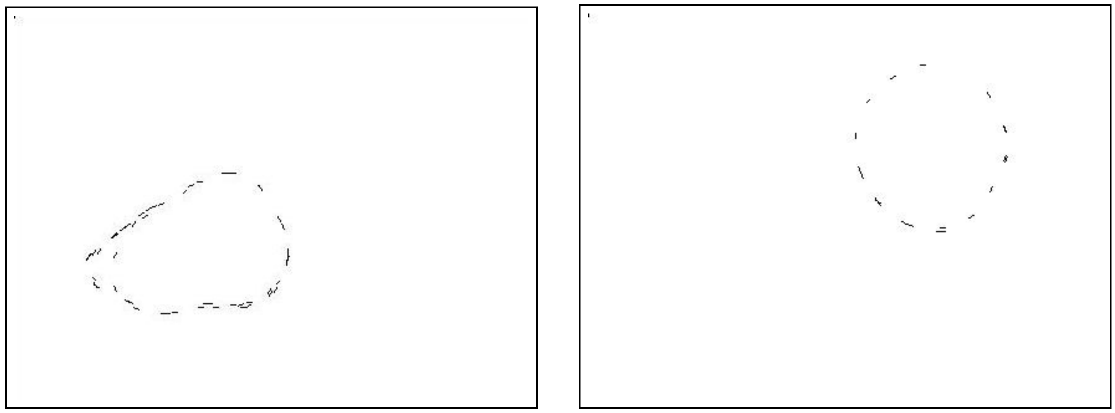
number of omitted edges is not enough to affect the objects' shapes.

It should be pointed out that in the whole experiment we used the special reversal



(a) (b)

Figure 4.9: The first object isolated from *Fruit Group 3.* (a) Using the improved algorithm. (b) Using the original algorithm.



(a) (b)

Figure 4.10: The second object isolated from *Fruit Group 3.* (a) Using the improved algorithm. (b) Using the original algorithm.

symmetry property of the P matrix. This validates the theorems we derived from the experimental point of view. Finally, we note that in the experiment shown for Mahamud's algorithm, we used our corrected definition of set intersection, instead of that used by Mahamud.

Chapter 5

Future Work

The three parameters τ, T and γ are important and they affect the segmentation solution directly. Among them, the half-life, τ, models the principle of proximity, the variance of the directional change of the particle, T, models the principle of smooth-continuation, and the speed, γ, determines the effective scale. We have succeeded in finding γ automatically. However, finding suitable values for these parameters automatically is a challenging problem, especially when efficiency is a consideration. For example, it is hard to find three parameters which yield both good solutions to the segmentation problem and also permit fast convergence for the eigenvalue problem. We need to study how particular combinations of two or three parameters values affect the segmentation.

Our algorithm for the segmentation problem is still too slow.[1] Improving efficiency is therefore an important issue. It is natural to consider parallel computation techniques and apply them to our problem. We are now in a position to develop parallel network algorithms for the segmentation problem.

[1]The total running time (including P matrix creation time) for *Fruit Group* 1 is about 35 seconds. Therefore, the average time per object is about 7 seconds.

We also would like to integrate our results into engineering applications. We can apply them to visual feedback control problems. Based on the object boundary we extract, we would like to control a robot so that it picks the desired object in the scene even when there are multiple objects.

Chapter 6

Conclusions

The purpose of segmentation is to partition a set of edges into subsets so that the edges in different subsets bound different objects in the scene. Thornber and Williams[16]'s analytic expression which characterizes the probability distribution of boundary-completion shapes provides the basic foundation for solving the segmentation problem. Mahamud *et al.*[9] developed a method to extract out salient contours bounding unknown objects from real images using a saliency measure based on the global property of contour closure for the first time. In this thesis, we analyzed and proved the properties of the affinity matrix of Williams and Thornber. This was significant for improving computational efficiency. We then described an improved algorithm. We integrated an optimization method into the segmentation algorithm to find one of the parameters in Thornber and William[16]'s expression. In the improved algorithm, we used the magnitude of the k-largest (usually $k = 2$ or 3) as a fixed threshold to isolate the most salient edges instead of the biggest jump as was done in Mahamud *et al.*[9]. In this way, we addressed the problem of omitted edges. The experimental results show that this approach improved the segmentation. In particular, it eliminated the interference with the next contour which results from incomplete segmentation. In addition, we identified and addressed an issue with the simple set intersection method used in Mahamud's algorithm.

Appendices

Appendix A

A Fundamental Expression Used in This Thesis

In this appendix we will give an analytic expression characterizing the probability distribution of boundary-completion shapes derived by Thornber and Williams[16]. We define the affinity, P_{ji}, between two directed edges, i and j, to be

$$P(j|i) = \int_0^\infty P(j|i;t)\,dt \;\approx\; FP(j|i;t_{opt})$$

where $P(j|i;t)$ is the probability that a particle which begins its stochastic motion at (x_i, x_i, θ_i) at time *0* will be at (x_j, y_j, θ_j) at time t. This probability is defined to be the sum over the probabilities of all paths that a particle can take between the two edges. Williams and Thornber[18] approximate the above integral analytically. The approximation is the product of P evaluated at the time at which the integral is maximized, i.e.: t_{opt}, and a weighting factor, F. The expression for P at time t is

$$P(j|i;t) = \frac{3\ \exp[-\frac{6}{Tt^3}(at^2 - bt + c)]\ \exp(-\frac{t}{\tau})}{\sqrt{\frac{\pi^3 T^3 t^7}{2}}}$$

where

$$
\begin{aligned}
a &= \frac{2+\cos(\theta_j-\theta_i)}{3} \\
b &= \frac{x_{ji}(\cos\theta_j+\cos\theta_i)+y_{ji}(\sin\theta_j+\sin\theta_i)}{\gamma} \\
c &= \frac{(x_{ji}^2+y_{ji}^2)}{\gamma^2}
\end{aligned}
$$

for $x_{ji} = x_j - x_i$ and $y_{ji} = y_j - y_i$. The distribution of shapes is determined by the half-life, τ, the variance, T, and the speed of the particle, γ. The expression for P should be evaluated at $t = t_{opt}$, where t_{opt} is real, positive, and satisfies the cubic equation

$$-\frac{7t^3}{4} + \frac{3(at^2 - 2bt + 3c)}{T} = 0.$$

If more than one real, positive root exists, then the root maximizing $P(j|i;t)$ is chosen. Finally, the extra factor F is

$$F = \sqrt{\frac{2\pi t_{opt}^5}{\frac{12(3c-bt_{opt})}{T} + \frac{7t_{opt}^3}{2}}}.$$

Appendix B

Proving Some Other Properties of the Affinity Matrix

Using the method described in Lemma 2, we can obtain the affinity matrix, $\mathbf{P}$, by

$$\mathbf{P} = \begin{bmatrix} \{P_{ij}\} & \{P_{i\bar{j}}\} \\ \{P_{\bar{i}j}\} & \{P_{\bar{i}\bar{j}}\} \end{bmatrix}$$

where

$$\{P_{ij}\} = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N1} & P_{N2} & \cdots & P_{NN} \end{bmatrix}$$

$$\{P_{i\bar{j}}\} = \begin{bmatrix} P_{1\bar{1}} & P_{1\bar{2}} & \cdots & P_{1\bar{N}} \\ P_{2\bar{1}} & P_{2\bar{2}} & \cdots & P_{2\bar{N}} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N\bar{1}} & P_{N\bar{2}} & \cdots & P_{N\bar{N}} \end{bmatrix}$$

$$\{P_{\bar{i}j}\} = \begin{bmatrix} P_{\bar{1}1} & P_{\bar{1}2} & \cdots & P_{\bar{1}N} \\ P_{\bar{2}1} & P_{\bar{2}2} & \cdots & P_{\bar{2}N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{\bar{N}1} & P_{\bar{N}2} & \cdots & P_{\bar{N}N} \end{bmatrix}$$

$$\{P_{\bar{i}\bar{j}}\} = \begin{bmatrix} P_{\bar{1}\bar{1}} & P_{\bar{1}\bar{2}} & \cdots & P_{\bar{1}\bar{N}} \\ P_{\bar{2}\bar{1}} & P_{\bar{2}\bar{2}} & \cdots & P_{\bar{2}\bar{N}} \\ \vdots & \vdots & \ddots & \vdots \\ P_{\bar{N}\bar{1}} & P_{\bar{N}\bar{2}} & \cdots & P_{\bar{N}\bar{N}} \end{bmatrix}.$$

We claim that the submatrices, $\{P_{i\bar{j}}\}$, and, $\{P_{\bar{i}j}\}$, are symmetric, that the elements in the upper triangle of $\{P_{ij}\}$ are equal to those in the lower triangle of $\{P_{\bar{i}\bar{j}}\}$, and that the elements in the lower triangle of $\{P_{ij}\}$ are equal to those in the upper triangle of $\{P_{\bar{i}\bar{j}}\}$, *i.e.* that $\{P_{ij}\} = \{P_{\bar{i}\bar{j}}\}^T$.

Proof. From the Lemma 1 we know that $P_{ij} = P_{\bar{j}\bar{i}}$. Therefore,

$$\{P_{ij}\} = \{P_{\bar{i}\bar{j}}\}^T.$$

Now we prove that the submatrix, $\{P_{\bar{i}j}\}$, is symmetric. In equation $P_{ij} = P_{\bar{j}\bar{i}}$, if we substitute $\bar{\imath}$ for i, we have $P_{\bar{i}j} = P_{\bar{j}i}$. We will use this relation in the following proof. The submatrix, $\{P_{\bar{i}j}\}$, is located in the lower-left quadrant of $\mathbf{P}$. Imagine that we move this submatrix to the upper-left quadrant to overwrite the original upper-left one. This movement cannot change the symmetric property of the submatrix $\{P_{\bar{i}j}\}$ because we do not change the values of the elements of the submatrix. Recall that we arrange the set of input edges, S, so that the order of the first half of the $2N$ elements is the same as the input order (i.e., edge *i's*) and the second N of the *2N* elements are edges with opposite direction to each element in the first half (i.e., edge $\bar{\imath}$*'s*). Therefore the index $\bar{\imath} = i + N$, and the index $\bar{\jmath} = j + N$. From $P_{\bar{i}j} = P_{\bar{j}i}$, we have $P_{(i+N)j} = P_{(j+N)i}$. If we perform the movement just described, we have

$P_{(i+N-N)j} = P_{(j+N-N)i}$, which is $P_{ij} = P_{ji}$. Therefore, the shifted submatrix is symmetric. It follows that the original submatrix is symmetric. Similarly, we can show that $\{P_{i\bar{j}}\} = \{P_{i\bar{j}}\}^T$.

References

[1] Bennett, D., *Visual C++ 5 Developer's Guide*, Sams Publishing, 1997.

[2] Brent, R.P., *Algorithms for Minimization without Derivatives*, Prentice-Hall, 1973.

[3] Chatelin, F., *Eigenvalues of Matrices*, Chichester, New York, 1993.

[4] Elder, J.H. and Zucker, S.W., A Measure of Closure, *Vision Research*, vol. 34, no. 24, pp. 3361-3370, 1994.

[5] Elder, J.H. and Zucker, S.W., Computing Contour Closure, *ECCV'96*, Cambridge, UK. Vol. 1, p14-18, 1996.

[6] Goossens, M., Mittelbach, F. and Samarin, A., *The Latex Companion*, Addison-Wesley, 1993.

[7] Lau, H.T., *A Numerical Library in C for Scientists and Engineers*, CRC Press, 1995.

[8] Mahamud, S., Thornber, K.K. and Williams, L.R., Extracting Multiple Salient Closed Contours from Real Images, *NEC Tech. Report* 98-120.

[9] Mahamud, S., Thornber, K.K., Williams, L.R., Segmentation of Salient Closed Contours from Real Images, *Proc. of the 7th Intl. Conf. on Computer Vision (ICCV '99)*, Corfu, Greece, 1999.

[10] Mumford, D., Elastica and Computer Vision, *Algebraic Geometry and Its Application*, Chandrajit Bajaj (ed.), Springer-Verlag, New York, 1994.

[11] Musser, D.R., *STL tutorial and reference guide : C++ programming with the standard template library*, Addison-Wesley, 2001.

[12] Parker, J.R., *Algorithms for Image Processing and Computer Vision*,Wiley Computer Publishing, John Wiley & Sons, Inc., 1997.

[13] Perona, P. and Freeman, W., A Factorization Approach to Grouping, *Proc. 5th European Conference of Computer Vision (ECCV98)*, Freiburg, Germany, Pages 655-670, 1998.

[14] Shashua, A. and Ullman, S., Structural Saliency: The Detection of Globally Salient Structures Using A Locally Connected Network, *ICCV*, FL, p321-328, 1988.

[15] Shi, J. and Malik, J., Normalized Cuts and Image Segmentation, *IEEE Transactions on Pattern Analysis and Machine Intelligence(PAMI)*, 2000.

[16] Thornber, K.K. and Williams, L.R., Analytic Solution of Stochastic Completion Fields, *Biological Cybernetics*, 1996.

[17] Williams, L.R., and Jacobs, D.W., Stochastic Completion Fields: A Neural Model of Illusory Contour Shape and Salience, *Neural Computation* 9, 837-858, 1997.

[18] Williams, L.R. and Thornber, K.K, A Comparison of Measures for Detecting Natural Shapes in Cluttered Backgrounds, *Inter. J. Computer Vision* 34(2/3), 81-96, 2000.

[19] Williams, L.R. and Thornber, K.K, Orientation, Scale, and Discontinutiy as Emergent Properties of Illusory Contour Shape, *UNM Tech. Report(TR-CS-2000-39)*, 2000.

[20] Yu, S. and Shi, J., Segmentation with Directed Relationships, *The Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition(EMMCVPR'01)*, INRIASophia-Antipolis, France, September 3 - 5, 2001.