View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by DigitalCommons@USU

Utah State University

DigitalCommons@USU

All Graduate Theses and Dissertations

Graduate Studies

5-2009

A Computational Geometry Approach to Digital Image Contour Extraction

Pedro J. Tejada
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/etd

 Part of the Computer Sciences Commons

Recommended Citation

Tejada, Pedro J., "A Computational Geometry Approach to Digital Image Contour Extraction" (2009). *All Graduate Theses and Dissertations*. 422.
https://digitalcommons.usu.edu/etd/422

This Thesis is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Theses and Dissertations by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

A COMPUTATIONAL GEOMETRY APPROACH TO DIGITAL IMAGE CONTOUR EXTRACTION

by

Pedro J. Tejada

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Minghui Jiang Major Professor	Dr. Xiaojun Qi Committee Member
Dr. Nicholas Flann Committee Member	Dr. Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2009

Copyright © Pedro J. Tejada 2009

All Rights Reserved

Abstract

A Computational Geometry Approach to

Digital Image Contour Extraction

by

Pedro J. Tejada, Master of Science

Utah State University, 2009

Major Professor: Dr. Minghui Jiang
Department: Computer Science

We present a method for extracting contours from digital images, using techniques from computational geometry. Our approach is different from traditional pixel-based methods in image processing. Instead of working directly with pixels, we extract a set of oriented feature points from the input digital images, then apply classical geometric techniques, such as clustering, linking, and simplification, to find contours among these points. Experiments on synthetic and natural images show that our method can effectively extract contours, even from images with considerable noise; moreover, the extracted contours have a very compact representation.

(176 pages)

Acknowledgments

Many thanks to my advisor, Dr. Minghui Jiang, for his encouragement and support, and for getting me interested in the field of computational geometry. His concern for me as a student and his advice have been a great help. He introduced me to the practice of doing research, and his many comments have helped me become a better researcher.

I also thank Dr. Xiaojun Qi and Dr. Nicholas Flann for their helpful comments on the drafts of this thesis. Some of their comments helped improve this thesis, and some will help improve related future work. Thanks also go to Joel Gillespie for his help with LaTeX, in particular for his help in preparing the appendices, and a thank you goes to Myra Cook for her help regarding the formatting of the whole document.

Finally, I thank my friends and family for their encouragement. I especially thank my parents, who have always been interested in my education.

This work was supported in part by NSF grant DBI-0743670 and an ADVANCE grant from Utah State University.

Pedro J. Tejada

Contents

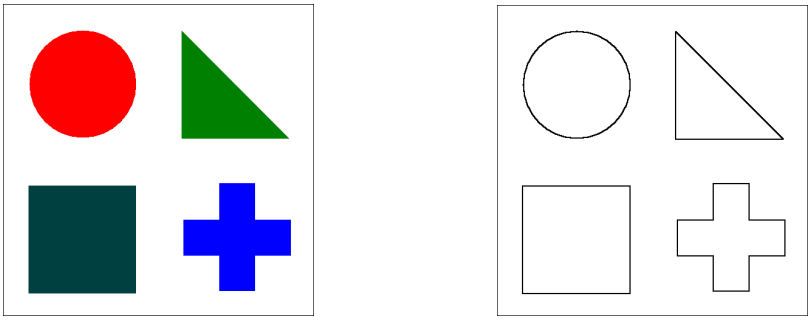
List of Tables

List of Figures

Chapter 1

Introduction

Contours[1] are the boundary lines of geometric shapes within digital images; see Fig. 1.1. Since the identification of contours is crucial for analyzing the contents of an image, contour extraction is one of the most important problems in computer vision and pattern recognition [12, p. 1135]. Once object contours have been extracted, several shape features that are useful for identifying and classifying objects can be determined. These features include perimeter length, irregularity, width, height, aspect ratio, and area [34, p. 262]. However, this problem is especially difficult for images with complex shapes and with noise.



(a) (b)

Fig. 1.1: (a) A digital image of shapes. (b) Contours.

We present a method for automatically extracting contours from digital images using techniques from computational geometry. Our aim is to show how a combination of simple geometric algorithms can be used effectively to extract contours even from images with a moderate amount of noise.

[1] We use the term contour as used in image processing to refer to lines and object boundaries. In mathematics, a contour line of a function of two variables is a curve along which the function has a constant value. In cartography, a contour line joins points of equal elevation (height) above a given level, such as mean sea level. For detailed information see `http://en.wikipedia.org/wiki/Contour_line`.

1.1 Standard Definitions

A *digital image* is a discrete approximation of an image obtained by *sampling* points with discrete coordinates and *quantizing* the values of each sample. It is formed by a finite number of sample elements equally spaced over a square grid with a rectangular shape. Each element is called a *pixel* and has an *intensity* value. The rows and columns of elements determine the *spatial* coordinates (x, y) of the pixel; and the intensity determines its gray-scale or color value. In a *gray-scale* image, all pixels have shades of gray ranging from black to white. In the case of *color* images, the intensity determines the color of each pixel according to some color model, such as RGB[2]. A digital image is stored as a two-dimensional array (usually of integers), wherein each element and its value correspond to a pixel and its intensity, respectively. When a digital image is displayed, each pixel is represented by a small square with a unique color determined by its intensity. See Fig. 1.2 for examples of continuous and digital images.

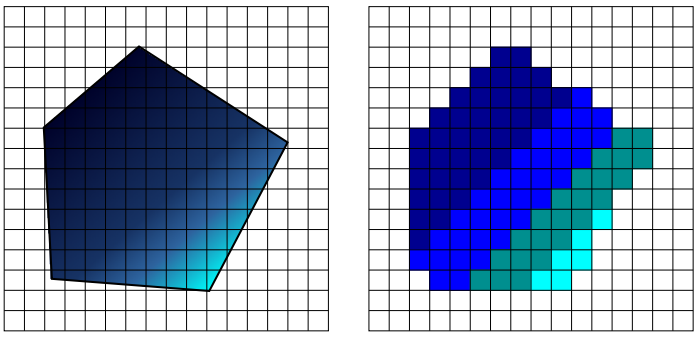


Fig. 1.2: Continuous image (left) and sampled digital image (right).

Contours are lines, straight or curved, that define forms or shapes. They can be open lines, or they can be closed boundaries. Contours can be represented by pixels, for example, with black pixels over a white background or vice versa. However, even when they can visually represent the contours of an image, these pixels alone do not provide a complete characterization of the contours. Contours are lines, not independent pixels; so in order to have contours defined by pixels, more information is needed. A sequence of consecutive

[2]In the RGB color model each color is specified by three components: red, green, and blue

adjacent pixels is a possible representation. However, this kind of representation has some disadvantages: it requires a lot of space because many pixels are needed; it is not scale independent since the number of pixels required changes with the size; and it is discrete, so lines are not generally smooth unless numerous pixels are used. Given these drawbacks, a geometric representation provides a better way to represent contours, for example, with line segments defined by pairs of points. This much simpler representation does not have the problems associated with the discrete counterpart. See Fig. 1.3 for examples of pixel-based and geometric-based contour representations.

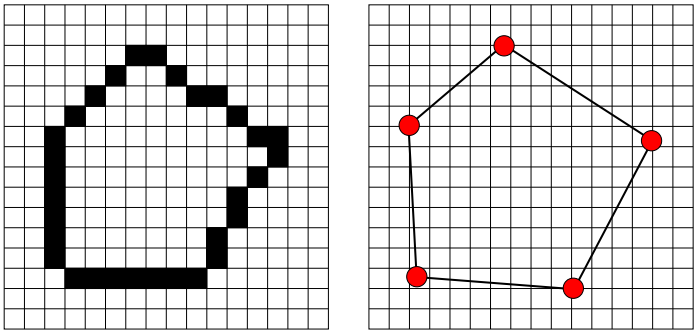


Fig. 1.3: Pixel-based contour (left) and geometric-based contour (right).

1.2 Previous Work

Traditional methods for finding contours can be classified by scope depending on whether they do local, regional, or global processing [15, pp. 725–738]. Local methods analyze a small neighborhood associated with every pixel and link adjacent pixels if they satisfy some criteria. Regional methods use different techniques to connect pixels which are previously known to be part of the same region or contour. In such cases, geometric algorithms, such as polygonal fitting, can be used to efficiently find approximations of contours; however, the knowledge required to apply such algorithms is not always available, so they are not generally applicable. Global methods, such as the Hough transform, do not rely on any kind of prior knowledge, and try to find sets of pixels which lie on curves of specific shapes. These three methods all present some drawbacks: local methods ignore valuable

global information about the geometric proximity of pixels, since they only look at a very small neighborhood; regional methods require prior knowledge about which pixels are part of which contour; and global methods such as the Hough transform can only be used to find certain types of shapes. Our method exploits the global information about the geometric proximity of pixels, requires no prior knowledge about the regional membership of pixels, and is not restricted to any particular shapes.

Another possible classification for contour extraction algorithms is by the way they work. The extraction of line segments from images is an important problem related to contour extraction: straight lines are a subset of all possible contours and since any curve can be approximated by small segments some contour extraction algorithms are actually line extraction algorithms. According to [16], several models have been reported in the literature for the extraction of line segments from images, and these are broadly classified into four categories: statistical based, gradient based, pixel connectivity-edge linking based, and Hough transform based. The linking algorithms work by connecting edge pixels based on proximity and orientation, and dividing the contours into straight line segments. Nevatia and Babu's line detector [28] is a classic example of these methods. Our method uses a linking algorithm, but instead of connecting edge pixels, it uses computational geometry to connect oriented points.

Many algorithms related to image processing and specifically contour extraction are related to the field of discrete or digital geometry [1, 24]. This field has become more important in the last decade, but it has evolved independently of the field of computational geometry. However, researchers are now using results from computational geometry to solve problems in discrete geometry, and the integration of both fields seems promising [1]. Our method for extracting contours is a case wherein computational geometry is used to solve a problem in discrete geometry: extracting contours from points in a discrete grid. We extract contours from digital images by moving to a continuous domain and using well established results from computational geometry. The result is not discrete, but the extracted contours could be rasterized to discretize them if it is necessary.

1.3 Our Method

Our method consists of two stages: a pre-processing stage that extracts a set of oriented points from the input image, and a second stage that finds the contours among the oriented points using geometric algorithms. The second stage is the most important and has three steps: (1) points are first filtered by a clustering technique; (2) then points are linked, based on proximity and orientation, into paths representing the contours; (3) and finally paths are simplified by reducing the number of points they have. See Fig. 1.4.

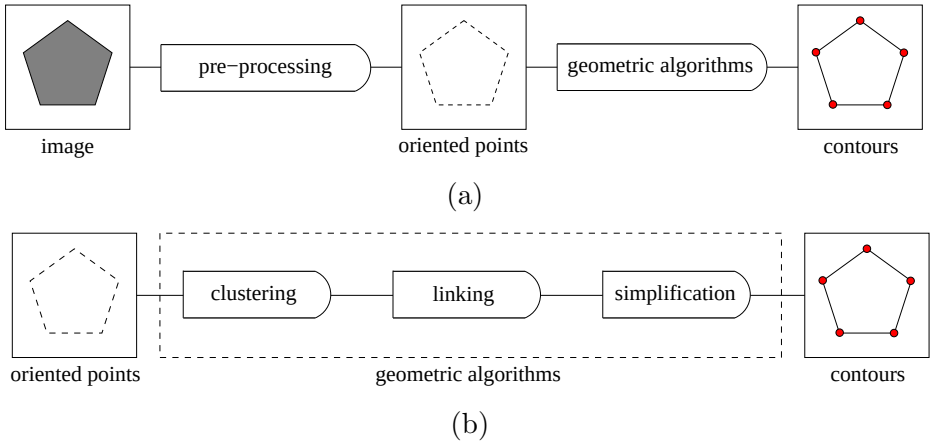


Fig. 1.4: Overview of our method. (a) Two stages. (b) Geometric algorithms for the second stage.

Chapter 2

Input Conversion

We first transform the problem of finding contours into a geometric problem. The goal of the pre-processing stage is to find a set of points that are possibly part of the contours. To do this, we use an edge detector to extract edge pixels from the image, and then we convert them into oriented points. Finding the contours is then a matter of connecting those points into meaningful boundaries.

At the pre-processing stage, a Sobel edge detector [15] is used to determine possible contour pixels, which are then transformed into oriented points. The edge detector outputs a set of *edge pixels* wherein the intensity of the image changes abruptly. Each edge pixel has a *magnitude* indicating how good or strong the edge at the pixel location is, and a *direction* indicated by an angle. Next, each pixel is transformed into an *oriented* point p_i located at the center (x_i, y_i) of the pixel, with its orientation α_i given by the edge direction, and a weight w_i initialized with the edge magnitude; see Fig. 2.1.

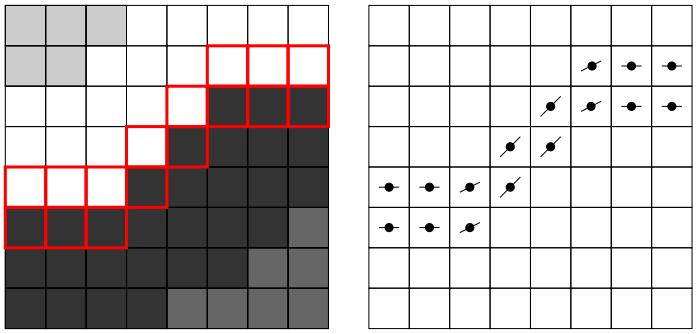


Fig. 2.1: Section of digital image with strong edge pixels highlighted (left) and corresponding oriented points (right).

There are many edge detectors that can be used to extract edge pixels; some examples include Sobel [8, 15], Laplacian [15], and Canny [4]. We use a Sobel edge detector, but others could be used as well. Since we want to focus on the geometric algorithms of the

second stage, and they are independent of the method used by the pre-processing stage, this choice is not so relevant. However, it is important that the method used return not only the edge pixels, but also their gradient, which is used by later parts of the algorithm.

When edge pixels are transformed into points, the spatial coordinates and the orientation angles are normalized to $[0, 1]$. The image rectangle is rescaled to fit inside a unit square, and the angle $\alpha = \pi$ is mapped to 1[1]. Since orientations can be specified by angles between 0 and π, if any orientation α is greater than π, we use $\alpha \bmod \pi$ instead.

We also apply some other transformations to the coordinates of the points. Digital images are usually represented with the x and y coordinates increasing right and down from the top-left corner, so we invert the y–axis so that the origin is at $(0, 0)$. We also center the rectangle occupied by the original image in the unit square so that it is centered when visualized. If I_w and I_h are the width and height of the image after being scaled down to fit inside the unit square, and if $I_h < I_w = 1$, we translate all the points a distance of $1 - \frac{I_h}{2}$ in the y direction. See Fig. 2.2.

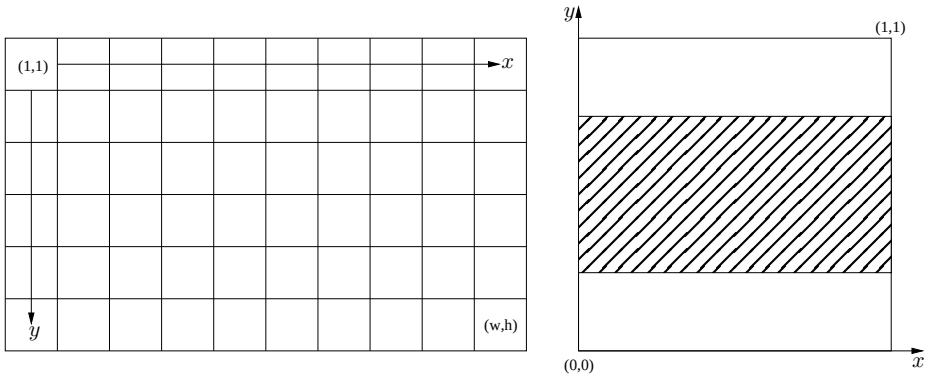


Fig. 2.2: Points coordinate mapping. The rectangle on the left shows a digital image: the coordinates start at the top-left corner and increase to the right and down. The right square shows how points are mapped to a unit square: the image is rescaled to fit inside the square, the y-axis is inverted, and the image is centered.

[1]We still use angles in radians when describing the algorithms, instead of the normalized values between 0 and 1 used in the implementations.

Chapter 3

Point Clustering

Clustering techniques are very useful for image processing and pattern recognition. For example, clustering methods are among the most powerful approaches for image segmentation [33], which is used as a middle step for the identification and extraction of objects of interest from images. The objective of clustering analysis is to partition a set of points into groups, or clusters, that are natural according to some similarity measure [12]. Some clustering methods include hierarchical methods, graph-theoretic methods, and the well known k-means algorithm [12]. Those and other clustering algorithms frequently use computational geometry, as the similarity between clusters is usually expressed in terms of Euclidean distances between points in a feature space.

We use a clustering-based algorithm to reduce the number of points for two key reasons. First, it reduces the processing time of the following steps, and second, it improves the results of the linking step, which when the points are close together might find multiple lines where there should be a single contour. Geometric algorithms can be more time consuming than local processing pixel-based algorithms. Therefore, it is convenient to reduce the input size. When doing so, however, it is important for the new point set to resemble the initial point set. This is similar to the geometric problem of *dot map simplification* which, given a point set, tries to find a smaller set whose distribution approximates that of the original set [6]. The difference is that in our case we also require the orientations to approximate those of the original set.

3.1 Algorithm

We reduce the number of points using a simple iterative greedy algorithm that repeatedly merges the closest pair of points into a new point until the distance between the closest

pair reaches a threshold $d_{\max}$; see Fig. 3.1.

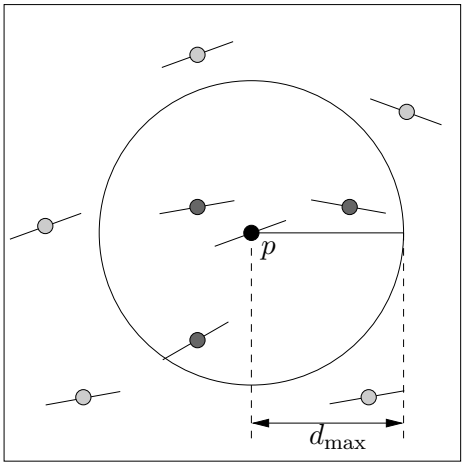


Fig. 3.1: When clustering, points are merged until the distance between the closest pair reaches a threshold $d_{\max}$.

When a pair of points is merged into one, the values for the new point are weighted averages of the values of the original points. This ensures that the distribution and orientations of the new point set approximate those of the original set; see Fig. 3.2. However, we must take special care to determine the new orientation. Merging points p_i and p_j into a new point p_k is done as follows:

$$x_k = \frac{x_i w_i + x_j w_j}{w_i + w_j}, \quad y_k = \frac{y_i w_i + y_j w_j}{w_i + w_j}, \quad \alpha_k = \frac{\alpha'_i w_i + \alpha'_j w_j}{w_i + w_j}, \quad w_k = w_i + w_j. \tag{3.1}$$

Where $\alpha'_i \in \{\alpha_i, \alpha_i + \pi\}$ and $\alpha'_j \in \{\alpha_j, \alpha_j + \pi\}$ are chosen so that the orientation of p_k is close to the orientations of both p_i and p_j: if $|\alpha_i - \alpha_j| \leq \frac{\pi}{2}$ then $\alpha'_i = \alpha_i$ and $\alpha'_j = \alpha_j$; otherwise, if $\alpha_i < \alpha_j$ then $\alpha'_i = \alpha_i + \pi$ and $\alpha'_j = \alpha_j$, and if $\alpha_j < \alpha_i$, then $\alpha'_i = \alpha_i$ and $\alpha'_j = \alpha_j + \pi$.

3.2 Implementation

Our purpose is to show that the geometric techniques work; we are not interested in speed at this point. Thus we use a simple but slow array-based implementation of the

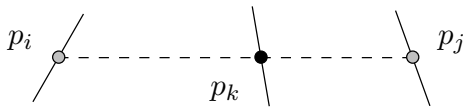


Fig. 3.2: Merging points p_i and p_j into point p_k. Both the location and orientation of the new point p_k are weighted averages of the values of the original points p_i and p_j. In this case, p_k is closer to p_j because p_j has a greater weight than p_i. After the new point is created, the merged points are removed.

clustering algorithm. However, we also describe a faster implementation in section 3.2.2. We base our analysis on the following observation:

Observation 3.1. *Let d_{min} be the closest pair distance between points in a set S, and let d_{max} be a distance such that $d_{max} \geq d_{min}$. Given point p in S, if $d_{max} = c \cdot d_{min}$ for a small constant c, then the number of points k in S with a distance to p of at most d_{max} is a constant.*

Since $d_{\min}$ is the closest pair distance, for any point p there is no other point inside the circle of radius $d_{\min}$ centered at p. Therefore, the number of points k within distance $d_{\max}$ of p is limited by the maximum number of non-overlapping circles of radius $d_{\min}$ that can fit inside a circle of radius $d_{\max} + d_{\min}$ centered at p. Based on the areas of these two circles we have that $k \leq \frac{\pi(d_{\max}+d_{\min})^2}{\pi d_{\min}^2} = \frac{\pi(c \cdot d_{\min}+d_{\min})^2}{\pi d_{\min}^2}$, and after simplification we have $k \leq (c+1)^2$. In practice however, the number is actually smaller, as packing non-overlapping circles always leaves empty space between them.

If $c = 1$ then $k \leq 2^2 = 4$, if $c = 2$ then $k \leq 3^2 = 9$, and if $c = 3$ then $k \leq 4^2 = 16$. We generally do not use $d_{\max} > 3 \cdot d_{\min}$. Therefore, the maximum number of points is always a small constant.

3.2.1 Array-based Implementation

We now explain our array-based implementation. It works as follows. First find all m candidate pairs of points separated by a distance of at most $d_{\max}$ and put them in an array. This is done by checking all pairs in $O(n^2)$ time. Next, merge pairs of points from the pairs array in $O(n + m)$ time with the following steps: (1) traverse the array to find

the closest pair in $O(m)$ time, (2) merge the pair in constant time, (3) remove from the array all pairs containing any of the two merged points in $O(m)$ time, (4) add all pairs containing the newly created point that are not separated by a distance greater than $d_{\max}$ to the candidate pairs array in $O(n)$ time by traversing the input array of points.

The total number of steps can be at most $n-1$ because at each step the number of points is reduced by one. Therefore, the total running time is $O(n^2 + nm)$, which can be $O(n^3)$ if the number of candidate pairs m is $O(n^2)$. However, Observation 3.1 implies that if $d_{\max}$ is small, m is $O(n)$, and the algorithm runs in $O(n^2)$ time.

Proposition 3.2. *Given a set of n points with closest pair distance d_{min}, the array-based implementation of the clustering algorithm runs in $O(n^3)$ time. However, if $d_{max} = c \cdot d_{min}$ for a small constant c, then it runs in $O(n^2)$ time.*

3.2.2 Improved Running Time

The array-based implementation must traverse either the entire pairs array to find the closest pair or the entire points array to find all candidate pairs for a given point. However, finding the closest pair or all candidate pairs for a given point could be done faster by using a variety of range searching techniques, because only close points are considered.

We now explain how to improve the running time. The total number of steps of our algorithm is at most $n-1$ because at each step the number of points is reduced by one. Therefore, the algorithm can easily be implemented to run in $O(n^2 \log n)$ time by finding the closest pair in $O(n \log n)$ time [2, 21] at every step, or in $O(\log n)$ time by using a data structure that can maintain the closest pair in $O(\log n)$ time per insertion and deletion [3].

Proposition 3.3. *The clustering algorithm can be implemented to run in $O(n^2 \log n)$ time, or in $O(n \log n)$ time using more complex data structures.*

Chapter 4

Point Linking

After an image has been segmented into a set of regions or edge pixels, the next step is to find the contours determined by those regions or pixels. This is usually done by a simple *contour tracing* algorithm, such as Moore's algorithm [27], that traces the boundaries by starting from a known contour pixel and repeatedly moving to adjacent contour pixels until a stopping condition is met (usually returning to the original pixel). Contours obtained in this way are then stored as a sequence of pixels encoded as a chain-code [10, 11]. However, these algorithms and encodings are very simple; the algorithms only work with very clean boundaries, and the encodings take a lot of space. Another possibility is to use edge linking algorithms, which link edge pixels if they are within a small neighborhood and have a similar magnitude or direction [28–30, 34].

The linking step is the one that actually finds the contours. Prior to this step, we have a set of oriented points that have to be connected in order to find the contours. The objective is to find a set of paths representing the contours. These *paths* are sequences of line segments between the given points, or polylines. Paths can be represented by other types of functions, such as spline curves, but we use polylines because they are efficient for computations and because any curve can still be approximated by straight line segments.

Similar to edge linking algorithms, our algorithm links points based on proximity and orientation. However, our linking algorithm is conceptually simpler. A typical pixel-based contour extraction algorithm might: (1) extract edge pixels with an edge detector, (2) fill gaps between edges, (3) connect pixels (contour tracing), and optionally (4) approximate the contours with line segments. In our method, the edge detection is the same, but steps 2 to 4 are all done by the linking algorithm. Gaps are filled by linking points, and there is no need for a line approximation step, as points are linked by line segments. Also, pixel

based algorithms usually have to consider multiple special cases. In contrast, our linking algorithm works by following very simple rules. We now describe the algorithm.

4.1 Algorithm

Our algorithm is in spirit similar to Prim's algorithm for minimum spanning tree (MST) [13, pp. 366–369] in the sense that it extends a path (grows a cluster) until it cannot be extended anymore. However, we do not find a MST because of some restrictions on which segments can be chosen, and because we have to continue to find multiple paths. Generating a path starts from a single segment and then greedily extends the path in both directions.

The initial segment of the new path is determined by a pair of isolated points (p_i, p_j) within the threshold distance $d_{\max}$, such that the weight $w(p_i, p_j)$ (to be defined) is the maximum. Next, the path is extended at both ends by repeatedly adding points until there are no more candidates or the added point is already part of some path before being added; see Fig. 4.2. When extending a path $P = (\ldots, p_{i-2}, p_{i-1}, p_i)$ from the end point p_i, the algorithm takes the point p_x with the best weight and that satisfies the following conditions: (1) the distance from p_x to p_i is at most $d_{\max}$, and (2) the turn angle from $p_{i-1}p_i$ to p_ip_x is at most $\pi/2$. Fig. 4.1 provides an example of the point with the best weight being selected from among p_a, p_b, and p_c, since p_d and p_e do not satisfy the given conditions.

The weight $w(p_i, p_j)$ of a pair of points (p_i, p_j) is determined by the distance between them $|p_ip_j|$ and the difference between their orientations and the orientation of the segment p_ip_j. Therefore, the weight depends on two parameters $d_{\max}$ and $\alpha_{\max}$. We use a function that decreases when the distance or the differences between the orientations increase, and that has a minimum value of 0 when the distance or the differences between the orientations are greater than or equal to $d_{\max}$ or $\alpha_{\max}$. Using this function, we link pairs of points if their weight is greater than 0.

We next describe the weight function. Let a_{ij}^i be the difference between the orientation of point p_i and segment p_ip_j; and let a_{ij}^j be the difference between the orientation of point p_j and segment p_ip_j. We determine a weight for the distance between the points $|p_ip_j|$ and

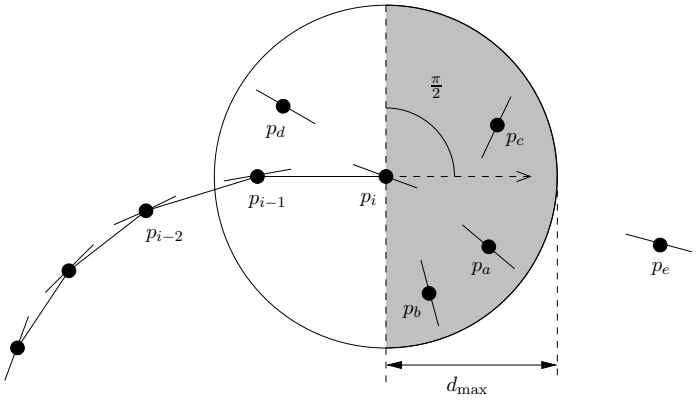


Fig. 4.1: Extending the path $P = (\ldots, p_{i-2}, p_{i-1}, p_i)$ at the end point p_i. Only points in the gray area can be linked. The maximum distance allowed to link points is $d_{\max}$.

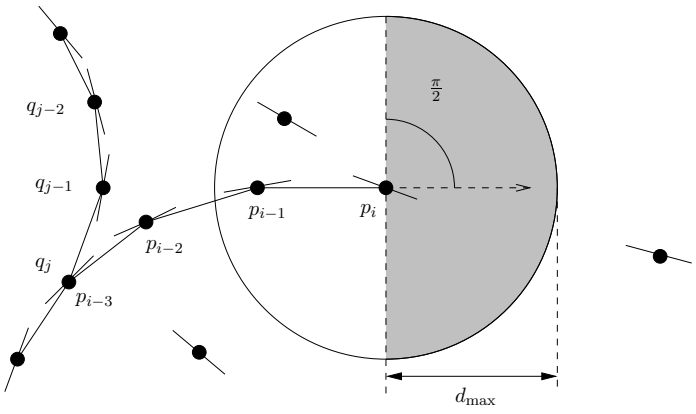


Fig. 4.2: Path $P = (\ldots, p_{i-2}, p_{i-1}, p_i)$ cannot be extended at the end point p_i since there is no point to add: the maximum distance allowed to link points is $d_{\max}$ and only points in the gray area can be linked. Path $Q = (\ldots, q_{j-2}, q_{j-1}, q_j)$ cannot be extended at the end point q_j since $q_j = p_{i-3}$ is already part of path P.

for each of the orientation differences a_{ij}^i and a_{ij}^j. These weights are higher when the values are smaller, and therefore, we take the minimum of the three values as the weight for the pair of points. Intuitively, this is the value for the worst of the three: the distance and each of the orientation differences.

Fig. 4.3 illustrates the weight function. The weight for each part, distance $w_d(p_i, p_j)$ and orientation differences $w_{\alpha_i}(p_i, p_j)$ and $w_{\alpha_j}(p_i, p_j)$, is determined by a linear function with a value of 1 when the value is optimal (like zero distance), and a value of 0 when the value is maximal (like $d_{\max}$ or $\alpha_{\max}$). For any value greater than the maximum, the function is 0. The equation for the function is as follows:

$$w(p_i, p_j) = \min\{w_d(p_i, p_j), w_{\alpha_i}(p_i, p_j), w_{\alpha_j}(p_i, p_j)\}, \tag{4.1}$$

where

$$w_d(p_i, p_j) = \begin{cases} 1 - \frac{|p_i p_j|}{d_{\max}} & \text{if } |p_i p_j| < d_{\max} \\ 0 & \text{otherwise,} \end{cases} \tag{4.2}$$

$$w_{\alpha_x}(p_i, p_j) = \begin{cases} 1 - \frac{\alpha_{ij}^x}{\alpha_{\max}} & \text{if } \alpha_{ij}^x < \alpha_{\max} \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

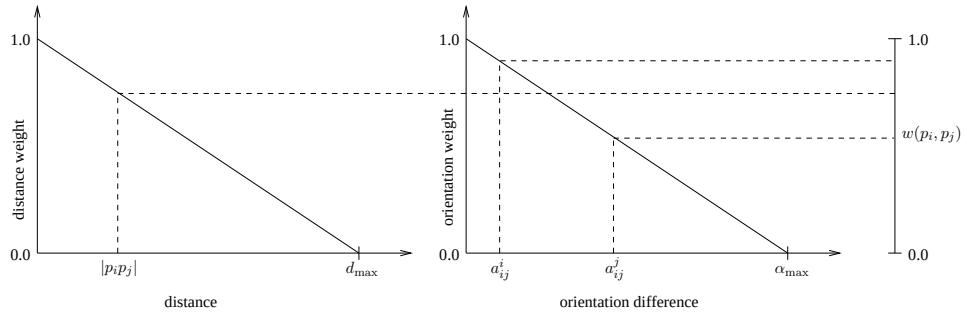


Fig. 4.3: Link weight function for the pair of points (p_i, p_j). A weight is assigned to the distance between the points $|p_i p_j|$, and to each of the differences between the orientation of each point and the orientation of the segment $p_i p_j$, a_{ij}^i and a_{ij}^j. The weight $w(p_i, p_j)$ of the pair is the minimum of those three values.

4.2 Implementation

Again we are not concerned with speed, but with showing that the geometric techniques work. We use a simple but slow array-based implementation of the linking algorithm, but we also describe a possible faster implementation in section 4.2.2.

4.2.1 Array-based Implementation

We now explain our array-based implementation. It works as follows. Find all possible pairs that can be linked in $O(n^2)$ time, put them in an array, and sort them in $O(n^2 \log n^2)$ time. A separate array is used to keep track of the degree (number of adjacent segments) of each point. Next, the best pair is found by traversing the pairs array until a pair with both points having degree zero is identified. After that, extending the path by one point can be done in $O(n)$ time by checking all points and taking the one with the best pair weight and satisfying the link conditions.

Since there can be $O(n^2)$ segments, the total time for the algorithm could be $O(n^3)$. But if $d_{\max}$ is small, so that $d_{\max} = c \cdot d_{\min}$ for a small constant c, Observation 3.1 implies that the number of candidate segments is $O(n)$ and the algorithm runs in $O(n^2 \log n^2)$ time.

Proposition 4.1. *Given a set of n points with closest pair distance d_{min}, the array-based implementation of the linking algorithm runs in $O(n^3)$ time. However, if $d_{max} = c \cdot d_{min}$ for a small constant c, it runs in $O(n^2 \log n^2)$ time.*

4.2.2 Improved Running Time

The running time can be improved by using a range searching technique, such as bucketing or a range tree. For our choice of $d_{\max} = c \cdot d_{\min}$ for a small constant c, the number of points within distance $d_{\max}$ of a point p_i is a constant, and the total number of pairs that may be linked is $O(n)$. By using a range searching technique, all these pairs can be found in linear time and sorted in $O(n \log n)$ time. Further, finding the initial pair for a path and the best point to extend it can be done in constant time. Therefore, the algorithm can be implemented to run in $O(n \log n)$ time.

Proposition 4.2. *Given a set of n points with closest pair distance d_{min}, if $d_{max} = c \cdot d_{min}$ for a small constant c, an implementation of the linking algorithm using range searching, runs in $O(n \log n)$ time.*

Chapter 5

Path Simplification

The paths obtained by the linking step are often more complex than necessary. Thus it is desirable to simplify them by reducing their number of points. So doing also makes it possible to reduce the space required to store them, reduce small inconsistencies due to noise, and improve the efficiency of any further processing based on them. For example, many features used for pattern recognition are extracted from object boundaries represented by contours, and they can be extracted faster if the contours are simplified. This is the goal of the simplification step.

Simplification is based on the fact that many consecutive points of the same contour are either collinear or close to the same straight line. Given a path $P = (p_1, p_2, \ldots, p_n)$, it is clear that any sub-path of consecutive collinear points $(p_i, p_{i+1}, \ldots, p_{j-1}, p_j)$ can always be replaced by the segment $p_i p_j$. On the other hand, in the case of points that are not all collinear, but that are still close to a straight line, it might still be acceptable to remove the intermediate points and keep a straight line approximation. In most cases, if the simplified paths are allowed to differ slightly from the originals, it is possible to have a significant reduction of the number of points. Moreover, if the allowed difference is small, the visual difference is hardly perceivable.

Since paths are represented by polylines, the problem of simplifying paths is the same as the geometric problem of *polygonal chain approximation* or *simplification*. We first introduce some preliminaries and then explain our approach for solving it.

5.1 Preliminaries

A *polygonal chain* $P = (p_1, p_2, \ldots, p_n)$ consists of n points p_i, also called *vertices*, joined by the segments $p_i p_{i+1}$. It can be an *open* chain if $p_1 \neq p_n$, or a *closed* polygon if $p_1 = p_n$.

We call p_1 and p_n *extreme points*, and $p_2, \ldots, p_{n-1}$ *intermediate points*, even if the chain is closed. For our purposes we assume that any polygonal chain must have at least two vertices and therefore at least one segment.

The polygonal chain approximation problem is defined as follows [12]: Given a polygonal chain $P = (p_1, p_2, \ldots, p_n)$, find another chain $Q = (q_1, q_2, \ldots, q_m)$ such that (1) $m < n$ (ideally $m \ll n$); (2) the q_j are selected from among the p_i, with $q_1 = p_1$ and $q_m = p_n$; and (3) any segment $q_j q_{j+1}$ that replaces the sub-chain $q_j = p_r \ldots p_s = q_{j+1}$ is such that the distance $\varepsilon(r, s)$ between $q_j q_{j+1}$ and each p_k, $r \leq k \leq s$, is less than some predetermined error tolerance ε according to some error criterion. Notice that the orientations of the points used by the linking step are not relevant here. Fig. 5.1 provides an example of a simplified polygonal chain.

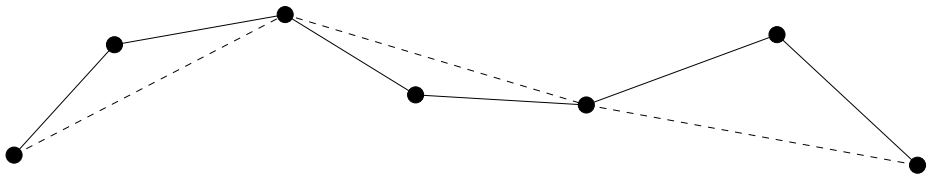


Fig. 5.1: A polygonal chain with seven vertices (solid line) and a corresponding approximation with four vertices (dashed line).

Several error criteria have been used for the approximation problem, each one with its own algorithmic issues. For each error criterion, an error $\varepsilon(q_j, q_{j+1})$ is associated with any segment $q_j q_{j+1}$ that replaces the sub-chain $q_j = p_r, \ldots, p_s = q_{j+1}$; and the error of Q is the maximum of the errors of each of its segments

$$\varepsilon(P, Q) = \max_{1 \leq j < m} \Big\{ \varepsilon(q_j, q_{j+1}) \Big\}. \tag{5.1}$$

Some example errors include: (1) the maximum of the distances from the vertices p_k, $r \leq k \leq s$, to the segment $q_j q_{j+1}$ (*segment* criterion); (2) the maximum of the distances from the vertices p_k, $r \leq k \leq s$, to the line passing through $q_j q_{j+1}$ (*parallel-strip* criterion); (3) the sum of the distances from the vertices of p_k, $r \leq k \leq s$, to the segment $q_j q_{j+1}$; and (4) the area of the closed polygon $(p_r, p_{r+1}, \ldots, p_{s-1}, p_s, p_r)$.

5.2 Algorithm

Several algorithms have been proposed for approximating polygonal chains [5, 9, 17, 20, 26, 32]. Most optimal algorithms take $\Omega(n^2)$ time to find approximations in $\mathbb{R}^2$, but some algorithms are faster for certain error criteria. Some heuristic algorithms that can achieve faster running times have also been used, and even if they are not optimal they can obtain acceptable solutions in many cases. For example, the Douglas-Puecker algorithm [31, pp. 604–606] is an output-sensitive heuristic algorithm that takes $O(mn)$ time to find an approximation with m vertices of a polygon with n vertices, but that has an expected running time of $O(n \log m)$ [17]. We propose a dynamic programming algorithm to simplify polygonal chains, based on the segment criterion. Our algorithm uses the detour to decide whether an approximation segment has an error less than the specified tolerance ε. To simplify the contours, we run this algorithm on every path and filter the output to remove *isolated points* not belonging to any path, as well as paths under a certain length that might exist due to noise in the original image. When polygonal chains are simplified, intersections can exist. We do not consider this a problem as there can be intersections even before the simplification step.

5.2.1 Using the Detour to Bound the Distance to Segment

The *detour* [7] of a chain P on the pair of points (p_i, p_j) is defined as the total length $|p_i \dots p_j|$ of the sub-chain $p_i \dots p_j$ divided by the length of the segment $p_i p_j$:

$$d(i, j) = \frac{|p_i \dots p_j|}{|p_i p_j|}.$$

Clearly $d(i, j) \geq 1$; it is equal to 1 when the segment $q_i q_j$ is an exact representation of the sub-chain $p_i \dots p_j$, and greater than 1 when it is not.

The algorithm uses the following property of the detour to determine if a segment of the approximation has an error within the desired tolerance: Given a segment $p_i p_j$ and an

error tolerance ε for the segment criterion, there is a bound

$$d_T(i,j) = \frac{\sqrt{4\varepsilon^2 + |p_i p_j|^2}}{|p_i p_j|} \tag{5.2}$$

on the detour $d(i,j)$ such that, if $d(i,j) \leq d_T(i,j)$, then $\varepsilon(i,j) \leq \varepsilon$. See Fig. 5.2.

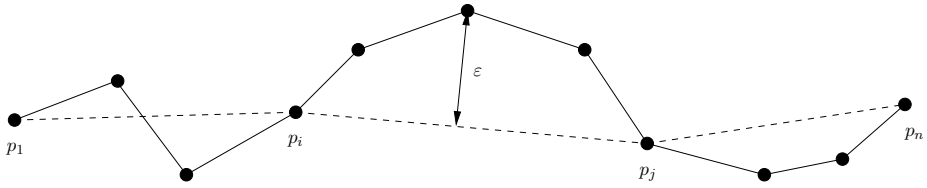


Fig. 5.2: Error under the segment criterion and detour of an approximation segment. The sub-chain $p_i \ldots p_j$ is approximated by the segment $p_i p_j$. The error, under the segment criterion, of segment $p_i p_j$ is ε. The detour for segment $p_i p_j$ is the length of the sub-chain $p_i \ldots p_j$ divided by the length of the segment $p_i p_j$.

Refer to Fig. 5.3 for a demonstration. The sub-chain $p_i \ldots p_k \ldots p_j$ (solid line) is approximated by the segment $p_i p_j$, with p_k being the farthest point of the sub-chain to the segment. Given an error tolerance ε for the segment criterion, all points of the sub-chain $p_i \ldots p_j$ must be inside the tolerance region (dashed line), which is a combination of a rectangle with sides of lengths $|p_i p_j|$ and 2ε, and two semicircles of radii ε. Given a detour, all points of the sub-chain $p_i \ldots p_j$ are inside an ellipse with foci p_i and p_j. Clearly, the maximum detour that ensures that the error is within the tolerance ε is determined by the ellipse (solid line) with foci p_i and p_j, and that is tangent to the boundary of the tolerance region. This is because if the detour is larger, then some point of the sub-chain $p_i \ldots p_j$ could be outside of the tolerance region. This ellipse is tangent to the tolerance region when the sub-chains $p_i \ldots p_k$ and $p_k \ldots p_j$ are equal to the segments $p_i p_k$ and $p_k p_j$, respectively, and $|p_i p_k| = |p_k p_j|$. Therefore, the maximum detour is

$$d_T(i,j) = \frac{2\sqrt{\varepsilon^2 + \left(\frac{|p_i p_j|}{2}\right)^2}}{|p_i p_j|} = \frac{\sqrt{4\varepsilon^2 + |p_i p_j|^2}}{|p_i p_j|}. \tag{5.3}$$

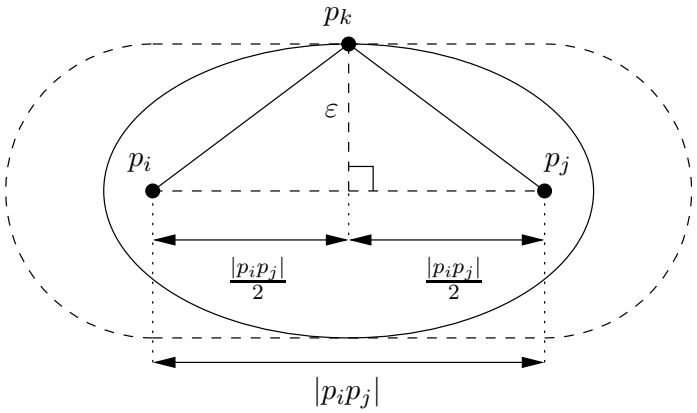


Fig. 5.3: Maximum detour for a specified error tolerance. The sub-chain $p_i \dots p_k \dots p_j$ (solid line) is approximated by the segment p_ip_j, with p_k being the farthest point of the sub-chain to the segment. Given an error tolerance ε for the segment criterion, all points of the sub-chain $p_i \dots p_j$ must be inside the tolerance region (dashed line). The maximum detour that ensures that the error is within the tolerance ε is determined by the ellipse (solid line) with foci p_i and p_j, and tangent to the boundary of the tolerance region.

5.2.2 Dynamic Programming

We now describe the dynamic programming algorithm. $K(i)$ denotes the minimum number of points of the best known approximation of the sub-chain $p_1 \dots p_i$. The dynamic programming algorithm is as follows:

Base case: For all i,

$$K(i) = i.$$

Recurrence: For all i and j such that $1 \leq j < i$ and $d(j,i) \leq d_T(j,i)$,

$$K(i) = \min\Big\{K(i),\, K(j)+1\Big\}.$$

Intuitively, for every sub-chain $p_1 \dots p_i$ the algorithm determines the minimum number of vertices required by any of the best approximations having p_j, $1 \leq j < i$, as the next to last vertex. The condition $d(j,i) \leq d_T(j,i)$ ensures that only valid segments p_jp_i, i.e., those with an error within the specified tolerance, are considered.

To recover the simplified chain, we use another table $P(i)$ to store the vertex choices for the best approximation. The choice table $P(i)$ stores the index of the next to last vertex

of the best approximation of the sub-chain $p_1, \ldots, p_j, p_i$. We initialize $P(i)$ to $i-1$, and during the computation of $K(i)$ we set $P(i)$ to j every time $K(i)$ is updated to $K(j)+1$. After the computation of the two tables $K(i)$ and $P(i)$, we recover the best approximation from $P(i)$. Start with the last vertex p_n and backtrack from $P(n)$ until the first vertex p_1 is reached. Every time $P(i)$ is visited, add vertex $k = P(i)$ to the approximation chain and move to $P(k)$.

5.3 Implementation

We now analyze the running time of the algorithm. The detour for any segment $p_i p_j$ can be computed in constant time, after linear time pre-processing. First compute the lengths $L(i)$ of all sub-chains $p_1 \ldots p_i$: $L(i) = L(i-1) + |p_{i-1}p_i|$. Then the length of the sub-chain $p_i \ldots p_j$ is $L(i) - L(j)$, and the detour can simply be computed as

$$d(i,j) = \frac{L(i) - L(j)}{|p_i p_j|}.$$

With a constant time to compute the detour, the dynamic programming algorithm clearly runs in $O(n^2)$ time, where n is the number of vertices in the original chain. After the computation of the tables, because every step of the backtracking procedure takes constant time and the approximation can have at most n vertices, recovering the best approximation takes linear time. Therefore, the overall running time of the simplification algorithm is $O(n^2)$.

After simplifying all paths, filtering the output to remove unused points takes linear time on the total number of points. Since every point is part of at most a constant number of segments, it is possible to filter the isolated points in linear time by traversing each path and marking as used the point that are visited. Removing short paths can also be done in linear time since the length of each path can be computed by traversing the path in $O(n)$ time, where n is the number of points in the path.

Our algorithm is not optimal. It does not find the minimum number of vertices possible for the specified tolerance, since there can be some segments with an error within the

tolerance that are discarded by the detour test. However, the algorithm is very simple and experiments show that it produces approximations with a considerable reduction in the number of points.

Proposition 5.1. *Given a polygonal chain P with n vertices, the dynamic programming algorithm finds an approximation chain Q, for the segment criterion, in $O(n^2)$ time.*

Chapter 6

Software

We implemented a suite of programs for the extraction of contours using the geometric algorithms described in this thesis. Our software package has the following components:

1. A library `lib.c` and `lib.h` that includes basic data structures and functions for the definition of contours (such as oriented points, and paths), input/output, and useful geometric functions.

2. Some data format definitions such as those used for points (`.pnt`) and paths (`.pth`), and conversion tools `jpg2ppm.java` and `pth2fig.c`.

3. A collection of programs for extracting and manipulating contours from an image: `sobel.c`, `cluster.c`, `link.c`, and `simplify.c`. Each of these programs correspond to the steps of our method, input conversion, clustering, linking, and simplification, respectively.

4. A visualization program `show.c` for displaying points and contours.

5. Helper programs for running experiments and analyzing results: `tcgen.c`, `tcapx.c`, `bimcpnt.c`, `noise.c`, `discretize.c`, `compare.c`, `stats.c`.

6. Traditional UNIX Makefiles for automating program compilation and experiments.

Appendix A contains the documentation of our programs and appendices B to E contain the source code of our programs. The documentation will be available online[1] later, with additional experiments and results.

[1] Possibly at `http://www.cs.usu.edu/~mjiang/contour/`.

6.1 Program Pipeline

Fig. 6.1 illustrates a typical use of our software. For simplicity, our programs work with images in PPM[2] file format. A tool is used to convert images in other formats[3] to PPM before processing, and the output of the programs is stored in point (`.pnt`) and path (`.pth`) files. The conversion tool `jpg2ppm` converts a given JPG image to PPM format, the edge detection program `sobel` extracts points from the PPM image, the clustering program `cluster` reduces the number of points, the linking program `link` connects the points into paths, the simplification program `simplify` reduces the number of points in the paths and gets rid of unused points, and finally the visualization program `show` displays the extracted contours.

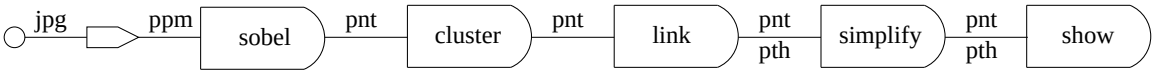


Fig. 6.1: Program pipeline for extraction and visualization of contours from a JPG image. The type of output of each step is shown: `.jpg` and `.ppm` are image formats, `.pnt` is a points file, and `.pth` is a paths file.

The pipeline is encapsulated by the Makefiles in simple commands. Given an image `peppers.jpg`, the command `make peppers.c.s.show` executes all the programs in the pipeline and saves the result in two files describing the contours, `peppers.c.s.pnt` and `peppers.c.s.pth`, that are displayed by the `show` program. When running the command, parameter values can be specified for all programs. If no value is specified, default values are used. Fig. 6.2 shows an example output.

To visualize the result better, the `show` program has options to hide and display points, point orientations, paths (contours), and a background image with different levels of transparency.

[2]Portable Pixel Map. For a detailed description see `http://netpbm.sourceforge.net/doc/ppm.html`.
[3]We only support JPG files, but support for other formats could easily be added.

(a) (b) (c)

Fig. 6.2: Example output for `peppers.jpg` after running command `make peppers.c.s.show ST=0.3`. (a) Original image. (b) Points and orientations. (c) Contours (paths).

6.2 Default Parameter Values

Since every step of the process requires some parameters, it is useful to have some good default values. We now describe how we choose default values for all parameters. The closest pair distance $d_{\min}$ is used to select appropriate values for some of the parameters.

For the Sobel edge detector we use $\mu + \frac{1-\mu}{3}$ as the magnitude threshold, where μ is the average magnitude after normalizing to $[0, 1]$ with the maximum magnitude of all pixels mapped to 1.

For clustering, if the maximum distance $d_{\max}$ allowed to merge points is too large, small details will be lost. Therefore, we use a small distance that would still at least merge adjacent pixels. Since the closest pair distance is most probably the distance between two adjacent pixels (4-adjacency[4]) extracted by the edge detector, by choosing $d_{\max} = 2 \cdot d_{\min}$ we make sure that any two points from adjacent pixels (including diagonals) can be merged.

When linking points, if the maximum distance $d_{\max}$ allowed to link points is too small, then contours will have breaks; and if the distance is too large, false contours can be detected. We use $d_{\max} = 3 \cdot d_{\min}$. This value allows the algorithm to find contours even when there are small discontinuities or when some intermediate points have bad orientations due to noise. For the maximum difference in orientation allowed to link points, it is important

[4]With 4-adjacency, pixels are considered adjacent if they are next to each other and in the same row or column: a pixel (x, y) has four adjacent pixels $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$, and $(x, y - 1)$.

to use a value that is not too small in order to be able to link points of curved contours, yet not too large in order to prevent linking points from different contours that are close to each other. We use $\alpha_{\max} = 35°$.

For simplification we need a small error tolerance ε to make sure that the simplified contours are similar to the originals. We assume that the distance between points is small before simplification and use $\varepsilon = \frac{1}{2}d_{\min}$. For most images this small value makes sure that the visual difference between the original and the simplified contours is hardly perceivable.

After the paths have been simplified, it is useful to do some filtering, in order to remove isolated points not belonging to any path, and paths under a certain length that might exist due to noise in the original image. By default we remove isolated points, but no paths are removed.

Chapter 7

Experiments

To evaluate our method, we performed tests with a variety of synthetic and natural images. We used synthetic images to test the accuracy of the extracted contours, and natural images to determine the level of compression attained by our method.

To test the accuracy, we used a set of randomly generated shapes and a set of binary images of different objects for which the expected results were known. The results obtained by our method were compared with the expected results, and the difference between them was measured to evaluate the quality of the extracted contours. For natural images, we do not have a description of the expected result; however, the quality of the results can still be evaluated subjectively by visual inspection. To illustrate the quality of the extracted contours, we show some example results from both synthetic and natural images.

To test the compression we compared the number of points extracted by the edge detector with the number of points after the clustering and simplification steps.

7.1 Test Platform

Experiments were performed on a Dell Inspiron 6400 laptop with a 1.83GHz Intel Core2 Duo T5600 processor and a Dell OptiPlex GX620 with 2.8 GHz Pentium D processor, both with 2 GB RAM, running Windows XP Professional Service Pack 3 and Cygwin 1.5.25(0.156/4/2).

7.2 Evaluating Accuracy

Recent methods for evaluation of contour extraction or boundary detection use images with associated ground truths [14, 25, 35]. Given an image, the *ground truth* specifies the real contours, which can be compared with the extracted contours to make objective

evaluations about their quality. For synthetic images, the ground truth can usually be determined automatically. For natural images, the ground truth is usually determined by one or multiple human subjects drawing over the original image. Once the ground truth has been determined it is usually stored as a binary image with contour pixels set to true and all other background pixels set to false.

To compare the ground truth of an image with the extracted contours, some methods count the number of *true positives* as the contour pixels correctly detected, *false negatives* as the contour pixels missed, and *false positives* as the contour pixels detected incorrectly. Then such methods use traditional statistical measures to evaluate the result, such as sensitivity, selectivity, and accuracy:

$$\begin{aligned}
\text{Sensitivity} &= \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false negatives}} \\
\text{Selectivity} &= \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false positives}} \\
\text{Accuracy} &= \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false negatives} + \#\text{ false positives}}
\end{aligned}$$

Both sensitivity and selectivity are widely used standards of quality measurement. In this case, sensitivity measures the proportion of the real contours that is detected correctly, and selectivity measures the proportion of the extracted contours that is correct. There is a trade-off between these measures, and both should be high to qualify the result as being good.

Accuracy is used in image processing as a performance measurement [14]. It is a mix of sensitivity and selectivity and can only be optimal (100%) if the extracted contours are an exact match of the ground truth. Given the ground truth and the contours extracted from an image, the accuracy is their intersection divided by their union [22]. In this sense, it is also used as the intersection of areas divided by their union for the performance measure to evaluate the detection of closed boundaries [35].

The problem with these measures is that they work with discrete binary data, such as pixels that do or do not belong to a set and, hence, cannot be applied directly to evaluate

the performance of our method, which works with continuous lines. For this reason, we must use another measure.

We use the Hausdorff distance as the performance measure to evaluate our results. Given two sets of points X and Y, the *directed* Hausdorff distance between X and Y, $h(X,Y)$, is defined as

$$h(X,Y) = \sup_{x \epsilon X} \inf_{y \epsilon Y} |xy|, \tag{7.1}$$

where $|xy|$ is the distance between x and y. Then the *combined* Hausdorff distance, or simply Hausdorff distance, is defined as

$$H(X,Y) = \max\{h(X,Y), h(Y,X)\}. \tag{7.2}$$

The directed Hausdorff distance indicates the maximum distance from a point in X to the nearest point in Y; if it is small, all points in X are close to some point in Y. The combined Hausdorff distance indicates the maximum distance from any point in one of the sets, X or Y, to a point in the other set; and if it is small, a high degree of similarity between the two sets is indicated.

The Hausdorff distance is commonly used in pattern recognition. For example, an extension of the Hausdorff distance is used in [18] to compare a portion of a model against an image; and a version of the Hausdorff distance that uses location and orientation is used in [19] to locate objects in an image. However, it is not commonly used to evaluate contour extraction performance, because it is quite sensitive to noise.

Nevertheless, it is useful for our purposes since it can be applied to continuous lines instead of discrete pixels. Given an image, let G be its ground truth, and C be a set of extracted contours. Then $h(C,G)$, $h(G,C)$, and $H(C,G)$, can be used as alternative measures to the sensitivity, selectivity, and accuracy. If $h(C,G)$ is small, all parts of the extracted contours are close to the ground truth, which means that all detected contours are real; if $h(G,C)$ is small, all parts of the ground truth are close to some extracted contour, which means that all detected contours are complete; and if $H(C,G)$ is small, then both

$h(C, G)$ and $h(G, C)$ are small, and the extracted contours are both real and complete.

Fig. 7.1 illustrates the method used to test the accuracy of the results. For each test case there is a *source* file and an *expected* result file. Contours are extracted from the source file, which can be an image or a set of points, and compared with the expected result file. For simplicity, we only compare sets of points; therefore, the result is always a set of points that approximate the real contours, and the extracted contours are discretized into points before making the comparison. For each test case we compute the Hausdorff distance between the obtained and the expected result, and then compute the average and standard deviation.

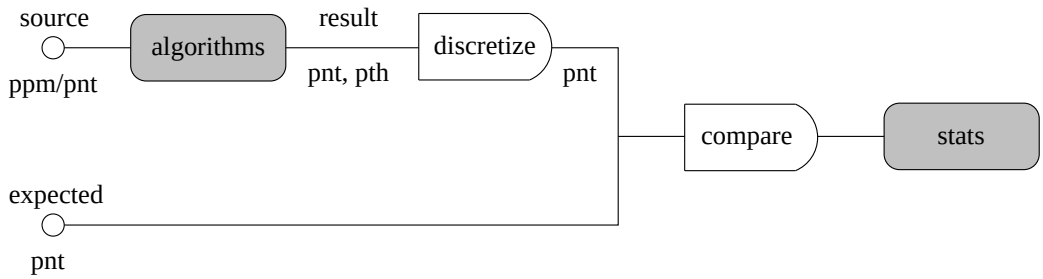


Fig. 7.1: Flowchart for evaluating accuracy by comparing obtained results with expected results. The type of file at each step is shown below each line: `.ppm` is an image, `.pnt` is a points file, and `.pth` is a paths file.

7.3 Evaluating Compression

We evaluate the amount of compression obtained by our method by comparing the number of points extracted by the edge detector with the number of points after the clustering and simplification steps. The ratio of the number of original points extracted by the edge detector to the number of points after each step gives a measure of the level of compression. If the ratio is larger, the compression is better, and the result is more compact.

7.4 Results

We now describe the results of our experiments. Default parameter values were used for all experiments, unless otherwise specified.

7.4.1 Random Shapes

To evaluate the accuracy of our method, we first tried it with a set of random pictures, each containing a few (from 1 to 20) randomly generated shapes. We tested it with 15 test sets, each one with 20 test cases, containing different combinations of line segments and ellipses.

Fig. 7.2 illustrates how the source files and expected files are generated for these tests. First, shapes are generated and stored in a test case file from which the source and expected files are generated. The test case file contains an exact definition of the shapes; for example, a line segment is defined by two points, and an ellipse is defined by its foci and the lengths of its axes. From the test case file, the program `tcapx` generates an approximation of the shapes by discretizing it into points; these points are then used as the expected result; and the `noise` program adds noise to obtain the source file. The process continues from the source and expected files, as shown in Fig. 7.1.

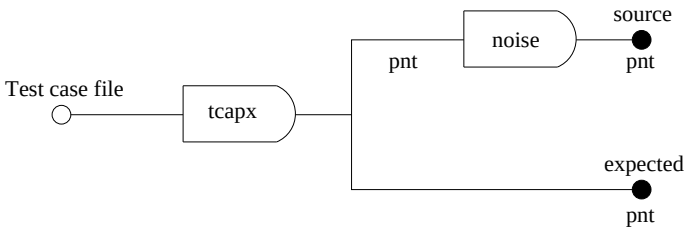


Fig. 7.2: Creating source files and expected files for random shapes tests.

The source files for the random shapes experiments are point files; therefore, with these experiments we can only test the geometric algorithms. When points are generated, their orientation is set to be tangent to the given curve (line or ellipse). When noise is generated, their orientations and their locations are random.

The results for different amounts of noise are shown in Table 7.1. Because the points are not equally spaced, as when they are extracted from an image, we set the value of the maximum distance parameter for the clustering algorithm to be 0.002 (approximately one pixel away for a 512 x 512 image). To remove false contours detected because of noise, we remove segments with length less than 0.05. The results for different amounts of noise

after removing short segments are shown in Table 7.2. Both Table 7.1 and Table 7.2 show the directed Hausdorff distance from the extracted contours to the expected results $h(C, G)$ and from the expected results to the extracted contours $h(G, C)$, the combined Hausdorff distance between the two $H(C, G)$, and the number of paths.

Table 7.1: Random Shapes Results Without Removing Short Segments.

Noise points	$h(C, G)$	$h(G, C)$	$H(C, G)$	Number of paths
0	0.0012 ± 0.0002	0.0020 ± 0.0036	0.0020 ± 0.0036	14.72 ± 22.62
5000	0.4822 ± 0.2216	0.0021 ± 0.0034	0.4822 ± 0.2216	192.66 ± 19.13
10000	0.4942 ± 0.2211	0.0020 ± 0.0027	0.4942 ± 0.2211	629.60 ± 11.62

Table 7.2: Random Shapes Results After Removing Segments Shorter Than 0.05.

Noise points	$h(C, G)$	$h(G, C)$	$H(C, G)$	Number of paths
0	0.0012 ± 0.0002	0.0148 ± 0.0323	0.0148 ± 0.0322	12.01 ± 16.21
5000	0.0018 ± 0.0013	0.0167 ± 0.0324	0.0169 ± 0.0324	12.38 ± 16.22
10000	0.0023 ± 0.0018	0.0179 ± 0.0328	0.0182 ± 0.0326	12.68 ± 16.30

The low values obtained after removing short segments indicate that most contours are detected correctly, even with a high amount of noise.

7.4.2 Binary Images

To evaluate the accuracy of our method, including the pre-processing stage, we tested it with the images from the MPEG7 CE Shape–1 Part B database[1]. This is a collection of 1400 black-and-white images of simple to moderately complex shapes commonly used for the evaluation of shape descriptors and object recognition and classification algorithms [23].

Fig. 7.3 illustrates how the source files and expected files are generated for these tests. First, the `jpg2ppm` program converts the JPG image into a PPM image; then the `noise` program adds noise to obtain the source file, and the `bimcpnt` program extracts known contour points from the PPM image to create the expected result file. Since the images are binary with white pixels representing the shapes, the points for the expected result file extracted by the `bimcpnt` program are the points corresponding to the white pixels that

[1] `http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm`.

have an adjacent black pixel in the horizontal or vertical direction. After the source and expected files are generated, the process continues as shown in Fig. 7.1.

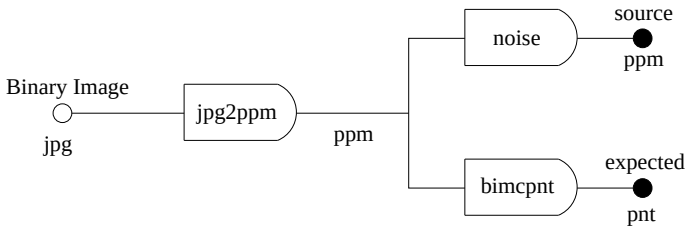


Fig. 7.3: Creating source files and expected files for binary image tests.

The results for different amounts of noise are shown in Table 7.3. To remove false contours detected because of noise, we remove segments with a length less than or equal to 0.05. The results for different amounts of noise after removing short segments are shown in Table 7.4. Tables 7.3 and 7.4 both show the directed Hausdorff distance from the extracted contours to the expected results $h(C, G)$ and from the expected results to the extracted contours $h(G, C)$, the combined Hausdorff distance between the two $H(C, G)$, and the number of paths. The noise density is the percent of the number of pixels of each image (randomly selected with replacement) that are set to random values.

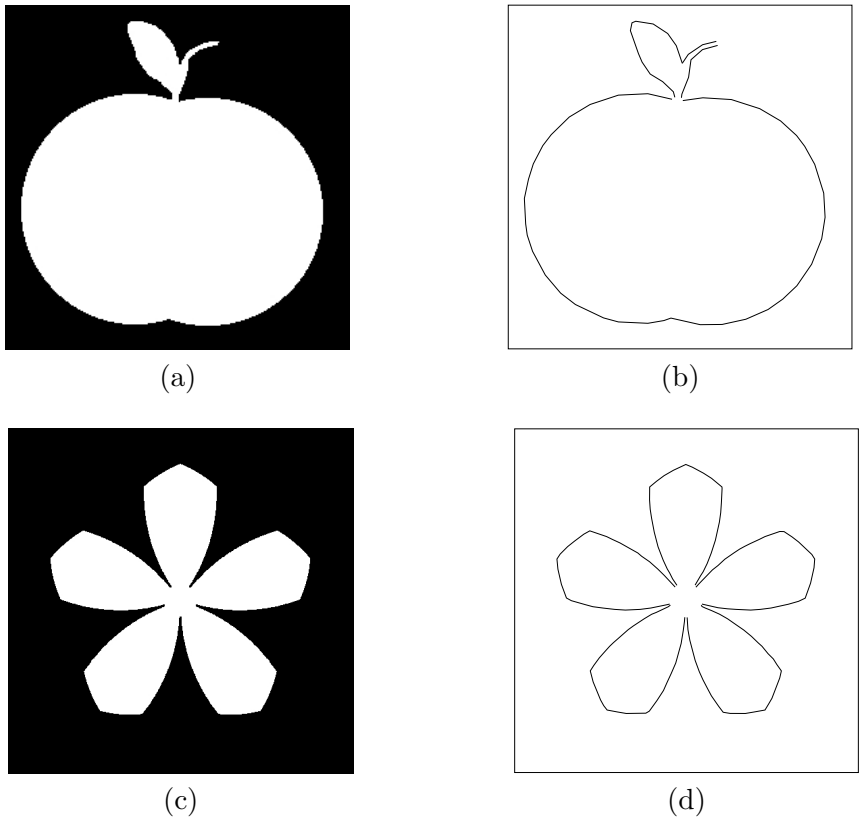
Table 7.3: Binary Images Results Without Removing Short Segments.

Noise density	$h(C, G)$	$h(G, C)$	$H(C, G)$	Number of paths
0 %	0.0050 ± 0.0053	0.0200 ± 0.0366	0.0201 ± 0.0368	20.42 ± 41.96
2 %	0.3032 ± 0.1530	0.0164 ± 0.0226	0.3044 ± 0.1509	57.01 ± 52.62
5 %	0.3422 ± 0.1593	0.0147 ± 0.0189	0.3429 ± 0.1580	127.18 ± 93.56

Table 7.4: Binary Images Results After Removing Segments Shorter Than 0.05.

Noise density	$h(C, G)$	$h(G, C)$	$H(C, G)$	Number of paths
0 %	0.0049 ± 0.0053	0.0338 ± 0.0507	0.0339 ± 0.0508	9.57 ± 11.19
2 %	0.0079 ± 0.0172	0.0347 ± 0.0550	0.0362 ± 0.0566	10.05 ± 11.38
5 %	0.0163 ± 0.0432	0.0349 ± 0.0512	0.0431 ± 0.0631	10.83 ± 11.66

The low values obtained after removing short segments indicate that most contours are detected correctly, even with a high amount of noise. Some example results are shown in Fig. 7.4.



(a) (b)

(c) (d)

Fig. 7.4: Binary images results. (a) Sample binary image `apple-11`. (b) Contours for `apple-11`: 61 points, 4 paths. (c) Sample binary image `device0-8`. (d) Contours for `device0-8`: 150 points, 5 paths.

7.4.3 Natural Images

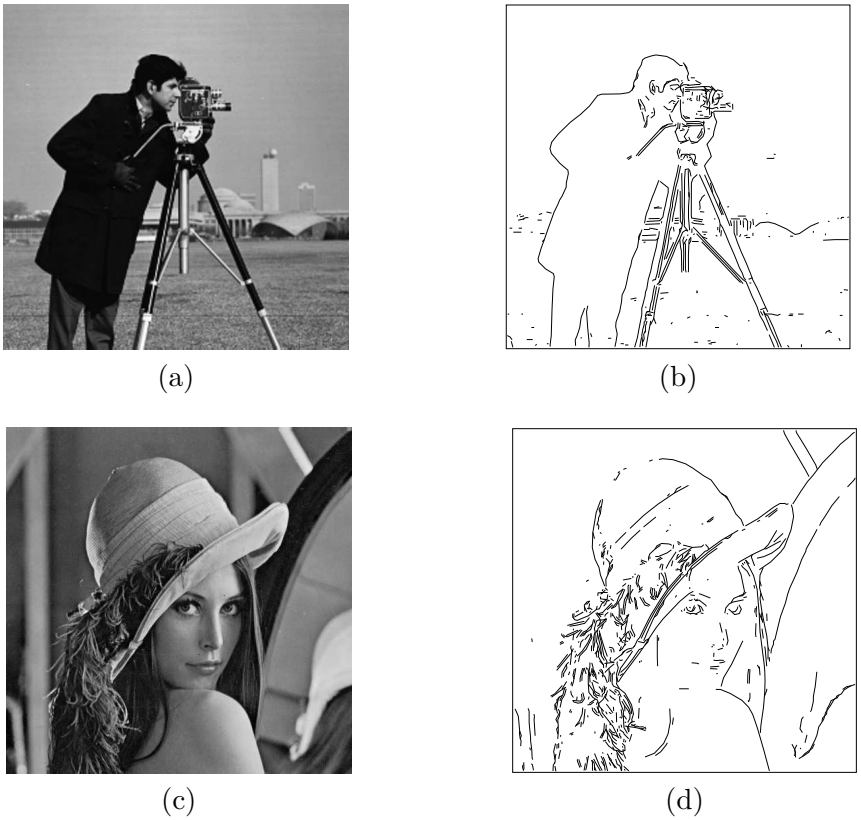
To evaluate the amount of compression obtained by our method, we employed some commonly used test images from "The USC-SIPI Image Database"[2] as well as some images of our own. For each image we determine the number of points extracted by the edge detector at the pre-processing stage, the number of points after the clustering step, and the number of points and paths after the linking and simplification steps. The compression results for natural images when using a Sobel threshold of 0.2 are shown in Table 7.5. The number of times reduction for points after clustering and simplification are shown in parenthesis. Images marked with a star (*) are autor's images; other images are standard test images. Most image sizes are 512 x 512.

Table 7.5: Compression Results.

Image (jpg)	Points (edge detector)	Points (clustering)	Points (simplification)	Paths
cameraman	18610	2959 (6.29)	979 (19.01)	289
jetplane	29466	5047 (5.84)	2046 (14.40)	711
lena	20233	4008 (5.05)	1532 (13.21)	506
livingroom	27227	6358 (4.28)	2385 (11.42)	878
peppers	14631	2802 (5.22)	890 (16.44)	278
rose*	35430	7303 (4.85)	2598 (13.64)	741
text*	13191	2221 (5.94)	502 (26.28)	69
walkbridge	59474	13381 (4.44)	6364 (9.35)	2547
woman_blonde	18137	4376 (4.14)	1596 (11.36)	539
Avg. ± stdv.	26266.56 ± 14427.99	5383.89 ± 3430.71 (5.12 ± 0.77)	2099.11 ± 1745.17 (15.01 ± 5.11)	728.67 ± 728.55

From the numbers in Table 7.5, we can see that the amount of compression obtained by the clustering and simplification steps is considerable. The number of points is always reduced at least 9 times, with an average of 15 times reduction. Some example results are shown in Fig. 7.5.

[2] http://sipi.usc.edu/database/index.html



(a) (b)

(c) (d)

Fig. 7.5: Natural images results. (a) Sample image `cameraman.jpg`. (b) Contours for `cameraman.jpg`: 979 points and 289 paths obtained from 18610 original points by running command `make cameraman.c.s.show SN=-1 ST=0.2`. (c) Sample image `lena.jpg`. (d) Contours for `lena.jpg`: 1532 points and 506 paths obtained from 20233 original points by running command `make lena.c.s.show SN=-1 ST=0.2`.

Chapter 8

Conclusions

Existing methods and benchmarks to evaluate the performance of contour extraction algorithms do not evaluate the connectivity of the contours. The result of most algorithms is a binary image indicating which pixels are part of the contours, and thus benchmarks only evaluate things like accuracy, noise sensitivity, and the ability to detect contour pixels in images with complex textures. In our case, however, we cannot use the existing evaluation methods because our output is a set of continuous lines and we want to evaluate the connectivity. Consequently, we use the Hausdorff distance as a measure of accuracy and the number of detected paths as an indication of connectivity. Our experimental results show that our method can effectively extract contours from digital images. The small Hausdorff distance measures indicate that most of the contours are detected correctly, while the small number of paths detected is an indication that the connectivity is good. Compression results also show that the resulting contours are much more compact than the ones obtained by a pixel-based representation. Also, in terms of speed it is possible to have fast implementations: $O(n \log n)$ time for clustering and linking, and $O(n^2)$ time for simplification.

Our method has some advantages over other methods. For example, it is conceptually simpler than other algorithms that have to consider multiple special cases; it is easy to select default parameters for it that can work well with many images; it can work on a set of points instead of an image; and it can easily fill contour gaps or discontinuities.

However, it also has some disadvantages. Like other linking-based algorithms, it is sensitive to noise; it can still find contours for images with a relatively high amount of noise, but the connectivity is affected as contours are sometimes broken at several places. It cannot detect small details because the spacing between the points is increased by the

clustering step. It does not detect sharp corners because the change in orientation is large at such corners, while the linking algorithm connects points with similar orientation.

It is interesting to note that some algorithms are customizable by either weight or error functions. Furthermore, they could, in principle, be used independently of the rest. For example, the clustering algorithm could be used all by itself, while the linking algorithm could be used with different weight functions. The simplification algorithm is also independent and could be used on any set of chains, not necessarily only those obtained as contours from an image. Also, when using our method some steps could be replaced for different applications, while the rest of the steps are left unchanged.

In summary, we have presented a method for extracting contours from digital images, that uses computational geometry as an alternative to pixel-based methods from image processing. The results show that while it has some disadvantages, its advantages are still very promising. Some possible directions for future work include finding a way to detect sharp corners, using elements other than straight lines to represent contours (arcs, circles, splines), and finding ways to reduce the sensitivity to noise, perhaps by using better weight functions or some filtering steps.

References

[1] T. Asano, V. E. Brimkov, and R. P. Barneva, "Some theoretical challenges in digital geometry: A perspective," *Discrete Applied Mathematics*, 2009, doi:10.1016/j.dam.2009.04.022.

[2] J. L. Bentley and M. I. Shamos, "Divide-and-conquer in multidimensional space," in *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, 1976.

[3] S. N. Bespamyatnikh, "An optimal algorithm for closest pair maintenance," in *Proceedings of the 11th Annual Symposium on Computational Geometry*, 1995.

[4] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[5] W. S. Chan and F. Chin, "Approximation of polygonal curves with minimum numbers of line segments or minimum error," *International Journal of Computational Geometry and Applications*, vol. 6, pp. 59–77, 1996.

[6] M. de Berg, P. Bose, O. Chcong, and P. Morin, "On simplifying dot maps," *Proceedings of the 18th European Workshop on Computational Geometry*, vol. 27, pp. 43–62, 2003.

[7] A. Ebbers-Baumann, R. Klein, E. Langetepe, and A. Lingas, "A fast algorithm for approximating the detour of a polygonal chain," *Computational Geometry: Theory and Applications*, vol. 27, pp. 123–134, 2004.

[8] N. Efford, *Digital Image Processing: A Practical Introduction Using Java*. Harlow, England; New York: Addison Wesley, 2000.

[9] D. Eu and G. T. Toussaint, "On approximating polygonal curves in two and three dimensions," *CVGIP: Graphical Models and Image Processing*, vol. 56, pp. 231–246, 1994.

[10] H. Freeman, "On the encoding of arbitrary geometric configurations," *IEEE Transactions Electronic Computers*, vol. 10, pp. 260–268, 1961.

[11] H. Freeman, "Computer processing of line drawings," *ACM Computing Surveys*, vol. 6, pp. 57–97, 1974.

[12] J. E. Goodman and J. O'Rourke, *Handbook of Discrete and Computational Geometry*, 2nd ed. Boca Raton, FL: CRC Press, 2004.

[13] M. T. Goodrich and R. Tamassia, *Algorithm Design: Foundations, Analysis, and Internet Examples*. Hoboken, NJ: John Wiley & Sons, 2002.

[14] C. Grigorescu, N. Petkov, and M. A. Westenberg, "Contour detection based on nonclassical receptive field inhibition," *IEEE Transactions on Image Processing*, vol. 12, no. 7, pp. 231–236, 2003.

[15] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2008.

[16] D. S. Guru, B. H. Shekar, and P. Nagabhushan, "A simple and robust line detection algorithm based on small eigenvalue analysis," *Pattern Recognition Letters*, vol. 23, pp. 1-13, 2003.

[17] P. S. Heckbert and M. Garland, "Survey of polygonal surface simplification algorithms," Technical Report, Carnegie Mellon University, School of Computer Science, 1997.

[18] D. Huttenlocher, G. Klanderman, and W. Rucklidge, "Comparing images using the hausdorff distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 850–863, 1993.

[19] D. Huttenlocher and C. Olson, "Automatic target recognition by matching oriented edge pixels," *IEEE Transactions on Image Processing*, vol. 6, no. 1, pp. 103–113, 1997.

[20] H. Imai and M. Iri, "Computational-geometric methods for polygonal approximations of a curve," *Computer Vision, Graphics, and Image Processing*, vol. 36, pp. 31–41, 1986.

[21] M. Jiang and J. Gillespie, "Engineering the divide-and-conquer closest pair algorithm," *Journal of Computer Science and Technology*, vol. 22, pp. 532–540, 2007.

[22] M. Jiang, P. J. Tejada, R. O. Lasisi, S. Cheng, and D. S. Fechser, "K-partite RNA secondary structures," in *Proceedings of the 9th Workshop on Algorithms in Bioinformatics (WABI'09)*. 2009, to appear.

[23] L. J. Latecki, R. Lakämper, and U. Eckhardt, "Shape descriptors for non-rigid shapes with a single closed contour," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2000.

[24] S. Marchand-Maillet and Y. M. Sharaiha, *Binary Digital Image Processing*. San Diego, CA: Academic Press, 2000.

[25] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proceedings of the 8th International Conference on Computer Vision (ICCV'01)*, 2001.

[26] A. Melkman and J. O'Rourke, "On polygonal chain approximation," in *Computational Morphology* (G. T. Toussaint, ed.), Amsterdam: North-Holland, 1988, pp. 87–95.

[27] G. A. Moore, "Automatic scanning and computer processes for the quantitative analysis of micrographs and equivalent subjects," *Pattern Recognition: Pictorial Pattern Recognition*, vol. 1, pp. 275–326, 1969.

[28] R. Nevatia and K. R. Babu, "Linear feature extraction and description," *Computer Graphics and Image Processing*, vol. 3, pp. 257–269, 1980.

[29] L. G. Roberts, "Machine perception of three dimensional solids," in *Optical and Electro-Optical Information Processing* (J. T. Tippett et al., eds.), Cambridge, MA: MIT Press, 1965, pp. 159–197.

[30] G. S. Robinson, "Detection and coding of edges using directional masks," in *Proceedings SPIE Conference on Advances in Image Transmission Techniques*, August 1976.

[31] S. S. Skiena, *The Algorithm Design Manual*, 2nd ed. London: Springer, 2008.

[32] G. T. Toussaint, "On the complexity of approximating polygonal curves in the plane," in *Proceedings of IASTED International Symposium on Robotics and Automation*. Lugano, Switzerland, 1985.

[33] G. T. Toussaint, "Computational geometry and computer vision," *Contemporary Mathematics*, vol. 119, pp. 213–224, 1991.

[34] S. E. Umbaugh, *Computer Imaging: Digital Image Analysis and Processing*. Boca Raton, FL: CRC Press, 2005.

[35] S. Wang, F. Ge, and T. Liu, "Evaluating edge detection through boundary detection," *EURASIP Journal on Applied Signal Processing*, vol. 2006, pp. 1–15, 2006.

Appendices

Appendix A

Documentation Website

Library

A library that includes basic data structures and functions for the definition of contours (such as points, and paths), input/output, and useful geometric functions.

Points

Structures

```
typedef struct {
        double x;        /* x coordinate */
        double y;        /* y coordinate */
        double a;        /* orientation angle */
        double w;        /* weight */
} s_point;
```

The structure `s_point` represents an oriented point. It has two fields `x` and `y` for the 2D coordinates of the point, a field `a` for the orientation, and a field `w` for the weight of the point. This weight is a non visible property which is used internally by the algorithms. The values for the `x`, `y`, `a` fields are normalized between `0.0` and `1.0`:

- `x`: the leftmost and rightmost possible coordinates are represented by `0.0` and `1.0`, respectively.
- `y`: the lowest and highest possible coordinates are represented by `0.0` and `1.0`, respectively.
- `a`: `0.0` and π are mapped to `0.0` and `1.0`, respectively.
- `w`: can be any value greater than or equal to `0.0`.

```
typedef struct {
        s_point *points;    /* points */
        int size;           /* array size */
        int n;              /* number of points */
} s_point_array;
```

The structure `s_point_array` maintains a resizable array of points. It is used for convenience when the number of points cannot be anticipated.

Utility functions

```
s_point *new_point(double x, double y, double a, double w);
void print_point(s_point *p);
void print_points(s_point *p, int n_points);
```

The function `new_point` creates an `s_point` with the specified property values. The functions `print_point` and `print_points` output one or a collection of points to the console, respectively.

```
s_point_array *new_point_array(int n);
void extend_point_array1(s_point_array *pnt_array, s_point *point);
void extend_point_array(s_point_array *pnt_array, s_point *points, int n_points);
```

The function `new_point_array` creates an `s_point_array` with capacity for at least `n` points. The functions `extend_point_array1` adds one point to the end of the array. The functions `extend_point_array` adds multiple points to the end of the array. When calling the `extend_point_array?` functions, the size of the array is automatically increased if more space is needed to add points.

Input/Output

```
s_point *input_points(char *file_name, int *n_points);
void output_points(char *file_name, s_point *points, int n_points);
```

The function `input_points` reads a `pnt` file and returns the array of points and the number of points read. The function `output_points` outputs a collection of points in `pnt` format.

Paths

Structures

```
typedef struct {
        int *indices;   /* indices to points in path */
        int size;       /* indices array size */
        int length;     /* number of points in path */
} s_path;
```

The structure `s_path` represents a path between multiple points. The `indices` field is an array of the indices of the points that are part of the path, in order. The field `size` stores the size of the `indices` array. The field `length` indicates the number of points in the path.

Utility functions

```
s_path *new_path(int length);
void reverse_path(s_path *p);
void extend_path(s_path *p, int i_point);
void print_path(s_path *p);
void print_paths(s_path *p, int n_paths);
```

The function `new_path` creates an `s_path` with initial capacity for at least `length` points. The function `reverse_path` reverses the order of the points in a path. The function `extend_path` adds a point, specified by the `i_point` index parameter, to the end of a path; and the size of the `indices` array is automatically increased if it is necessary. The functions `print_path` and `print_paths` output one or a collection of paths to the console, respectively.

Input/Output

```
s_path **input_paths(char *file_name, int *n_paths, int *n_points);
void output_paths(char *file_name, s_path **paths, int n_paths, int n_points);
```

The function `input_paths` reads a `pth` file and returns an array of path pointers, the number paths read, and the number of points (not all points are necessarily part of a path). The function `output_paths` outputs a collection of paths in `pth` format.

Image

Structures

```
#define MAX_VALUE 255

typedef struct {
        unsigned char r;
        unsigned char g;
        unsigned char b;
} s_pixel;

typedef struct {
        s_pixel **p;    /* pixel data */
        int w;          /* width */
        int h;          /* height */
} s_image;
```

The constant `MAX_VALUE` represents the maximum intensity for each color component (RGB) of an image.

The structure `s_pixel` represents an RGB pixel where the `r`, `g`, and `b` fields correspond to the red, green, and blue color components, respectively. Gray scale images are represented as color images, with all three components for each pixel having the same value.

The structure `s_image` represents a digital image where the fields `w` and `h` are its width and height and the field `p` is a 2D array of size `(w * h)` for the pixel's data.

Utility functions

```
s_image *new_image(int w, int h);
double get_point_x(int w, int h, int x);
double get_point_y(int w, int h, int y);
double get_pixel_x(int w, int h, double x);
double get_pixel_y(int w, int h, double y);
```

The function `new_image` creates an `s_image` of width `w` and height `h`, and with all pixels set to white. The function `get_point_x` returns the x coordinate between `0.0` and `1.0` corresponding to the `x` (horizontal) coordinate of an image of size `(w * h)`. The function `get_point_y` does the same but for the `y` (vertical) coordinate. The function `get_pixel_x` returns the x pixel coordinate between `0` and `w - 1`, of an image of size `(w * h)`, corresponding to the `x` coordinate of a point. The function `get_pixel_y` does the same but for the `y` coordinate, and returns a value between `0` and `h - 1`.

Input/Output

```
s_image *load_ppm(char *file_name);
void save_ppm(s_image *img, char *file_name);
```

The function `load_ppm` reads a `ppm` image file. The function `save_ppm` outputs an image in `ppm` format.

Geometric functions

```
double dist(s_point *p1, s_point *p2);
double dist2(s_point *p1, s_point *p2);
double dist2seg(s_point *p1, s_point *q1, s_point *q2);
```

The function `dist` returns the distance between two points. The function `dist2` returns squared distance between two points. The function `dist2seg` returns the shortest distance from point `p1` to any point of the segment specified by the points `q1` and `q2`.

```
double angle_diff(double a1, double a2);
double vector_angle(s_point *p1, s_point *p2, s_point *p3);
```

The function `angle_diff` returns the difference between two angles, less than or equal to $\pi/2$. The function `vector_angle` returns the angle, between `0.0` and π, between the vectors `<p1, p2>` and `<p2, p3>`.

Data Formats

1. Points (.pnt):

   The first line specifies the number of points, and each following line specifies a point. Each point line contains four numbers `x y a w`, for the 2D coordinates, the orientation, and the weight of the point, respectively.

   Example:

   ```
   4
   0.448242 0.260742 0.500000 1.000000
   0.432617 0.182617 0.500000 1.000000
   0.448242 0.258789 0.500000 1.000000
   0.432617 0.180664 0.500000 1.000000
   ```

   This file contains four points with different coordinates and all with an orientation angle of `0.5` and a weight of `1.0`.

2. Paths (.pth):

   The first line contains two numbers which specify the number of paths and points. Each following line specifies a path: the first number is the number of points in the paths; the following numbers are indices into an array of points.

   Example:

   ```
   4 430
   7 415 81 50 148 245 215 419
   7 73 2 208 38 110 28 11
   9 366 332 281 398 314 339 296 228 167
   2 414 119
   ```

   The first line of this file indicates that there are four paths among `430` possible points. The following four lines specify the paths. For example, the second line specifies a path with `7` points, starting at point `415` and ending at point `419`.

3. Image (.ppm):

   The first line is always "P3" which indicates the pixel values are in ASCII format. After that there can be an optional comment line, which must start with a '#'. Next are three numbers: width, height, and maximum intensity value. After that are the RGB values for each pixel.

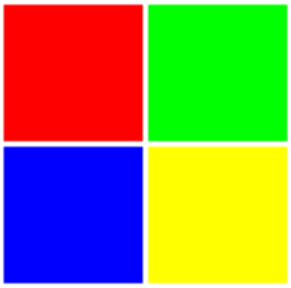
   See `Netpbm format`[1] for a description of the PPM image format.

   Example:

   ```
   P3
   # created by jpg2ppm
   2 2 255
   255 0 0
   0 255 0
   0 0 255
   255 255 0
   ```

   This is a 2 x 2 image with the following colors:

[1] `http://en.wikipedia.org/wiki/Netpbm_format`.



4. JPEG (.jpg):

   The algorithm can process JPEG images which are first converted to the `ppm` format.

   See `JPEG`[2].

   Example:

5. FIG (.fig):

   The `paths` output by the algorithm can be converted to FIG format, which can later be converted to EPS or PDF.

   See `Fig Format 3.2`[3].

[2] `http://en.wikipedia.org/wiki/JPEG`.

[3] `http://www.xfig.org/userman/fig-format.html`.

Convert

`jpg2ppm`

From `jpg` image to `ppm` image.

`pth2fig`

From `paths` to `fig`.

jpg2ppm

Converts a `jpg` image to `ppm` image. It can also display a `jpg` or `ppm` image.

USAGE

```
java -jar jpg2ppm.jar input_file_name [output_file_name] [options]
```

OPTIONS

`input_file_name`
: Input file name. The input file must be a `jpg` or `ppm` image. If no output file is specified then the input file is displayed.

`output_file_name`
: Optional output file. The output file is a `ppm` image. If no output file is specified then the input file is displayed.

`-g`
: Convert image to gray-scale before saving or displaying.

EXAMPLES

`java -jar jpg2ppm.jar jetplane.jpg`
: Displays `jetplane.jpg` image.

`java -jar jpg2ppm.jar jetplane.ppm -g`
: Displays `jetplane.ppm` image in gray-scale.

`java -jar jpg2ppm.jar jetplane.jpg jetplane.ppm`
: Convert `jetplane.jpg` to `ppm` and save it as `jetplane.ppm`.

`java -jar jpg2ppm.jar jetplane.jpg jetplane.ppm -g`
: Convert `jetplane.jpg` to `ppm` and save it in gray-scale as `jetplane.ppm`.

pth2fig

Converts `paths` to `fig` format.

USAGE

```
pth2fig points_file_name paths_file_name fig_file_name [options]
```

OPTIONS

`points_file_name`
: `Points` input file name.

`paths_file_name`
: `Paths` input file name.

`fig_file_name`
: `FIG` output file name.

`-s`
: Output paths as spline curves (open interpolated spline). If this option is not specified, paths are output as polylines.

`-p`
: Output points.

EXAMPLES

`./pth2fig.exe xxx.pnt xxx.pth xxx.fig`
: Generate `xxx.fig` from `xxx.pnt` and `xxx.pth`.

`./pth2fig.exe xxx.pnt xxx.pth xxx.fig -s -p`
: Generate `xxx.fig` from `xxx.pnt` and `xxx.pth`. Output paths as spline curves and also output points.

Contour

`sobel`

Extract `points` from `ppm` image, using Sobel edge detector.

`cluster`

Reduce number of `points` by merging close points.

`link`

Link `points` into `paths` to determine contours.

`simplify`

Simplify `paths` by reducing the number of intermediate `points`.

Sobel

Extract edge pixels from an image using a Sobel edge detector and convert them to `points`.

USAGE

```
sobel [options]
```

OPTIONS

`-i file`

Read digital image in `ppm` format.

`-o file`

Write points in `pnt` format.

`-t threshold`

Sobel magnitude threshold. Magnitudes are normalized between `0.0` and `1.0`, where `1.0` corresponds to the maximum magnitude of all pixels. Only pixels with magnitude greater than or equal to `threshold` are output. If a value out of range is specified then a value is determined automatically. By default a value is determined automatically and it is set to `mean + (1.0 - mean) / 3.0`.

`-n number`

Maximum number of points to output. If the number of points determine by the threshold parameter `t` is greater, then only the `number` points with the highest magnitudes are output. If a negative number is specified then the maximum number is not restricted. The default value is `20,000`.

`-f {0,1}`

Filter pixels using non-maxima suppression edge thinning: only pixels with a magnitude greater than its neighbors in the direction of maximum change are output. The default value is `0` (do not filter).

EXAMPLES

`./sobel.exe -i xxx.ppm -o xxx.pnt`

Extract points, using automatic threshold value, from `xxx.ppm` image and write them to `xxx.pnt`.

`./sobel.exe -i xxx.ppm -o xxx.pnt -t 0.3 -n 10000`

Use a threshold of `0.3`, and output up to `10,000` points.

`./sobel.exe -i xxx.ppm -o xxx.pnt -f`

Filter points using non-maxima suppression edge thinning.

Cluster

Cleanup `points`. Reduces the number of points, by merging close points, to improve the processing time of later stages and reduce the possibility of detecting multiple lines for a single real contour.

USAGE

```
cluster [options]
```

OPTIONS

`-i file`

Read points in `pnt` format.

`-o file`

Write points in `pnt` format.

`-d distance`

Maximum distance. Two points can be merged only if the distance between them is at most `distance`. If the value is negative then the `-r` option is used to determine the maximum distance. The default value is `-1.0`.

`-r range`

Distance range. Two points can be merged only if the distance between them is at most `range` times the minimum distance between two points in the input. The default value is `2.0`. See also the `-d` option.

EXAMPLES

`./cluster.exe -i xxx.pnt -o xxx.c.pnt`

Cluster points from `xxx.pnt` and write to `xxx.c.pnt`.

`./cluster.exe -i xxx.pnt -o xxx.c.pnt -d 0.05`

Allow points with a distance up to `0.05` to be merged (from the `1.0 x 1.0` total possible area).

`./cluster.exe -i xxx.pnt -o xxx.c.pnt -r 3.0`

Allow points up to `3.0` times the original closest pair distance away to be merged.

Link

Link `points` into `paths`.

USAGE

```
link [options]
```

OPTIONS

`-i file`
: Read points in `pnt` format.

`-o file`
: Write paths in `pth` format.

`-d distance`
: Maximum distance. Two points can only be linked if the distance between them is at most `distance`. If the value is negative then the `-r` option is used to determine the maximum distance. The default value is `-1.0`.

`-r range`
: Distance range. Two points can only be linked if the distance between them is at most `range` times the minimum distance between two points. The default value is `3.0`. See also the `-d` option.

`-a angle`
: Maximum orientation difference. Two points can be only be linked if the difference between their orientations and that of the segment between them is at most `angle`. The default value is `35.0` degrees.

EXAMPLES

`./link.exe -i xxx.pnt -o xxx.pth`
: Link points from `xxx.pnt` and write paths to `xxx.pth`.

`./link.exe -i xxx.pnt -o xxx.pth -d 0.05`
: Allow points with a distance up to `0.05` to be linked (from the `1.0 x 1.0` total possible area).

`./link.exe -i xxx.pnt -o xxx.pth -r 5.0 -a 60.0`
: Allow points up to `5.0` times the closest pair distance away, and with a difference in orientation with the segment between them up to `60.0` degrees, to be linked.

Simplify

Simplify `paths` by reducing the number of intermediate `points`.

USAGE

```
simplify [options]
```

OPTIONS

`-i1 file`
: Read points in `pnt` format.

`-i2 file`
: Read paths in `pth` format.

`-o1 file`
: Write points in `pnt` format.

`-o2 file`
: Write paths in `pth` format.

`-d distance`
: Maximum distance allowed between an original path and the corresponding simplified path. If the value is negative then the `-df` option is used to determine the maximum distance. The default value is `-1.0`.

`-df factor`
: Distance factor used to determine maximum distance. If no value is specified for the `-d` option, the maximum distance is equal to the closest pair distance times `factor`. The default value is `0.5`. See also the `-d` option.

`-ml length`
: Minimum path length. Paths with total length shorter than `length`, after simplification, are removed. The default value is `0.0` (nothing is removed).

`-ri {0,1}`
: Remove isolated points not belonging to any path after simplification. The default value is `1` (remove).

EXAMPLES

`./simplify.exe -i1 xxx.pnt -i2 xxx.pth -o1 yyy.pnt -o2 yyy.pth`
: Simplify paths from `xxx.pth` and `xxx.pnt` and write to `yyy.pnt` and `yyy.pth`.

`./simplify.exe -i1 xxx.pnt -i2 xxx.pth -o1 yyy.pnt -o2 yyy.pth -d 0.01`
: Simplify as much as possible without letting the distance between the original and the simplified paths be greater than `0.01`.

`./simplify.exe -i1 xxx.pnt -i2 xxx.pth -o1 yyy.pnt -o2 yyy.pth -df 0.75`

Simplify as much as possible without letting the distance between the original and the simplified paths be greater than `0.75` times the distance between the closest pair of points.

`./simplify.exe -i1 xxx.pnt -i2 xxx.pth -o1 yyy.pnt -o2 yyy.pth -ml 0.1 -ri 0`

Simplify and remove paths shorter than `0.1`, but keep isolated points.

Show

A visualization program for displaying points and contours.

USAGE

`show [options]`

OPTIONS

`-p file`
: Read `points` to display.

`-P file`
: Read `paths` to display. The points for these paths are specified by the `-p` option.

`-p2 file`
: Read secondary set of `points`. Used to visualize two sets of points at the same time.

`-i file`
: Read `ppm` image to display in background.

EXAMPLES

`./show.exe -p xxx.pnt`
: Display points `xxx.pnt`.

`./show.exe -p xxx.pnt -p2 yyy.pnt`
: Display points `xxx.pnt` and `yyy.pnt`.

`./show.exe -p xxx.pnt -P xxx.pth`
: Display points `xxx.pnt` and paths `xxx.pth`.

`./show.exe -p xxx.pnt -P xxx.pth -i xxx.ppm`
: Display points `xxx.pnt` and paths `xxx.pth`, with image `xxx.ppm` in the background.

Graphical User Interface

KEYBOARD

`<Escape>`
: Exit from the visualization program.

`'p'`
: Toggle points: display or hide points.

`'P'`
: Toggle points2: display or hide second set of points.

`'o'`
: Toggle orientations: display or hide point orientations.

`'O'`
: Toggle orientations2: display or hide orientations for second set of points.

`'c'`
: Toggle paths (contours): display or hide paths.

`'b'`
: Toggle box: display or hide bounding box.

`'i'`
: Toggle image: display or hide background image.

`'0' - '9'`
: Set background image transparency: '1' to '9' represent values from 10% to 90% opacity, while '0' is equal to 100% opacity (no transparency).

`<Space>`
: Reset display preferences.

COLORS

Different objects are represented with different colors to make them easy to differentiate.

- Points: red
- Orientations: blue
- Points 2: green
- Orientations 2: green
- Paths: black when image is not displayed, or blue when it is.
- Bounding box: black
- Image: image colors

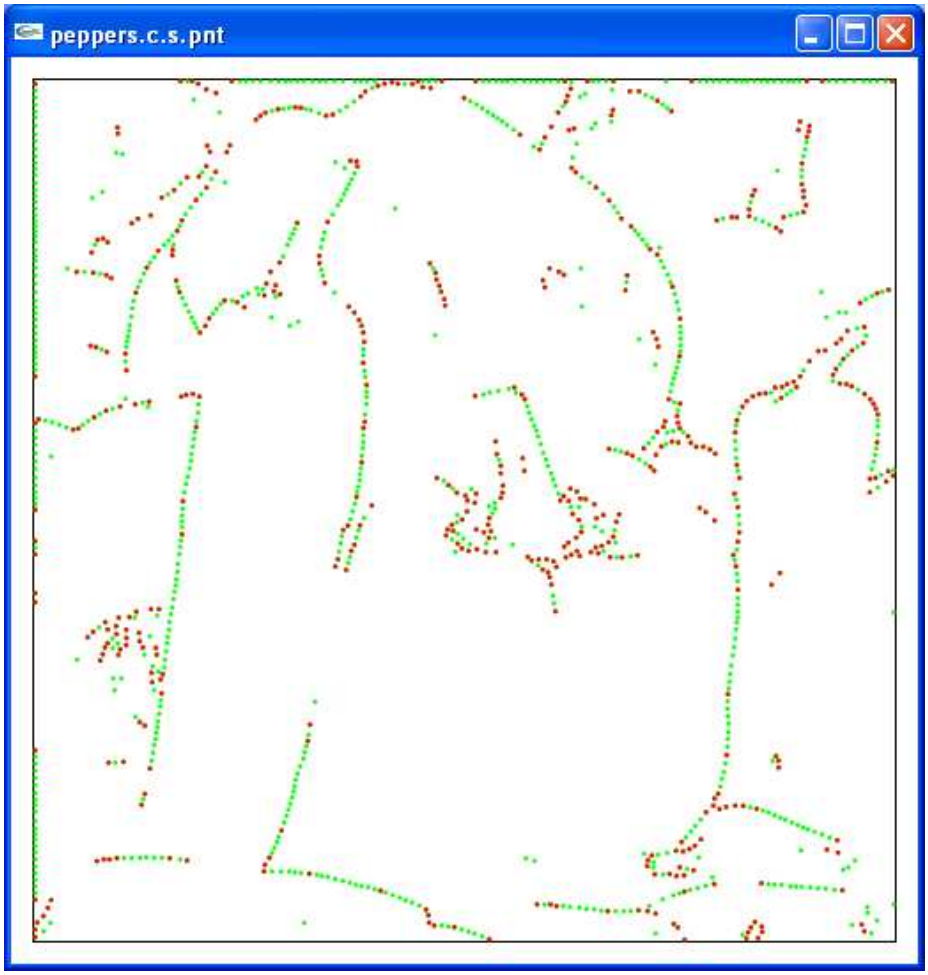


Fig. A.1: Clustered points (green) and points for simplified paths (red).

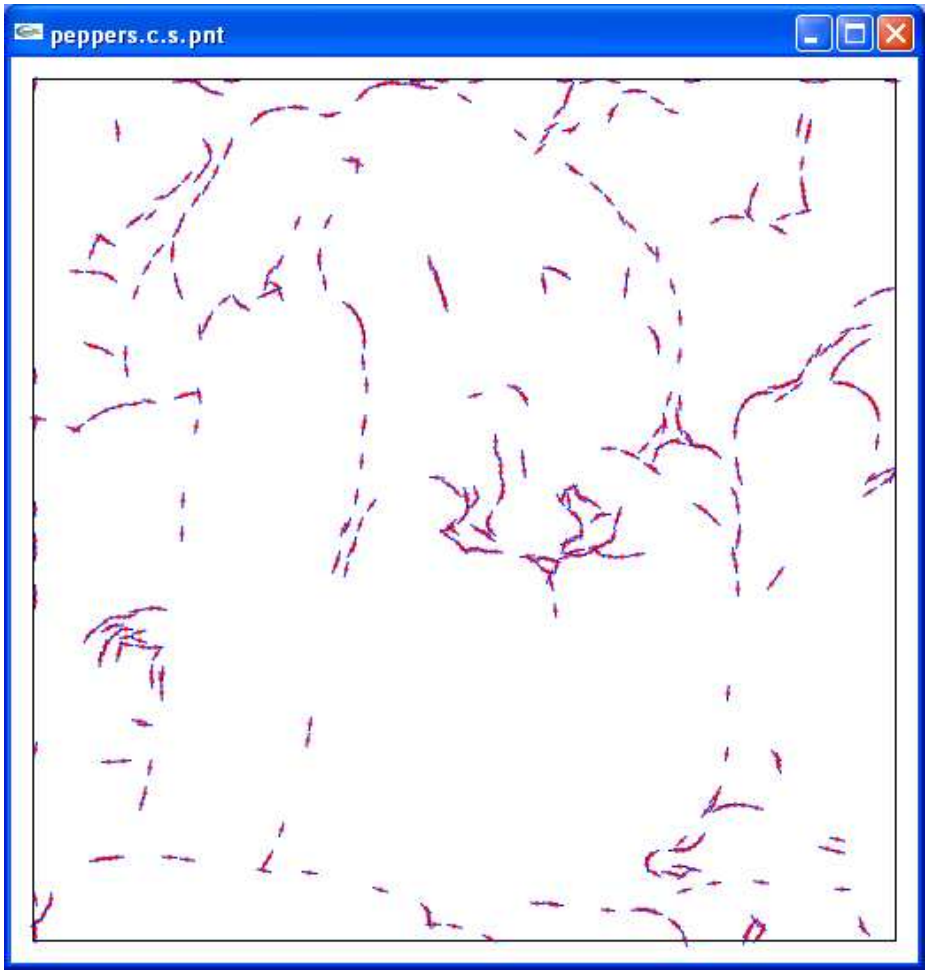


Fig. A.2: Points and orientations displayed.

Fig. A.3: Points, paths, and background image displayed.

Testing

General

`noise`
: Add noise to a `ppm` image or a `pnt` file.

`discretize`
: Discretize `paths` into `points`.

`compare`
: Compare `point` sets.

`stats`
: Average and standard deviation.

Random shapes (test1)

`tcgen`
: Generate test cases with random shapes, from definition file.

`tcapx`
: Generate `point` set approximation of random shapes test case.

Binary images (test2)

`bimcpnt`
: Extract contour `points` from a binary image.

noise

Add noise to a `ppm` image or a `pnt` file.

USAGE

```
noise pnt_input_file pnt_output_file [options]
```

OPTIONS

`pnt_input_file`
: Read `ppm` image or `pnt` file.

`pnt_output_file`
: Write `ppm` image or `pnt` file.

`-ppm`
: Add noise to `ppm` image.

`-pnt`
: Add noise to `pnt` file.

`-s seed`
: Initialize random number generator with `seed` instead of the current time.

`-d density`
: Noise density. Used to determine the number of noise pixels to add to a `ppm` image. The number of points to add (with replacement) is equal to the number of pixels in the image times the `density`. The default value is `0.05`. See also the `-ppm` option.

`-n number`
: Number of noise points to add to a `pnt` file. The default value is `1,000`. See also the `-pnt` option.

EXAMPLES

`./noise.exe xxx.ppm yyy.ppm -ppm -d 0.05`
: Add noise to file `xxx.ppm` and write to `yyy.ppm`. Use noise density of `0.05` to determine the number of random pixels to add.

`./noise.exe xxx.pnt yyy.pnt -pnt -s 123 -n 5000`
: Add noise to file `xxx.pnt` and write to `yyy.pnt`. Initialize random number generator with seed `123` and add `5,000` noise points.

discretize

Discretize `paths` into `points`.

USAGE

```
discretize pnt_input_file pth_input_file pnt_output_file
```

OPTIONS

`pnt_input_file`
: Read `pnt` file.

`pth_input_file`
: Read `pth` file.

`pnt_output_file`
: Write `pnt` file.

EXAMPLES

`./discretize.exe xxx.pnt xxx.pth yyy.pnt`
: Discretize paths defined by `xxx.pnt` and `xxx.pth` into points and write them to `yyy.pnt`. The maximum distance between consecutive points of all discretized paths is guaranteed to be 1 / 1024 (half a pixel for a 512 x 512 image).

compare

Compare two sets of `points` by computing the `Hausdorff distance`[4] between them.

USAGE

```
compare pnt_input_file_1 pnt_input_file_2
```

OPTIONS

`pnt_input_file_1`
: Read first `pnt` file.

`pnt_input_file_2`
: Read second `pnt` file.

EXAMPLES

`./compare.exe xxx.pnt yyy.pnt`
: Computes the Hausdorff distance between the points in `xxx.pnt` and `yyy.pnt`. Outputs a line with the directed Hausdorff distance from `xxx.pnt` to `yyy.pnt`, the directed Hausdorff distance from `yyy.pnt` to `xxx.pnt`, the combined Hausdorff distance between `xxx.pnt` and `yyy.pnt`, and the two file names `xxx.pnt` and `yyy.pnt`, separated by tab characters.

  For example:

```
0.00102118      0.000436232     0.00102118      xxx.pnt      yyy.pnt
```

[4] `http://en.wikipedia.org/wiki/Hausdorff_distance`.

stats

Reads lines with one number from `stdin`, calculates the average and standard deviation of these numbers, and writes two numbers (average, stdev) to `stdout`.

INPUT

```
0.0199178
0.00976164
0.00665529
0.00736282
0.0100466
0.0109013
0.0233229
```

OUTPUT

```
0.0126  0.0064
```

USAGE

```
cat data ./stats.exe > results_file
```

tcgen

Generate test cases with random shapes, from definition file.

USAGE

`tcgen [options]`

OPTIONS

`-i file`
: Read tests definition file.

`-o file`
: Write test cases file.

`-s seed`
: Initialize random number generator with seed instead of the current time.

FILES

1. Test definitions:

   The file can contain several test sets. Each set is indicated by a line starting with ¿¿änd followed by a name and a number of cases for the set. The following lines indicate the shapes in the test cases and their quantities. Shapes can be `LINE` or `ELLIPSE`.

   Example:

   ```
   >> SET01 5
   LINE 1
   >> SET02 5
   ELLIPSE 1
   >> SET03 5
   LINE 3
   ELLIPSE 2
   ```

   This file indicates that there should be three sets of tests. For each set, `5` test cases should be generated. Each test case of set 1 should have `1` line, each test case of set 2 should have `1` ellipse, each test case of set 3 should have `3` lines and `2` ellipses.

2. Test cases:

   Contains information about test cases generated from tests definition file.

   Example:

   ```
   >> SET01 5
   > CASE 1
   LINE 0.505212 0.547695 0.169739 0.228077
   > CASE 2
   LINE 0.956507 0.860088 0.990341 0.55033

   ...
   ```

```
>> SET02 5
> CASE 1
ELLIPSE 0.434674 0.773235 0.0392856 0.603021
> CASE 2
ELLIPSE 0.650386 0.258973 0.944218 0.249173

...
```

This file shows some generated test cases. A `LINE` segment is defined by the extreme points `x1 y1 x2 y2`. An `ELLIPSE` is defined by four parameters `x1 y1 a b`, where `(x1, y1)` is the center and `a` and `b` are the horizontal and vertical semiaxes.

EXAMPLES

`./tcgen.exe -i randomtests.definition -o randomtests.cases.all -s 123`

Initialize random number generator with seed `123`, generate test cases from `randomtests.definition` file, and write them to `randomtests.cases.all`.

tcapx

Generate `point` set approximation of random shapes test case.

USAGE

`tcapx [options]`

OPTIONS

`-i file`
: Read test case file.

`-o file`
: Write `pnt` file.

`-s image_size`
: Used to determine the distance between points of a shape. The program tries to make the the spacing between consecutive points of a shape in the approximation at most `delta = 1 / image_size`. For line segments, the extreme points and the smallest number of equidistant points between them, and separated by a distance no greater than `delta`, are output. For ellipses, a number of points equal to two times the circumference divided by `delta`, and separated by the same angle around the center, are output. The default value is `512`.

FILES

1. Test case:

   Each line has a definition of a geometric shape.

   Example:

   ```
   LINE 0.139844 0.588652 0.0200644 0.126591
   LINE 0.765406 0.310126 0.404557 0.858652
   LINE 0.0689787 0.734898 0.259536 0.0729198
   ELLIPSE 0.0346091 0.873056 0.986836 0.087186
   ELLIPSE 0.49671 0.0709248 0.792153 0.833205
   ```

   This file has three lines and two ellipses. A `LINE` segment is defined by the extreme points `x1 y1 x2 y2`. An `ELLIPSE` is defined by four parameters `x1 y1 a b`, where `(x1, y1)` is the center and `a` and `b` are the horizontal and vertical semiaxes.

EXAMPLES

`./tcapx.exe -i SET03_5.test1.case -o SET03_5.test1.exp.pnt -s 256`
: Generate point approximation of shapes defined in file `SET3_5.test1.case` and write them to file `SET3_5.test1.exp.pnt`. Use image size of `256`.

bimcpnt

Extract contour `points` from a binary image (black and white): white pixels that have an adjacent black pixel above, below, to the left or right.

USAGE

```
bimcpnt ppm_file_name pnt_file_name
```

OPTIONS

`ppm_file_name`
: Read `ppm` file.

`pnt_file_name`
: Write `pnt` file.

EXAMPLES

`./bimcpnt.exe xxx.ppm xxx.pnt`
: Extract edge points from binary image `xxx.ppm` and write them to `xxx.pnt`.

Make

The three platform-specific makefiles `Makefile.Cygwin`, `Makefile.Linux`, and `Makefile.MacOSX` handles the small differences in operating systems. The generic Makefile `Makefile.generic`, which is included by the three platform-specific makefiles, does the bulk of the work. The file `Makefile.generic` consists of several sections:

- **Top Section**: This section defines macros for groups of files and dependencies for executables. In particular,
  - `make all` compiles all programs,
  - `make xxx.clean` removes temporary files for file `xxx.jpg`,
  - `make clean` removes temporary files,
  - `make clobber` removes compiled executables,
  - `make spotless` removes results files.
- **Contour extraction and simplification**: This section provides rules to extract and simplify contours from images. It contains targets to obtain `ppm` images (`.ppm`) from `JPG` images, points (`.pnt`) from `ppm` images, clustered points (`.c.pnt`), paths (`.pth`), and simplified paths (`.s.pth`). For example, for the image `cameraman.jpg`,
  - `make cameraman.ppm` converts it to `ppm` format,
  - `make cameraman.pnt` extracts points from `cameraman.ppm`,
  - `make cameraman.c.pnt` extracts and clusters points,
  - `make cameraman.c.l.pnt` extracts, clusters, and links points,
  - `make cameraman.c.s.pnt` extracts, clusters, links, and simplifies points.
- **File format**: This section provides rules to convert paths to different file formats: `fig`, `eps`, and `pdf`.
- **Display**: This section provides rules to display points or paths using the show program. It contains targets to display `ppm` and `JPG` images, points, and paths. For example, for the image `cameraman.jpg`,
  - `make cameraman.jpg.show` displays `cameraman.jpg`,
  - `make cameraman.ppm.show` displays `cameraman.ppm`,
  - `make cameraman.pnt.show` or `make cameraman.show` display points extracted from `cameraman.ppm`,
  - `make cameraman.c.s.pth.show` displays already extracted contours stored in `make cameraman.c.s.pnt` and `make cameraman.c.s.pth`,
  - `make cameraman.c.show` displays contours obtained after clustering points,
  - `make cameraman.c.l.show` displays contours obtained after clustering and linking points,
  - `make cameraman.c.s.show` displays contours obtained after clustering and linking points, and simplifying paths.
- **Tests**: This section includes all the scripts for automating experiments and analysis. It has several sub-sections: a general one and one for each experiment: random shapes, binary images, and natural images. Note in particular the targets `test1.all`, `test2.all`, and `test3.all` discussed in `Repeat Experiments`.

- **Software Package** This section provides rules for two targets: `all.zip` is an archive of the complete software package; `code.pdf` is a single-pdf printout of the complete source code.
- **Movie**: This section provides rules to process multiple files which can be used to create a movie. The target `dir/*.movie` finds the contours of every image `dir/*.jpg` and for every one it creates a JPG image `dir/?????.movie.jpg` of the contours. These images can be combined to create a video using other softwares (not provided).

A list of parameters that can be specified when running the algorithm with the makefile can be found in `Parameters`.

Parameters

When using the makefile, parameters can be changed for the following programs. Parameters are shown in parenthesis (NAME=DEFAULT_VALUE).

1. jpg2ppm:

   convert to gray scale (J2PG=1, use 0 to keep the original colors).

2. sobel:

   threshold (ST=-1.0), maximum number of points (SN=20000), filter (SF=0).

3. cluster:

   distance (CD=-1.0), range (CR=2.0).

4. link:

   distance (LD=-1.0), range (LR=3.0), angle (LA=35.0).

5. simplify:

   distance (ZD=-1.0), distance factor (ZF=0.5), remove isolated points (ZRI=1), minimum path length (ZML=0.0).

6. pth2fig:

   use splines (FS=0), output points (FP=0).

7. show:

   display background image (IMG=xxx.ppm), display secondary set of points (PNT2=xxx.pnt). By default, no image or secondary set of points are displayed.

8. tcgen:

   seed (SEED=5050).

9. tcapx:

   image size (IMG_SIZE=512).

10. noise:

    noise density (ND=0.05), number of noise points (NN=1000), seed (SEED=5050).

The following examples change many of the parameter values (defaults are used if no value is specified):

- `make cameraman.c.s.pnt SN=-1 ST=0.3 CR=2.5 LR=4.0 ZML=0.02` extracts contours from `cameraman.jpg` using a sobel threshold of `0.3` with no limit on the number of points (`SN=-1`), a clustering range of `2.5`, a linking range of `4.0`, and removing paths shorter than `0.02` after simplification.
- `make cameraman.c.s.show IMG=cameraman.ppm PNT2=cameraman.pnt` displays extracted contours from `cameraman.jpg` and also the original image `cameraman.ppm` and the original point set `cameraman.pnt`.

Screenshots

The following sequence of screenshots illustrates the steps of the algorithm.

Each screenshot is followed by

- the command used to generate it,
- additional details of interest.

Original image

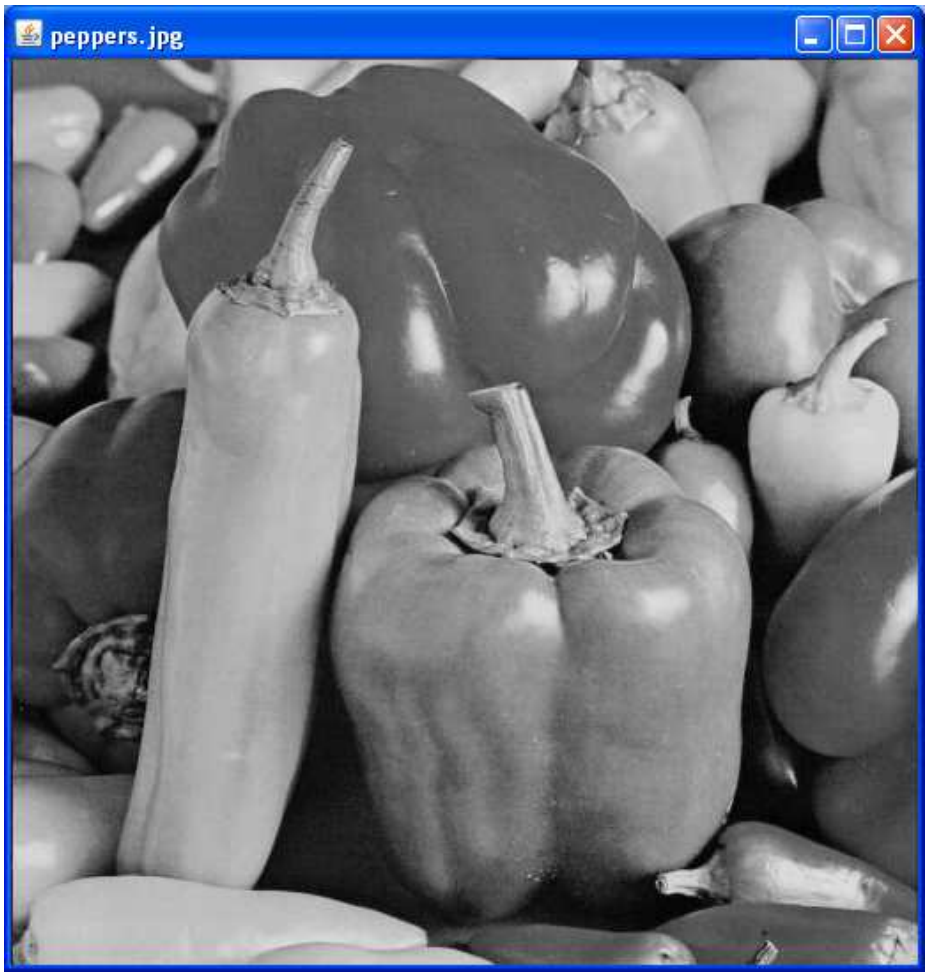


Fig. A.4: Command: `./make peppers.jpg.show`. Image size: `512 x 512`

Extracted points

Fig. A.5: Command: `./make peppers.pnt.show ST=0.3 IMG=peppers.ppm`. 7500 points

Clustered points

Fig. A.6: Command: `./make peppers.c.show ST=0.3 IMG=peppers.ppm`. `1721` points

Linked points

Fig. A.7: Command: `./make peppers.c.l.show ST=0.3 IMG=peppers.ppm`. `167` paths and `1721` points

Simplified paths

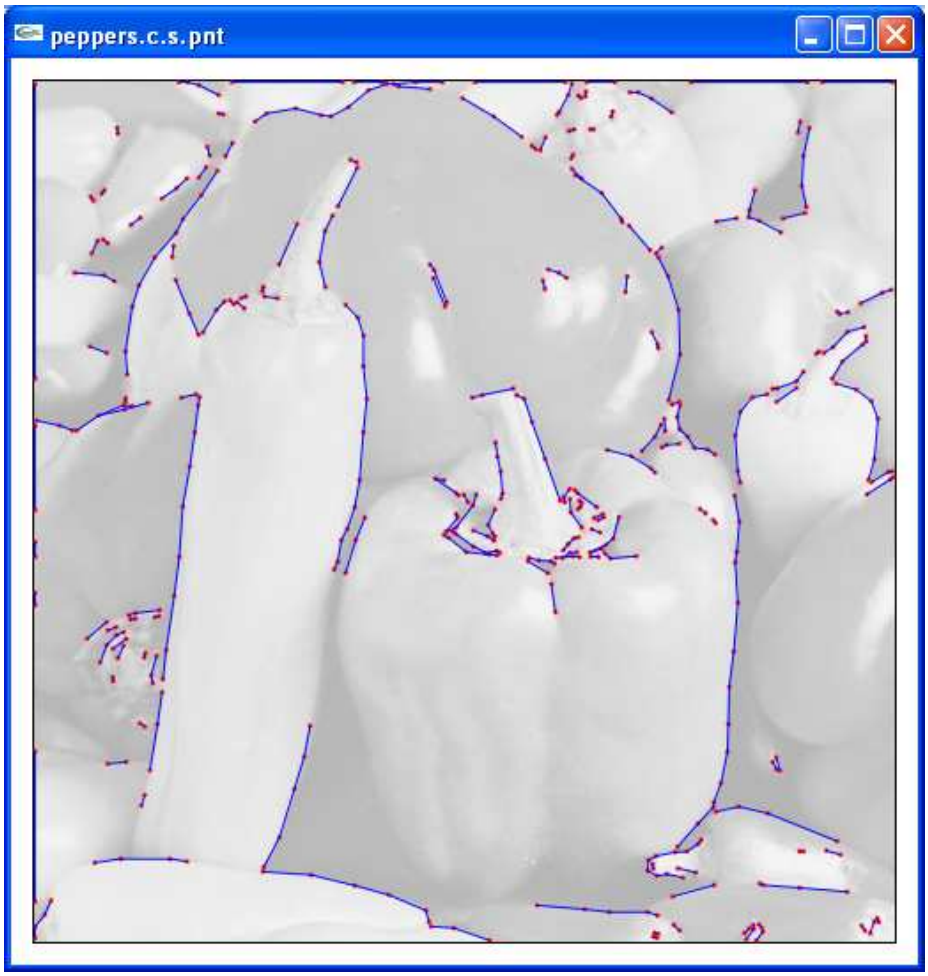


Fig. A.8: Command: `./make peppers.c.s.show ST=0.3 IMG=peppers.ppm`. `167` paths and `498` points

Final result

Fig. A.9: Command: `./make peppers.c.s.show ST=0.3 IMG=peppers.ppm`. `167` paths and `498` points

Download and Install Software

Download the file all.zip and unzip it. There are three platform-specific Makefiles:

- `Makefile.Cygwin`
- `Makefile.Linux`
- `Makefile.MacOSX`

Copy the file `Makefile.[Your Platform]` to a new file `Makefile`. For example, if you are using an Apple, copy `Makefile.MacOSX` to `Makefile`. Or, if you are using a Windows PC, install Cygwin[5] (with at least these additional packages), then copy `Makefile.Cygwin` to `Makefile`.

Install a recent version of the Java Development Kit[6].

To verify that everything works correctly, run the following commands:

1. `make all`
2. `make cameraman.c.s.show`

The first command `make all` compiles the programs.

The second command `make cameraman.c.s.show` finds and displays the contours of the image `cameraman.jpg`. Press the escape key to exit from the visualization program.

The output of the commands should be similar to the following two screen shots:

[5] `http://cygwin.com/`.
[6] `http://java.sun.com/`.

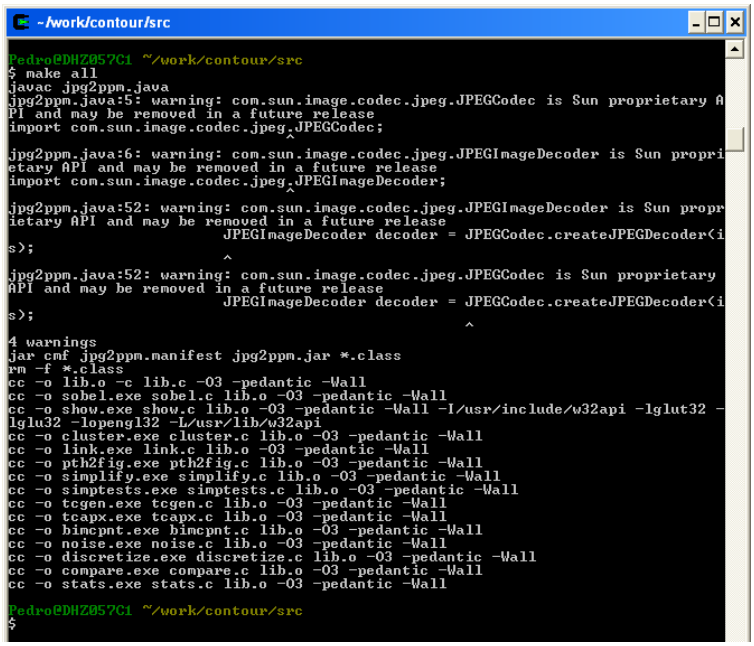


Fig. A.10: Command window.

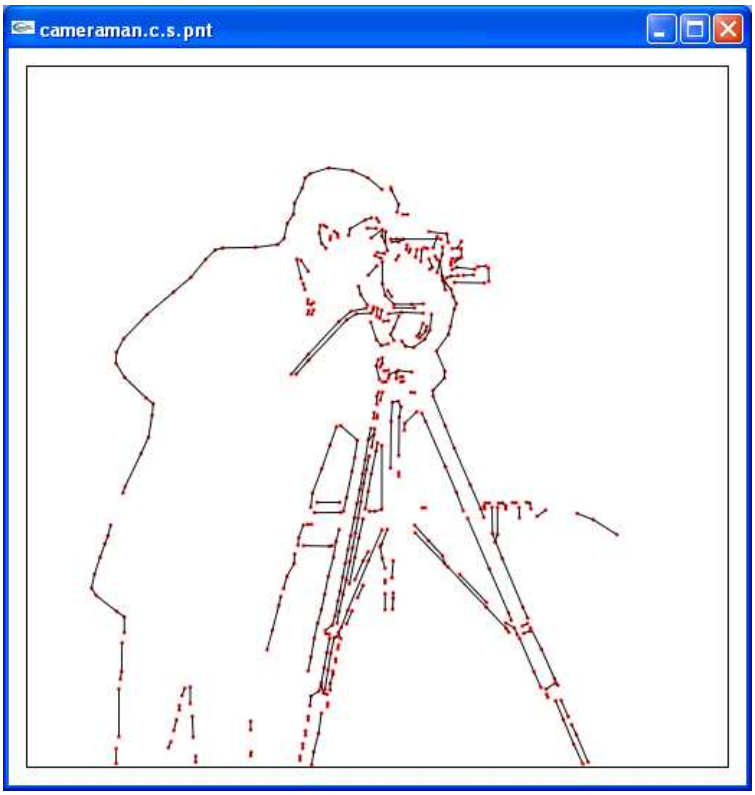


Fig. A.11: Visualization program.

Required Cygwin Packages

- archive
  - unzip
  - zip
- devel
  - gcc-core
  - make
- graphics
  - freeglut
  - ghostscript
  - gnuplot
  - imagemagick
  - opengl
  - transfig
- text
  - enscript
  - tetex

Repeat Experiments

Random Shapes (accuracy)

Data set

Test cases are automatically generated from the `randomtests.definition` file. To use a different file, specify it when running the make command using the parameter `TEST1_DEF=xxx.definition`.

make test1.all

This command runs the algorithms on a set of pictures with randomly generated shapes (line segments and ellipses). These pictures are generated from the `randomtests.definition` file. The results are compared with an approximation of the expected results.

In our case, the running time was approximately 6 hours, and results after running the experiments took 1.10 GB of disk space.

Binary Images (accuracy)

Data set

Download the file `mpeg7.zip` (22.3 MB) and unzip it inside the programs directory. Optionally, a smaller file `mpeg7-gif.zip` (3.16 MB) can be downloaded; it contains all test images in `gif` format, which can be converted to `jpg` format using the provided Matlab file `gif2jpg.m`. By default, the images in the `mpeg7` directory are used; to use a different directory, specify it when running the make command using the parameter `TEST2_DAT=other_dir`.

make test2.all

Run the command `make test2.all`. This command runs the algorithms on a set of binary (black and white) images. The results are compared with an approximation of the expected results.

In our case, the running time was approximately 13 hours, and the results after running the experiments took 12.4 GB of disk space.

Natural Images (compression)

Data set

All test images are provided with the software.

make test3.all

Run the command `test3.all` to process a set of natural images. This command extracts contours from each image and determines the number of points after edge detection, clustering, and simplification, to evaluate the level of compression attained by our method.

Results of Our Experiments

- `test1_results.zip` contains results for random shapes tests. There are three results files `test1_n_m.txt`, `test1_n_m.cnt`, and `test1_n_m.sts` for each set of parameters; with `n` indicating the amount of noise (the number of noise points is equal to `n * 1000`), and `m` indicating the minimum length of the contours that are kept (all paths with length smaller than `m / 100` are removed). The `test1_n_m.txt` files contain the Hausdorff distances between each result and the expected result; the `test1_n_m.cnt` files contain the number of paths extracted from each file; and the `test1_n_m.sts` files contain the averages and standard deviations of the Hausdorff distances and number of extracted paths.

  An example `sts` file:

  ```
  Total pictures:         300
  With paths detected:    299
  With no paths detected: 1
  Hausdorff distances:
  0.0012  0.0002
  0.0148  0.0323
  0.0148  0.0322
  Average number of paths:
  12.0133 16.2124
  ```

  First it shows that contours where extracted from 300 pictures with contours found in 299 of them. Then, for the pictures for which contours were found, the average and standard deviation of the Hausdorff distances: directed from obtained result to answer, directed from answer to obtained result, and combined. Finally, for the pictures for which contours were found, the average number of paths and standard deviation.

- `test2_results.zip` contains results for binary images tests. There are three results files `test2_n_m.txt`, `test2_n_m.cnt`, and `test2_n_m.sts` for each set of parameters; with `n` indicating the amount of noise (noise density is equal to `n / 100`), and `m` indicating the minimum length of the contours that are kept (all paths with length smaller than `m / 100` are removed). The `test2_n_m.txt` files contain the Hausdorff distances between each result and the expected result; the `test2_n_m.cnt` files contain the number of paths extracted from each image; and the `test1_n_m.sts` files contain the averages and standard deviations of the Hausdorff distances and number of extracted paths.

  The files are similar to the ones from the random shapes tests (test1).

- `test3_results.zip` contains results for natural images tests.

  An example `sts` file:

```
Sobel points  : 26266.5556  14427.9875
Cluster points: 5383.8889   3430.7101
Cluster comp. : 5.1177      0.7706
Simp. points  : 2099.1111   1745.1654
Simp. comp.   : 15.0108     5.1086
Simp. paths   : 728.6667    728.5450
```

  It shows the average and standard deviation of: the number of points extracted by the Sobel edge detector, the number of points after clustering, the number of times reduction after clustering (from the number of points extracted by the edge detector), the number of points after simplification, the number of times reduction after simplification (from the number of points extracted by the edge detector), the number of paths.

Appendix B

Contour Extraction and Visualization Source Code

B.1 lib.h

```
/*
 *  lib.h - library header for edge detector
 *
 *  Pedro Tejada and Minghui Jiang
 *  Fri Jun 12 18:24:14 MDT 2009
 */

#define RAND(X) (rand() / (((double)RAND_MAX + 1) / X))

/*
 *  Point
 */

/* x, y, a are normalized between 0.0 and 1.0 */
typedef struct {
    double x;   /* x coordinate */
    double y;   /* y coordinate */
    double a;   /* orientation angle */
    double w;   /* weight */
} s_point;

typedef struct {
    s_point *points;    /* points */
    int size;           /* array size */
    int n;              /* number of points */
} s_point_array;

s_point *new_point(double x, double y, double a, double w);
void print_point(s_point *p);
void print_points(s_point *p, int n_points);

s_point_array *new_point_array(int n);
void extend_point_array1(s_point_array *pnt_array, s_point *point);
void extend_point_array(s_point_array *pnt_array, s_point *points, int n_points);

/*
 * file format: the first line specifies the number of points;
 *  each following line specifies a point: x y a w
 */
s_point *input_points(char *file_name, int *n_points);
void output_points(char *file_name, s_point *points, int n_points);

/*
```

```
 *  Path
 */

typedef struct {
    int *indices;   /* indices to points in path */
    int size;       /* indices array size */
    int length;     /* number of points in path */
} s_path;

s_path *new_path(int length);
void reverse_path(s_path *p);
void extend_path(s_path *p, int i_point);
void print_path(s_path *p);
void print_paths(s_path *p, int n_paths);

/*
 * file format: the first line specifies the number of paths and points;
 *  each following line specifies a path: n i_1 i_2 ... i_n
 *  The first number is the number of points in the paths;
 *  the following numbers are the indices of the points.
 */
s_path **input_paths(char *file_name, int *n_paths, int *n_points);
void output_paths(char *file_name, s_path **paths, int n_paths, int n_points);

/*
 *  ppm image
 */

#define MAX_VALUE 255

typedef struct {
    unsigned char r;
    unsigned char g;
    unsigned char b;
} s_pixel;

typedef struct {
    s_pixel **p;    /* pixel data */
    int w;          /* width */
    int h;          /* height */
} s_image;

s_image *new_image(int w, int h);
s_image *load_ppm(char *file_name);
void save_ppm(s_image *img, char *file_name);
double get_point_x(int w, int h, int x);
double get_point_y(int w, int h, int y);
double get_pixel_x(int w, int h, double x);
double get_pixel_y(int w, int h, double y);

/*
 *  Geometry functions
 */

double dist(s_point *p1, s_point *p2);
double dist2(s_point *p1, s_point *p2); /* squared distance */
```

```
100 double dist2seg(s_point *p1, s_point *q1, s_point *q2);
    double angle_diff(double a1, double a2); /* orientation difference */
    double vector_angle(s_point *p1, s_point *p2, s_point *p3);
```

B.2 lib.c

```
/*
 *  lib.c - library implementation for edge detector
 *
 *  Pedro Tejada and Minghui Jiang
 *  Wed Jun 24 00:48:44 MDT 2009
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

/*
 *  Point
 */

s_point *new_point(double x, double y, double a, double w) {
    s_point *p;

    if ((p = malloc(sizeof(s_point))) == NULL) {
        fprintf(stderr, "new_point: malloc failed\n");
        exit(1);
    }
    p->x = x;
    p->y = y;
    p->a = a;
    p->w = w;
    return p;
}

void print_point(s_point *p) {
    printf("(%g %g %g %g)\n", p->x, p->y, p->a, p->w);
}

void print_points(s_point *p, int n_points) {
    int i;

    for (i = 0; i < n_points; i++)
        print_point(&p[i]);
}

s_point *input_points(char *file_name, int *n_points) {
    FILE *file;
    s_point *points;
    int n_points_;
    int i;

    if ((file = fopen(file_name, "r")) == NULL) {
        fprintf(stderr, "input_points: fopen(%s) failed\n", file_name);
        exit(1);
    }
    if (fscanf(file, "%d\n", &n_points_) != 1) {
        fprintf(stderr, "input_points: %s has wrong format\n", file_name);
        exit(1);
```

```
    }
    if ((points = malloc(sizeof(s_point) * n_points_)) == NULL) {
        fprintf(stderr, "input_points: malloc failed\n");
        exit(1);
    }
    for (i = 0; i < n_points_; i++) {
        s_point *p = &points[i];

        if (fscanf(file, "%lf%lf%lf%lf\n", &p->x, &p->y, &p->a, &p->w) != 4) {
            fprintf(stderr, "input_points: points[%d] has wrong format\n", i);
            exit(1);
        }
    }
    fclose(file);
    *n_points = n_points_;
    return points;
}

void output_points(char *file_name, s_point *points, int n_points) {
    FILE *file;
    int i;

    if ((file = fopen(file_name, "w")) == NULL) {
        fprintf(stderr, "output_points: fopen(%s) failed\n", file_name);
        exit(1);
    }
    fprintf(file, "%d\n", n_points);
    for (i = 0; i < n_points; i++) {
        s_point *p = &points[i];

        fprintf(file, "%g %g %g %g\n", p->x, p->y, p->a, p->w);
    }
    fclose(file);
}

s_point_array *new_point_array(int n) {
    s_point_array *p;

    if ((p = malloc(sizeof(s_point_array))) == NULL) {
        fprintf(stderr, "new_point_array: malloc failed\n");
        exit(1);
    }
    p->n = n;
    p->size = 2;
    while (p->size < p->n)
        p->size *= 2;
    if ((p->points = malloc(p->size * sizeof(s_point))) == NULL) {
        fprintf(stderr, "new_point_array: malloc failed\n");
        exit(1);
    }
    return p;
}

void extend_point_array1(s_point_array *pnt_array, s_point *point) {
    if (pnt_array->n == pnt_array->size) { /* no space available */
        s_point *t_points;
```

```
        int i;

        pnt_array->size *= 2;
        if ((t_points = malloc(pnt_array->size * sizeof(s_point))) == NULL) {
            fprintf(stderr, "extend_point_array1: malloc failed\n");
            exit(1);
        }
        for (i = 0; i < pnt_array->n; i++)
            t_points[i] = pnt_array->points[i];
        free(pnt_array->points);
        pnt_array->points = t_points;
    }
    pnt_array->points[pnt_array->n] = *point;
    pnt_array->n++;
}

void extend_point_array(s_point_array *pnt_array,
        s_point *points, int n_points) {
    int i;

    for (i = 0; i < n_points; i++)
        extend_point_array1(pnt_array, &points[i]);
}

/*
 *  Path
 */

s_path *new_path(int length) {
    s_path *p;

    if ((p = malloc(sizeof(s_path))) == NULL) {
        fprintf(stderr, "new_path: malloc failed\n");
        exit(1);
    }
    p->length = length;
    p->size = 2;
    while (p->size < p->length)
        p->size *= 2;
    if ((p->indices = malloc(p->size * sizeof(int))) == NULL) {
        fprintf(stderr, "new_path: malloc failed\n");
        exit(1);
    }
    return p;
}

void reverse_path(s_path *p) {
    int i;

    for (i = 0; i < p->length / 2; i++) {
        int t = p->indices[i];

        p->indices[i] = p->indices[p->length - 1 - i];
        p->indices[p->length - 1 - i] = t;
    }
}
```

```
void extend_path(s_path *p, int i_point) {
    if (p->length == p->size) { /* no space available */
        int *t_indices;
        int i;

        p->size *= 2;
        if ((t_indices = malloc(p->size * sizeof(int))) == NULL) {
            fprintf(stderr, "extend_path: malloc failed\n");
            exit(1);
        }
        for (i = 0; i < p->length; i++)
            t_indices[i] = p->indices[i];
        free(p->indices);
        p->indices = t_indices;
    }
    p->indices[p->length] = i_point;
    p->length++;
}

void print_path(s_path *p) {
    int i;

    for (i = 0; i < p->length; i++)
        printf("%d ", p->indices[i]);
    printf("\n");
}

void print_paths(s_path *p, int n_paths) {
    int i;

    for (i = 0; i < n_paths; i++)
        print_path(&p[i]);
}

s_path **input_paths(char *file_name, int *n_paths, int *n_points) {
    FILE *file;
    s_path **paths;
    int n_paths_, n_points_;
    int i, j;

    if ((file = fopen(file_name, "r")) == NULL) {
        fprintf(stderr, "input_paths: fopen(%s) failed\n", file_name);
        exit(1);
    }
    if (fscanf(file, "%d %d\n", &n_paths_, &n_points_) != 2) {
        fprintf(stderr, "input_paths: %s has wrong format\n", file_name);
        exit(1);
    }
    if ((paths = malloc(sizeof(s_path *) * n_paths_)) == NULL) {
        fprintf(stderr, "input_paths: malloc failed\n");
        exit(1);
    }
    for (i = 0; i < n_paths_; i++) {
        s_path *p;
        int length;
```

```
            if (fscanf(file, "%d", &length) != 1) {
                fprintf(stderr, "input_paths: paths[%d] has wrong format\n", i);
                exit(1);
            }
            p = new_path(length);
            for (j = 0; j < length; j++)
                if (fscanf(file, "%d", &p->indices[j]) != 1) {
                    fprintf(stderr, "input_paths: "
                        "paths[%d] points[%d] has wrong format\n", i, j);
                    exit(1);
                }
            paths[i] = p;
        }
        fclose(file);
        *n_paths = n_paths_;
        *n_points = n_points_;
        return paths;
    }

    void output_paths(char *file_name, s_path **paths, int n_paths, int n_points) {
        FILE *file;
        int i, j;

        if ((file = fopen(file_name, "w")) == NULL) {
            fprintf(stderr, "output_paths: fopen(%s) failed\n", file_name);
            exit(1);
        }
        fprintf(file, "%d %d\n", n_paths, n_points);
        for (i = 0; i < n_paths; i++) {
            s_path *p = paths[i];

            fprintf(file, "%d", p->length);
            for (j = 0; j < p->length; j++)
                fprintf(file, " %d", p->indices[j]);
            fprintf(file, "\n");
        }
        fclose(file);
    }

    /*
     *  ppm image
     */

    s_image *new_image(int w, int h) {
        s_image *img;
        s_pixel *pixels;
        int i, j;

        if ((img = malloc(sizeof(s_image) + sizeof(s_pixel *) * w
                + sizeof(s_pixel) * (w * h))) == NULL) {
            fprintf(stderr, "new_image: malloc failed\n");
            exit(1);
        }
        img->w = w;
        img->h = h;
```

```
    img->p = (s_pixel **) (img + 1);
    pixels = (s_pixel *) (img->p + w);
    for (i = 0; i < w; i++) {
        img->p[i] = (s_pixel *) (pixels + i * h);

        /* set all pixels to white */
        for (j = 0; j < h; j++) {
            img->p[i][j].r = MAX_VALUE;
            img->p[i][j].g = MAX_VALUE;
            img->p[i][j].b = MAX_VALUE;
        }
    }
    return img;
}

s_image *load_ppm(char *file_name) {
    FILE *file;
    s_image *img = NULL;
    int x, y, w, h, m;
    char s[3];
    char c;

    if ((file = fopen(file_name, "r")) == NULL) {
        fprintf(stderr, "load_ppm: fopen(%s) failed\n", file_name);
        exit(1);
    }

    /* read signature */
    if ((fscanf(file, "%2[^\n]\n", s) != 1) || strcmp(s, "P3")){
        fprintf(stderr, "load_ppm: %s has wrong format\n", file_name);
        exit(1);
    }
    while ((c = fgetc(file)) == '#') {  /* skip comments */
        fscanf(file, "%*[^\n]\n");
    }
    ungetc(c, file);
    if (fscanf(file, "%d %d %d\n", &w, &h, &m) != 3) {
        fprintf(stderr, "load_ppm: %s has wrong format\n", file_name);
        exit(1);
    }
    img = new_image(w, h);
    for (y = 0; y < img->h; y++)
        for (x = 0; x < img->w; x++) {
            int r, g, b;
            fscanf(file, "%d %d %d", &r, &g, &b);
            img->p[x][y].r = r;
            img->p[x][y].g = g;
            img->p[x][y].b = b;
        }
    return img;
}

void save_ppm(s_image *img, char *file_name) {
    FILE *file;
    int x, y;
```

```
    if ((file = fopen(file_name, "w")) == NULL) {
        fprintf(stderr, "save_ppm: fopen(%s) failed\n", file_name);
        exit(1);
    }
    fprintf(file, "P3\n");
    fprintf(file, "%d %d %d\n", img->w, img->h, MAX_VALUE);
    for (y = 0; y < img->h; y++)
        for (x = 0; x < img->w; x++) {
            int r, g, b;
            r = img->p[x][y].r;
            g = img->p[x][y].g;
            b = img->p[x][y].b;
            fprintf(file, "%d %d %d\n", r, g, b);
        }
    fclose(file);
}

double get_point_x(int w, int h, int x) {
    double wh = (w > h) ? w : h;
    double x_ = x + 0.5; /* point at center of pixel */

    x_ = x_ / wh + 0.5 * (wh - w) / wh; /* center in 1.0 x 1.0 square */
    return x_;
}

double get_point_y(int w, int h, int y) {
    double wh = (w > h) ? w : h;
    double y_ = y + 0.5; /* point at center of pixel */

    y_ = y_ / wh + 0.5 * (wh - h) / wh; /* center in 1.0 x 1.0 square */
    return 1.0 - y_; /* invert y axis */
}

double get_pixel_x(int w, int h, double x) {
    double wh = (w > h) ? w : h;
    double pixel_size = 1.0 / wh;
    int x_;

    x -= 0.5 * (wh - w) / wh;
    x_ = (int) floor(x / pixel_size);
    if (x_ == w)
        x_--;
    return x_;
}

double get_pixel_y(int w, int h, double y) {
    double wh = (w > h) ? w : h;
    double pixel_size = 1.0 / wh;
    int y_;

    y -= 0.5 * (wh - h) / wh;
    y_ = (int) floor(y / pixel_size);
    y_ = h - y_;
    if (y_ == h)
        y_--;
    return y_;
```

```
}

/*
 *  Geometry functions
 */

double dist(s_point *p1, s_point *p2) {
    return sqrt(dist2(p1, p2));
}

double dist2(s_point *p1, s_point *p2) {
    double dx = p1->x - p2->x;
    double dy = p1->y - p2->y;

    return dx * dx + dy * dy;
}

double dist2seg(s_point *p, s_point *q1, s_point *q2) {
    double ds, dl, d1, d2;
    double x0 = p->x;
    double y0 = p->y;
    double x1 = q1->x;
    double y1 = q1->y;
    double x2 = q2->x;
    double y2 = q2->y;

    /* segment length */
    ds = dist(q1, q2);
    if (ds == 0.0)
        return dist(p, q1);

    /* distance to line */
    /* http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html */
    dl = fabs((x2 - x1) * (y1 - y0) - (x1 - x0) * (y2 - y1)) / ds;

    /* distance to end points */
    d1 = dist(p, q1);
    d2 = dist(p, q2);

    if (d1 * d1 <= ds * ds + dl * dl
            && d2 * d2 <= ds * ds + dl * dl)
        return dl;
    else
        return fmin(d1, d2);
}

/* a1 and a1 are in radians */
double angle_diff(double a1, double a2) {
    double a;

    a1 = fmod(a1, M_PI);
    if (a1 < 0.0)
        a1 += M_PI;
    a2 = fmod(a2, M_PI);
    if (a2 < 0.0)
        a2 += M_PI;
```

```
        a = fabs(a1 - a2);
        if (a > M_PI / 2.0)
            a = M_PI - a;
        return a;
    }

    double vector_angle(s_point *p1, s_point *p2, s_point *p3) {
        double x1 = p2->x - p1->x;
        double y1 = p2->y - p1->y;
        double x2 = p3->x - p2->x;
        double y2 = p3->y - p2->y;
        double n1 = sqrt(x1 * x1 + y1 * y1);
        double n2 = sqrt(x2 * x2 + y2 * y2);

        if (n1 == 0.0 || n2 == 0.0)
            return 0.0;
        return acos((x1 * x2 + y1 * y2) / (n1 * n2));
    }
```

B.3 sobel.c

```
/*
 *  sobel.c - sobel edge detector
 *
 *  Pedro Tejada and Minghui Jiang
 *  Tue Jul  7 11:44:39 MDT 2009
 */

#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

double T = -1.0;/* Minimum required magnitude (fraction of max value, -t) */
int N = 20000;  /* Maximum number of points to output */
int filter = 0; /* Use nonmaxima suppression */

double **gm;    /* magnitude */
double **ga;    /* angle */
s_point ***P;   /* output point matrix */

void allocate_arrays(int w, int h) {
    int i, j;

    if ((gm = malloc(sizeof(double*) * w)) == NULL) {
        fprintf(stderr, "allocate_arrays: malloc failed\n");
        exit(1);
    }
    if ((ga = malloc(sizeof(double*) * w)) == NULL) {
        fprintf(stderr, "allocate_arrays: malloc failed\n");
        exit(1);
    }
    if ((P = malloc(sizeof(s_point**) * w)) == NULL) {
        fprintf(stderr, "allocate_arrays: malloc failed\n");
        exit(1);
    }
    for (i = 0; i < w; i++) {
        if ((gm[i] = malloc(sizeof(double) * h)) == NULL) {
            fprintf(stderr, "allocate_arrays: malloc failed\n");
            exit(1);
        }
        if ((ga[i] = malloc(sizeof(double) * h)) == NULL) {
            fprintf(stderr, "allocate_arrays: malloc failed\n");
            exit(1);
        }
        if ((P[i] = malloc(sizeof(s_point*) * h)) == NULL) {
            fprintf(stderr, "allocate_arrays: malloc failed\n");
            exit(1);
        }
        for (j = 0; j < h; j++) {
            gm[i][j] = 0.0;
            ga[i][j] = 0.0;
            P[i][j] = NULL;
        }
```

```
    }
}

/* convolution kernel: CM and CN are odd numbers */
#define CM  3
#define CN  3

double hx[CM][CN] = {{-1.0, 0.0, 1.0},
                     {-2.0, 0.0, 2.0},
                     {-1.0, 0.0, 1.0}};
double hy[CM][CN] = {{-1.0, -2.0, -1.0},
                     { 0.0,  0.0,  0.0},
                     { 1.0,  2.0,  1.0}};

void convolution(s_image *img, int x, int y) {
    double gx_, gy_;
    int i, j;

    gx_ = gy_ = 0.0;
    for (i = 0; i < CM; i++)
        for (j = 0; j < CN; j++) {
            int x_ = x + (CM / 2 - j);
            int y_ = y + (CN / 2 - i);
            int rgb = 0; /* zero padding */

            /* ignore border */
            if (x_ >= 0 && x_ < img->w && y_ >= 0 && y_ < img->h)
                rgb = img->p[x_][y_].r;
            gx_ += hx[i][j] * rgb;
            gy_ -= hy[i][j] * rgb; /* y-axis is inverted */
        }
    gm[x][y] = sqrt(gx_ * gx_ + gy_ * gy_);
    ga[x][y] = atan2(gy_, gx_); /* [-pi, pi] */
    if (ga[x][y] < 0.0)         /* [ 0, 2pi] */
        ga[x][y] += 2 * M_PI;
}

int compare_w(s_point *p1, s_point *p2) {
    if (p1->w < p2->w)
        return 1;
    else if (p1->w > p2->w)
        return -1;
    else
        return 0;
}

void print_help_and_exit(int argc, char *argv[]) {
    fprintf(stderr, "usage:  %s [options]\n" "options:\n"
            "\t-i\t<input_filename>\n"
            "\t-o\t<output_filename>\n"
            "\t-t\t<threshold>\n"
            "\t-n\t<number>\n"
            "\t-f\t<filter>\n",
        argv[0]);
    exit(1);
}
```

```
int main(int argc, char *argv[]) {
    char *input_filename = NULL;
    char *output_filename = NULL;
    s_image *img;
    s_point *points;
    int n_points;
    double w, w_;
    double mean;
    int i, j;

    for (i = 1; i < argc; i++)
        if (!strcmp(argv[i], "-i")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                input_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-o")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                output_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-t")) {
            if (i + 1 < argc) /* allow negative value */
                T = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-n")) {
            if (i + 1 < argc) /* allow negative value */
                N = atoi(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-f")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                filter = atoi(argv[++i]);
            else
                print_help_and_exit(argc, argv);
            if (filter != 0 && filter != 1)
                print_help_and_exit(argc, argv);
        } else
            print_help_and_exit(argc, argv);
    if (!input_filename || !output_filename)
        print_help_and_exit(argc, argv);

    img = load_ppm(input_filename);
    printf("image width: %d\n", img->w);
    printf("image height: %d\n", img->h);
    allocate_arrays(img->w, img->h);
    w_ = 0.0;
    for (i = 1; i < img->w - 1; i++)
        for (j = 1; j < img->h - 1; j++) {
            convolution(img, i, j);
            if ((w = gm[i][j]) > w_)
                w_ = w;
        }
```

```
    if (T < 0.0 || T > 1.0) {
        mean = 0.0;
        for (i = 1; i < img->w - 1; i++)
            for (j = 1; j < img->h - 1; j++)
                mean += gm[i][j];
        mean /= (double) (img->w * img->h); /* average */
        mean /= w_; /* normalize to [0.0, 1.0] */
        T = mean + (1.0 - mean) / 3.0; /* make sure T <= 1.0 */
    }
    printf("threshold: %g\n", T);

    if ((points = malloc(sizeof(s_point) * img->w * img->h)) == NULL) {
        fprintf(stderr, "main: malloc failed\n");
        exit(1);
    }
    n_points = 0;
    for (i = 1; i < img->w - 1; i++)
        for (j = 1; j < img->h - 1; j++) {
            double a = ga[i][j];
            double w = gm[i][j];
            s_point *p;

            if ((a = fmod(a, M_PI) / M_PI) < 0.0)
                a += 1.0;
            a = fmod(a + 0.5, 1.0); /* edge is orthogonal to gradient */
            w /= w_;

            if (w >= T) {
                if (filter) {

                    /* determine discrete direction */
                    int dx[4] = {0, 1};
                    int dy[4] = {1, 0};
                    int di = 0;     /* horizontal edge */

                    if (a > 1.0 / 4.0 && a <= 3.0 / 4.0)
                        di = 1;     /* vertical edge */

                    if (gm[i + dx[di]][j + dy[di]] > gm[i][j] + DBL_EPSILON * 100
                            || gm[i - dx[di]][j - dy[di]] > gm[i][j] + DBL_EPSILON * 100)
                        continue;
                }
                p = &points[n_points++];
                p->x = get_point_x(img->w, img->h, i);
                p->y = get_point_y(img->w, img->h, j);
                p->a = a;
                p->w = w;
            }
        }

    qsort(points, n_points, sizeof(s_point),
            (int (*)(const void *, const void *)) compare_w);
    if (N >= 0 && n_points > N)
        n_points = N;
    printf("%d points\n", n_points);
    output_points(output_filename, points, n_points);
```

```
    return 0;
}
```

B.4 cluster.c

```
/*
 *  cluster.c
 *
 *  Pedro Tejada and Minghui Jiang
 *  Tue Jun 23 22:29:46 MDT 2009
 */

#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

double dmax = -1.0; /* maximum distance to merge points */
double range = 2.0; /* number of closest-pair distances */
double T = DBL_MAX; /* weight threshold (accept everything) */

typedef struct {
    int i;
    int j;
    double value;
} s_pair;

s_point *points = NULL;

double dist2_ij(int i, int j) {
    return dist2(&points[i], &points[j]);
}

double value_ij(int i, int j) {
    return dist2_ij(i, j);
}

int cluster(s_point *points, int n_points) {
    s_pair *p, *pairs = NULL;
    double d2, d2_, dmax2;
    double value;
    int *valid_points, *valid_pairs;
    int n_pairs;
    int i, j;

    /* determine dmax */
    if (dmax < 0.0) {

        /* determine closest pair */
        d2 = 0.0;
        d2_ = 2.0;
        for (i = 0; i < n_points; i++)
            for (j = i + 1; j < n_points; j++)
                if ((d2 = dist2_ij(i, j)) < d2_ && d2 > 0.0)
                    d2_ = d2;
        dmax = sqrt(d2_) * range;
    }
    dmax2 = dmax * dmax;
```

```
    printf("dmax = %g\n", dmax);

    /* gather close pairs */
    n_pairs = 0;
    for (i = 0; i < n_points; i++)
        for (j = i + 1; j < n_points; j++)
            if ((d2 = dist2_ij(i, j)) <= dmax2
                    && (value = value_ij(i, j)) <= T)
                n_pairs++;
    if ((pairs = malloc(sizeof(s_pair) * n_pairs)) == NULL) {
        fprintf(stderr, "cluster: malloc failed\n");
        exit(1);
    }
    p = pairs;
    for (i = 0; i < n_points; i++)
        for (j = i + 1; j < n_points; j++)
            if ((d2 = dist2_ij(i, j)) <= dmax2
                    && (value = value_ij(i, j)) <= T) {
                p->i = i;
                p->j = j;
                p->value = value;
                p++;
            }

    /* prepare flag arrays */
    if ((valid_points = malloc(sizeof(int) * n_points)) == NULL) {
        fprintf(stderr, "cluster: malloc failed\n");
        exit(1);
    }
    for (i = 0; i < n_points; i++)
        valid_points[i] = 1;
    if ((valid_pairs = malloc(sizeof(int) * n_pairs)) == NULL) {
        fprintf(stderr, "cluster: malloc failed\n");
        exit(1);
    }
    for (i = 0; i < n_pairs; i++)
        valid_pairs[i] = 1;

    /* clustering */
    while (n_pairs) {
        double value_ = DBL_MAX;
        int i, j, i_ = 0, j_ = 0;
        double xi, xj, yi, yj, wi, wj, ai, aj, a;

        for (i = 0; i < n_pairs; i++) {
            s_pair *p = &pairs[i];

            if (!valid_pairs[i])
                continue;
            if ((value = p->value) < value_) {
                value_ = value;
                i_ = p->i;
                j_ = p->j;
            }
        }
```

```
            /* merge i_ and j_ into i_*/
            xi = points[i_].x; xj = points[j_].x;
            yi = points[i_].y; yj = points[j_].y;
            wi = points[i_].w; wj = points[j_].w;
            ai = points[i_].a; aj = points[j_].a;
            if (aj < ai - 0.5)
                aj += 1.0;
            if (aj > ai + 0.5)
120             ai += 1.0;
            a = (ai * wi + aj * wj) / (wi + wj);
            if ((a = fmod(a, 1.0)) < 0.0)
                a += 1.0;
            points[i_].x = (xi * wi + xj * wj) / (wi + wj);
            points[i_].y = (yi * wi + yj * wj) / (wi + wj);
            points[i_].a = a;
            points[i_].w = wi + wj;
            valid_points[j_] = 0;

130         /* remove edges incident to old i_ and j_ */
            for (i = 0; i < n_pairs; i++)
                if (pairs[i].i == i_ || pairs[i].i == j_
                        || pairs[i].j == i_ || pairs[i].j == j_)
                    valid_pairs[i] = 0;

            /* compress pairs */
            j = 0;
            for (i = 0; i < n_pairs; i++)
                if (valid_pairs[i]) {
140                 if (j < i) {
                        pairs[j] = pairs[i];
                        valid_pairs[j] = 1;
                    }
                    j++;
                }
            n_pairs = j;

            /* add new edges incident to new i_ */
            for (i = 0; i < n_points; i++) {
150             if (!valid_points[i] || i == i_)
                    continue;
                if ((d2 = dist2_ij(i, i_)) <= dmax2
                        && (value = value_ij(i, i_)) <= T) {
                    pairs[n_pairs].i = i_;
                    pairs[n_pairs].j = i;
                    pairs[n_pairs].value = value;
                    valid_pairs[n_pairs] = 1;
                    n_pairs++;
                }
160         }
        }

        /* compress points */
        j = 0;
        for (i = 0; i < n_points; i++)
            if (valid_points[i]) {
                if (j < i)
```

```
                points[j] = points[i];
            j++;
        }
    return j;
}

void print_help_and_exit(int argc, char *argv[]) {
    fprintf(stderr, "usage:  %s [options]\n" "options:\n"
            "\t-i\t<input_filename>\n"
            "\t-o\t<output_filename>\n"
            "\t-d\t<distance>\n"
            "\t-r\t<range>\n",
        argv[0]);
    exit(0);
}

int main(int argc, char **argv) {
    char *input_filename = NULL;
    char *output_filename = NULL;
    int n_points;
    int i;

    for (i = 1; i < argc; i++)
        if (!strcmp(argv[i], "-i")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                input_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-o")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                output_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-d")) {
            if (i + 1 < argc) /* allow negative value */
                dmax = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-r")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                range = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else
            print_help_and_exit(argc, argv);
    if (!input_filename || !output_filename)
        print_help_and_exit(argc, argv);

    points = input_points(input_filename, &n_points);
    printf("%d points\n", n_points);
    n_points = cluster(points, n_points);
    printf("%d points\n", n_points);
    output_points(output_filename, points, n_points);
    return 0;
}
```

B.5 link.c

```
/*
 *  link.c - compute paths
 *
 *  Pedro Tejada
 *  Tue Jun 23 23:02:21 MDT 2009
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

double dmax = -1.0;     /* maximum distance to link points */
double range = 3.0;     /* number of closest-pair distances */
double angle = 35.0;    /* angle (between 0 and 90 deg.) of undirected pairs */
double T = 0.0;         /* pair weight threshold (between 0.0 and 1.0) */

s_point *points;
int n_points;

int *degrees;   /* degrees[i] is number of pairs incident to points[i] */

typedef struct {
    int i;
    int j;
} s_pair;

s_pair *pairs = NULL;
int *used_pairs;
int n_pairs;

s_path **paths;
int n_paths;
int paths_size;

double weight(s_point *p1, s_point *p2) {
    double t, a1, a2, d;
    double w1, w2, wd, w;

    t = atan2((p2->y - p1->y), (p2->x - p1->x));
    a1 = angle_diff(p1->a * M_PI, t);
    a2 = angle_diff(p2->a * M_PI, t);
    d = dist(p1, p2);

    w1 = 1 - a1 / (angle * M_PI / 180.0);
    if (w1 < 0)
        w1 = 0;
    w2 = 1 - a2 / (angle * M_PI / 180.0);
    if (w2 < 0)
        w2 = 0;
    wd = 1 - d / dmax;
    if (wd < 0)
        wd = 0;
```

```
    w = w1;
    if (w > w2)
        w = w2;
    if (w > wd)
        w = wd;
    return w;
}

double dist2_ij(int i, int j) {
    return dist2(&points[i], &points[j]);
}

double value_ij(int i, int j) {
    return weight(&points[i], &points[j]);
}

double v_angle(int i, int j, int k) {
    return vector_angle(&points[i], &points[j], &points[k]);
}

int pair_cmp_weight(s_pair *p1, s_pair *p2) {
    double w1 = value_ij(p1->i, p1->j);
    double w2 = value_ij(p2->i, p2->j);

    if (w1 < w2)
        return 1;
    else if (w1 > w2)
        return -1;
    else
        return 0;
}

void init() {
    int i, j;
    double d2, d2_, dmax2;

    /* determine dmax */
    if (dmax < 0.0) {

        /* find closest pair */
        d2 = 0.0;
        d2_ = 2.0;
        for (i = 0; i < n_points; i++)
            for (j = i + 1; j < n_points; j++)
                if ((d2 = dist2_ij(i, j)) < d2_ && d2 > 0.0)
                    d2_ = d2;
        dmax = sqrt(d2_) * range;
    }
    dmax2 = dmax * dmax;
    printf("dmax = %g\n", dmax);

    /* gather and sort pairs within range */
    n_pairs = 0;
    for (i = 0; i < n_points; i++)
        for (j = i + 1; j < n_points; j++)
            if ((d2 = dist2_ij(i, j)) <= dmax2
```

```
                    && value_ij(i, j) > T)
                n_pairs++;
        if ((pairs = malloc(sizeof(s_pair) * n_pairs)) == NULL) {
            fprintf(stderr, "init: malloc failed\n");
            exit(1);
        }
        n_pairs = 0;
        for (i = 0; i < n_points; i++)
            for (j = i + 1; j < n_points; j++)
                if ((d2 = dist2_ij(i, j)) <= dmax2
                        && value_ij(i, j) > T) {
                    pairs[n_pairs].i = i;
                    pairs[n_pairs].j = j;
                    n_pairs++;
                }
        printf("%d pairs\n", n_pairs);
        qsort(pairs, n_pairs, sizeof(s_pair),
            (int (*)(const void*, const void*)) pair_cmp_weight);

        /* prepare flag arrays for used pairs */
        if ((used_pairs = malloc(sizeof(int) * n_pairs)) == NULL) {
            fprintf(stderr, "init: malloc failed\n");
            exit(1);
        }
        memset(used_pairs, 0, n_pairs * sizeof(int));

        /* degrees array */
        if ((degrees = malloc(sizeof(int) * n_points)) == NULL) {
            fprintf(stderr, "init: malloc failed\n");
            exit(1);
        }
        memset(degrees, 0, n_points * sizeof(int));

        /* paths array */
        paths_size = 2;
        if ((paths = malloc(sizeof(s_path *) * paths_size)) == NULL) {
            fprintf(stderr, "init: malloc failed\n");
            exit(1);
        }
        n_paths = 0;
    }

    int find_next_pair(int i, int j) {
        s_pair *p;
        int i_pairs, i_next, k;
        double value, value_;

        i_next = -1;    /* no next */
        value_ = -1.0;
        for (i_pairs = 0; i_pairs < n_pairs; i_pairs++) {
            p = &pairs[i_pairs];
            if (p->i != j && p->j != j)
                continue;
            if ((p->i == i || p->j == i) && i != j)
                continue;
            k = p->i;
```

```
            if (k == j)
                k = p->j;
        if (i == j || v_angle(i, j, k) <= (M_PI / 2.0))
            if ((value = value_ij(j, k)) >= T && value > value_) {
                value_ = value;
                i_next = i_pairs;
            }
    }
    return i_next;
}

void extend_path_ij(s_path *p, int i_points, int j_points) {
    int i_pairs;

    if (degrees[j_points] > 1)
        return;
    if ((i_pairs = find_next_pair(i_points, j_points)) == -1)
        return;
    if (used_pairs[i_pairs])
        return;

    used_pairs[i_pairs] = 1;
    degrees[pairs[i_pairs].i]++;
    degrees[pairs[i_pairs].j]++;

    /* move to next place */
    i_points = j_points;
    j_points = (pairs[i_pairs].i == i_points) ?
                pairs[i_pairs].j : pairs[i_pairs].i;

    extend_path(p, j_points);
    extend_path_ij(p, i_points, j_points);
}

void extend_paths(s_path *p) {
    if (n_paths == paths_size) { /* no space available */
        s_path **tmp;
        int i;

        paths_size *= 2;
        if ((tmp = malloc(paths_size * sizeof(s_path *))) == NULL) {
            fprintf(stderr, "extend_paths: malloc failed\n");
            exit(1);
        }
        for (i = 0; i < n_paths; i++)
            tmp[i] = paths[i];
        free(paths);
        paths = tmp;
    }
    paths[n_paths++] = p;
}

/**
 * Every time a pair is selected, extend it to both sides as far as possible.
 */
void link() {
```

```
        s_path *p;
        int i_pairs;

        init();
        for (i_pairs = 0; i_pairs < n_pairs; i_pairs++) {
            s_pair *pair = &pairs[i_pairs];

            if (used_pairs[i_pairs])
                continue;
            if (degrees[pair->j] > 0 || degrees[pair->i] > 0)
                continue;

            /* initial pair for path */
            p = new_path(0);
            extend_paths(p);
            extend_path(p, pair->i);
            extend_path(p, pair->j);
            used_pairs[i_pairs] = 1;
            degrees[pair->i]++;
            degrees[pair->j]++;

            extend_path_ij(p, pair->i, pair->j);    /* extend in first direction */
            reverse_path(p);    /* necessary for array implementation */
            extend_path_ij(p, pair->j, pair->i);    /* extend in other direction */
        }
    }

    /*
     *  Initialization
     */

    void print_help_and_exit(int argc, char *argv[]) {
        fprintf(stderr, "usage:  %s [options]\n" "options:\n"
                "\t-i\t<input_file>\n"
                "\t-o\t<output_file>\n"
                "\t-d\t<distance>\n"
                "\t-r\t<range_threshold>\n"
                "\t-a\t<angle_threshold>\n",
            argv[0]);
        exit(0);
    }

    int main(int argc, char *argv[]) {
        char *input_filename = NULL;
        char *output_filename = NULL;
        int i;

        for (i = 1; i < argc; i++)
            if (!strcmp(argv[i], "-i")) {
                if (i + 1 < argc && argv[i + 1][0] != '-')
                    input_filename = argv[++i];
                else
                    print_help_and_exit(argc, argv);
            } else if (!strcmp(argv[i], "-o")) {
                if (i + 1 < argc && argv[i + 1][0] != '-')
                    output_filename = argv[++i];
```

```
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-d")) {
            if (i + 1 < argc) /* allow negative value */
                dmax = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-r")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                range = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-a")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                angle = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else
            print_help_and_exit(argc, argv);
    if (!input_filename || !output_filename)
        print_help_and_exit(argc, argv);

    points = input_points(input_filename, &n_points);
    printf("%d points\n", n_points);
    link();
    output_paths(output_paths, paths, n_paths, n_points);
    printf("%d points\n", n_points);
    printf("%d paths\n", n_paths);
    return 0;
}
```

B.6 simplify.c

```
/*
 *  simplify.c - reduce number of points in paths
 *
 *  Pedro Tejada
 *  Tue Jun 23 23:20:38 MDT 2009
 */

#include <math.h>
#include <float.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

#define DEBUG   0

int rem_isolated = 1;       /* remove isolated points */
double min_length = 0.0;    /* minimum length of paths to output */

double d_max = -1.0; /* max dist. allowed from original path */
double factor = 0.5; /* max dist. from original path = (min dist.) x factor */

s_point *points;
int n_points;

s_path **paths;
int n_paths;

int *K;     /* path length (number of points) */
int *P;     /* previous point */

double dist_ij(int i, int j) {
    return dist(&points[i], &points[j]);
}

double dist_pij(int i_path, int i, int j) {
    return dist_ij(paths[i_path]->indices[i], paths[i_path]->indices[j]);
}

double dist2seg_(int p, int q1, int q2) {
    return dist2seg(&points[p], &points[q1], &points[q2]);
}

/* remove isolated points (not part of any path) */
void remove_isolated() {
    int *new_points;
    int i, j, k;

    if ((new_points = malloc(sizeof(int) * n_points)) == NULL) {
        fprintf(stderr, "remove_isolated: malloc failed\n");
        exit(1);
    }
    for (i = 0; i < n_points; i++)
        new_points[i] = -1;
```

```
    /* flag points in paths */
    for (i = 0; i < n_paths; i++)
        for (j = 0; j < paths[i]->length; j++)
            new_points[paths[i]->indices[j]] = 1;

    /* assign new indices */
    k = 0;
    for (j = 0; j < n_points; j++)
        if (new_points[j] != -1)
            new_points[j] = k++;
    for (i = 0; i < n_paths; i++)
        for (j = 0; j < paths[i]->length; j++) {
            k = paths[i]->indices[j];
            paths[i]->indices[j] = new_points[k];
        }

    /* remove unused points */
    k = 0;
    for (j = 0; j < n_points; j++)
        if (new_points[j] != -1) {
            if (k < j)
                points[k] = points[j];
            k++;
        }
    n_points = k;
}

void remove_short_paths() {
    int *valid_paths;
    int i, j, k;

    if ((valid_paths = malloc(sizeof(int) * n_paths)) == NULL) {
        fprintf(stderr, "remove_short_paths: malloc failed\n");
        exit(1);
    }
    for (i = 0; i < n_paths; i++) {
        double length = 0.0;

        for (j = 0; j < paths[i]->length - 1; j++)
            length += dist_ij(paths[i]->indices[j], paths[i]->indices[j + 1]);
        valid_paths[i] = length >= min_length;
    }
    k = 0;
    for (j = 0; j < n_paths; j++)
        if (valid_paths[j] != 0) {
            if (k < j)
                paths[k] = paths[j];
            k++;
        }
    n_paths = k;
    free(valid_paths);
}

/* Recover path from DP tables */
int recover_path1(int i_paths) {
    s_path *p;
```

```
    int *used;
    int i, j, k, prev;

    p = paths[i_paths];
    if ((used = malloc(sizeof(int) * p->length)) == NULL) {
        fprintf(stderr, "recover_path1: malloc failed\n");
        exit(1);
    }
    memset(used, 0, p->length * sizeof(int));

    /* Mark all points used by simplified path */
    used[0] = used[p->length - 1] = 1; /* keep extreme points */
    i = p->length - 1; /* last point in path */
    while (i >= 0 && (prev = P[i]) >= 0) {
        used[prev] = 1;
        i = prev;
    }

    /* Remove unused points */
    k = 0;
    for (j = 0; j < p->length; j++)
        if (used[j] == 1) {
            if (k < j)
                p->indices[k] = p->indices[j];
            k++;
        }
    p->length = k;
    free(used);
    return k;
}

/* DP algorithm for single path */
void simplify_path1(int i_paths, double e) {
    s_path *p;
    double *L;      /* sub-paths lengths */
    double d, r, T; /* distance, ratio, threshold */
    int i, j;

    p = paths[i_paths];
    if ((K = malloc(sizeof(int) * p->length)) == NULL) {
        fprintf(stderr, "simplify_path1: malloc failed\n");
        exit(1);
    }
    if ((P = malloc(sizeof(int) * p->length)) == NULL) {
        fprintf(stderr, "simplify_path1: malloc failed\n");
        exit(1);
    }
    if ((L = malloc(sizeof(double) * p->length)) == NULL) {
        fprintf(stderr, "simplify_path1: malloc failed\n");
        exit(1);
    }
    L[0] = 0.0;
    for (i = 1; i < p->length; i++)
        L[i] = L[i - 1] + dist_pij(i_paths, i - 1, i);

    /* base cases */
```

```
        for (i = 0; i < p->length; i++) {
            K[i] = i + 1;
            P[i] = i - 1;
        }

        /* recursion */
        for (i = 1; i < p->length; i++)
            for (j = 0; j < i; j++) {
                d = dist_pij(i_paths, j, i);
                r = (L[i] - L[j]) / d;
                T = (2 * sqrt(e * e + d * d / 4)) / d;
                if (d > 0.0 && r <= T)
                    if (K[j] + 1 < K[i]) {
                        K[i] = K[j] + 1;
                        P[i] = j;
                    }
            }
        recover_path1(i_paths);
        free(K);
        free(P);
        free(L);
    }

    void simplify() {
        double d, d_min;
        int i, j;

        /* determine d_max automatically if not specified */
        if (d_max < 0.0) {
            d_min = 2.0;
            for (i = 0; i < n_points; i++)
                for (j = i + 1; j < n_points; j++)
                    if ((d = dist_ij(i, j)) < d_min && d > 0.0)
                        d_min = d;
            printf("dmin: %g\n", d_min);
            d_max = d_min * factor;
        }
        printf("dmax: %g\n", d_max);

        /* simplify each path */
        for (i = 0; i < n_paths; i++)
            simplify_path1(i, d_max);

        /* filter output */
        if (min_length > 0.0)
            remove_short_paths();
        if (rem_isolated)
            remove_isolated();
    }

    void print_help_and_exit(int argc, char *argv[]) {
        fprintf(stderr, "usage:  %s [options]\n" "options:\n"
                "\t-i1\t<pnt_input_filename>\n"
                "\t-i1\t<pth_input_filename>\n"
                "\t-o1\t<pnt_output_filename>\n"
                "\t-o2\t<pth_output_filename>\n"
```

```
            "\t-d\t<maximum distance>\n"
            "\t-df\t<min distance factor>\n"
            "\t-ml\t<minimum path length>\n"
            "\t-ri\t<remove isolated points>\n",
        argv[0]);
    exit(0);
}

int main(int argc, char *argv[]) {
    char *pnt_input_filename = NULL;
    char *pth_input_filename = NULL;
    char *pnt_output_filename = NULL;
    char *pth_output_filename = NULL;
    int i;

    for (i = 1; i < argc; i++)
        if (!strcmp(argv[i], "-i1")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                pnt_input_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-i2")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                pth_input_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-o1")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                pnt_output_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-o2")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                pth_output_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-d")) {
            if (i + 1 < argc) /* allow negative value */
                d_max = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-df")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                factor = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-ml")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                min_length = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-ri")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                rem_isolated = atoi(argv[++i]);
            else
                print_help_and_exit(argc, argv);
```

```
280         } else
                print_help_and_exit(argc, argv);
        if (!pnt_input_filename || !pth_input_filename
                || !pnt_output_filename || !pth_output_filename)
            print_help_and_exit(argc, argv);

        points = input_points(pnt_input_filename, &n_points);
        paths = input_paths(pth_input_filename, &n_paths, &n_points);
        printf("%d points\n", n_points);
        printf("%d paths\n", n_paths);
290     simplify();
        output_points(pnt_output_filename, points, n_points);
        output_paths(pth_output_filename, paths, n_paths, n_points);
        printf("%d points\n", n_points);
        printf("%d paths\n", n_paths);
        return 0;
    }
```

B.7 show.c

```
/*
 *  show.c - visualizer
 *
 *  Pedro Tejada and Minghui Jiang
 *  Sat Jun 27 18:42:06 MDT 2009
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifdef __APPLE__
#include <GLUT/glut.h>  /* MacOSX */
#else
#include <GL/glut.h>    /* Cygwin/Linux */
#endif
#include "lib.h"

#define RADIUS_SLOPE 0.0090
#define WINDOW_SIZE 512
#define LINE_WIDTH 0.5
#define POINT_SIZE 2.8

typedef struct {
    unsigned long w;
    unsigned long h;
    char *data;
} s_gl_image;

unsigned int texture;

s_gl_image *image1;

s_point *points1 = NULL;
int n_points1 = 0;

s_point *points2 = NULL;
int n_points2 = 0;

s_path **paths = NULL;
int n_paths = 0;

/*
 *  Display
 */

double space = 0.025; /* space around boundary */

int show_points1 = 1;
int show_points2 = 1;
int show_orient1 = 0;
int show_orient2 = 0;
int show_paths = 1;
int show_border = 1;
int show_image = 1;
```

```
double alpha = 0.3; /* image transparency */

const float COLOR_RED[4]    = {1.0, 0.0, 0.0, 1.0};
const float COLOR_YELLOW[4] = {1.0, 1.0, 0.0, 1.0};
const float COLOR_GREEN[4]  = {0.0, 1.0, 0.0, 1.0};
const float COLOR_BLUE[4]   = {0.0, 0.0, 1.0, 1.0};
const float COLOR_WHITE[4]  = {1.0, 1.0, 1.0, 1.0};
const float COLOR_BLACK[4]  = {0.0, 0.0, 0.0, 1.0};
const float COLOR_GRAY[4]   = {0.4, 0.4, 0.4, 1.0};

void draw_point(s_point *p) {
    glVertex2d(p->x, p->y);
}

void draw_orientation(s_point *p) {
    double a = p->a * M_PI;
    double dx = RADIUS_SLOPE * cos(a);
    double dy = RADIUS_SLOPE * sin(a);

    glVertex2d(p->x - dx, p->y - dy);
    glVertex2d(p->x + dx, p->y + dy);
}

void display() {
    double x, y, w, h;
    int i, j;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glPushMatrix();

    w = h = 1.0;
    x = y = 0.0;
    if (image1) {
        if (image1->w > image1->h)
            h = (double) image1->h / image1->w;
        else
            w = (double) image1->w / image1->h;
        y = (1.0 - h) / 2;
        x = (1.0 - w) / 2;
    }

    if (image1 && show_image) {
        glBindTexture(GL_TEXTURE_2D, texture);
        glColor4f(1.0, 1.0, 1.0, alpha);
        glPolygonMode(GL_FRONT, GL_FILL);

        glBegin(GL_POLYGON);
        glTexCoord2f(0.0, 0.0);     /* top left */
        glVertex2d(x, y);
        glTexCoord2f(1.0, 0.0);     /* top right */
        glVertex2d(x + w, y);
        glTexCoord2f(1.0, 1.0);     /* bottom right */
        glVertex2d(x + w, y + h);
        glTexCoord2f(0.0, 1.0);     /* bottom left */
        glVertex2d(x, y + h);
```

```
            glEnd();

            glBindTexture(GL_TEXTURE_2D, 0);
        }

        if (show_border) {
            glBindTexture(GL_TEXTURE_2D, 0);
            glColor3f(0.0, 0.0, 0.0);
120         glPolygonMode(GL_FRONT, GL_LINE);
            glDisable(GL_LINE_SMOOTH);

            glBegin(GL_POLYGON);
            glVertex2d(x, y);           /* top left */
            glVertex2d(x + w, y);       /* top right */
            glVertex2d(x + w, y + h);   /* bottom right */
            glVertex2d(x, y + h);       /* bottom left */
            glEnd();
            glEnable(GL_LINE_SMOOTH);
130     }

        if (show_orient2) {
            glColor3f(COLOR_GREEN[0], COLOR_GREEN[1], COLOR_GREEN[2]);
            glBegin(GL_LINES);
            for (i = 0; i < n_points2; i++)
                draw_orientation(&points2[i]);
            glEnd();
        }

140     if (show_points2 && points2) {
            glColor3f(COLOR_GREEN[0], COLOR_GREEN[1], COLOR_GREEN[2]);
            glBegin(GL_POINTS);
            for (i = 0; i < n_points2; i++)
                draw_point(&points2[i]);
            glEnd();
        }

        if (show_orient1) {
            glColor3f(COLOR_BLUE[0], COLOR_BLUE[1], COLOR_BLUE[2]);
150         glBegin(GL_LINES);
            for (i = 0; i < n_points1; i++)
                draw_orientation(&points1[i]);
            glEnd();
        }

        if (show_points1 && points1) {
            glColor3f(COLOR_RED[0], COLOR_RED[1], COLOR_RED[2]);
            glBegin(GL_POINTS);
            for (i = 0; i < n_points1; i++)
160             draw_point(&points1[i]);
            glEnd();
        }

        if (show_paths) {
            if(image1 && show_image)
                glColor3f(COLOR_BLUE[0], COLOR_BLUE[1], COLOR_BLUE[2]);
            else
```

```
            glColor3f(COLOR_BLACK[0], COLOR_BLACK[1], COLOR_BLACK[2]);
        glBegin(GL_LINES);
        for (i = 0; i < n_paths; i++) {
            s_path *p = paths[i];

            for (j = 0; j < p->length - 1; j++) {
                glVertex2d(points1[p->indices[j]].x, points1[p->indices[j]].y);
                glVertex2d(points1[p->indices[j+1]].x, points1[p->indices[j+1]].y);
            }
        }
        glEnd();
    }

    glPopMatrix();
    glutSwapBuffers();
}

void reshape(int w, int h) {
    double range = 1.0 + space * 2.0;
    double width, height;
    double h_margin, v_margin;

    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    width = height = range;
    if (w > h)
        width = ((double) w / h) * range;
    else
        height = ((double) h / w) * range;
    h_margin = (width - 1.0) / 2.0;
    v_margin = (height - 1.0) / 2.0;
    gluOrtho2D(-h_margin, 1.0 + h_margin, -v_margin, 1.0 + v_margin);

    glPointSize(fmin(w, h) / (WINDOW_SIZE / POINT_SIZE));
    glLineWidth(fmin(w, h) / (WINDOW_SIZE / LINE_WIDTH));
    glMatrixMode(GL_MODELVIEW);
}

/*
 *  Event handlers
 */

void keyboard(unsigned char key, int x, int y) {
    if (key >= '0' && key <= '9')
        alpha = (key == '0') ? 1.0 : (key - '0') * 0.1;
    switch (key) {
        case 'p':
            show_points1 = !show_points1;
            break;
        case 'P':
            show_points2 = !show_points2;
            break;
        case 'o':
            show_orient1 = !show_orient1;
```

```
            break;
        case 'O':
            show_orient2 = !show_orient2;
            break;
        case 'c':
            show_paths = !show_paths;
            break;
        case 'b':
            show_border = !show_border;
            break;
        case 'i':
            show_image = !show_image;
            break;
        case ' ':   /* reset */
            show_points1 = 1;
            show_points2 = 1;
            show_orient1 = 0;
            show_orient2 = 0;
            show_paths = 1;
            show_border = 1;
            show_image = 1;
            alpha = 0.3;
            break;
        case 27:    /* escape to quit */
            exit(0);
    }
    glutPostRedisplay();
}

/*
 *  Initialization
 */

s_gl_image *load_gl_image(char *ppm_filename) {
    s_gl_image *gl_img;
    s_image *ppm_img;
    unsigned long i_data;
    int x, y;

    ppm_img = load_ppm(ppm_filename);

    if ((gl_img = (s_gl_image *) malloc(sizeof(s_gl_image))) == NULL) {
        fprintf(stderr, "load_gl_image: malloc failed\n");
        exit(1);
    }
    gl_img->w = ppm_img->w;
    gl_img->h = ppm_img->h;
    gl_img->data = (char *) malloc(gl_img->w * gl_img->h * 3);

    i_data = 0;
    for (y = ppm_img->h - 1; y >= 0; y--)
        for (x = 0; x < ppm_img->w; x++) {
            gl_img->data[i_data] = ppm_img->p[x][y].r;
            gl_img->data[i_data + 1] = ppm_img->p[x][y].g;
            gl_img->data[i_data + 2] = ppm_img->p[x][y].b;
            i_data += 3;
```

```
        }
    free(ppm_img);
    return gl_img;
}

/* Load images and convert to textures.
 * See: http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=06
 */
void load_gl_textures(char *ppm_filename) {
    image1 = load_gl_image(ppm_filename);

    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

    /* linear scaling when image size different that texture size */
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

    /* 2d texture, level of detail 0 (normal), 3 components (red, green, blue),
     * x size from image, y size from image, border 0 (normal), rgb color data,
     * unsigned byte data, and finally the data itself.
     */
    glTexImage2D(GL_TEXTURE_2D, 0, 3, image1->w, image1->h,
                 0, GL_RGB, GL_UNSIGNED_BYTE, image1->data);
}

void init_gl(int w, int h) {
    glPolygonMode(GL_FRONT, GL_LINE);
    glEnable(GL_LINE_SMOOTH);
    glEnable(GL_POINT_SMOOTH);
    glHint(GL_LINE_SMOOTH_HINT, GL_DONT_CARE);
    glHint(GL_POINT_SMOOTH_HINT, GL_DONT_CARE);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glLineWidth(LINE_WIDTH);
    glPointSize(POINT_SIZE);
    glClearColor(1.0, 1.0, 1.0, 0.0);   /* white background */
    glColor3f(0.0, 0.0, 0.0);           /* black */
    glEnable(GL_TEXTURE_2D);
}

void print_help_and_exit(int argc, char *argv[]) {
    fprintf(stderr, "usage:\n"
            "\t%s pnt_input_file [options]\n"
            "options:\n"
            "[-p2 pnt2_input_file] "
            "[-c pth_input_file] "
            "[-i ppm_input_file]\n",
        argv[0]);
    exit(0);
}

int main(int argc, char **argv) {
    char *pnt1_input_file = argv[1];
    char *pnt2_input_file = NULL;
```

```
    char *pth_input_file = NULL;
    char *ppm_input_file = NULL;
    int i;

    for (i = 1; i < argc; i++)
        if (!strcmp(argv[i], "-p")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                pnt1_input_file = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-P")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                pth_input_file = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-p2")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                pnt2_input_file = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-i")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                ppm_input_file = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else
            print_help_and_exit(argc, argv);
    if (!pnt1_input_file)
        print_help_and_exit(argc, argv);

    points1 = input_points(pnt1_input_file, &n_points1);
    if (pnt2_input_file != NULL)
        points2 = input_points(pnt2_input_file, &n_points2);
    if (pth_input_file != NULL)
        paths = input_paths(pth_input_file, &n_paths, &n_points1);

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(WINDOW_SIZE, WINDOW_SIZE);
    glutCreateWindow(pnt1_input_file);
    glutDisplayFunc(&display);
    glutKeyboardFunc(keyboard);
    glutReshapeFunc(&reshape);

    init_gl(WINDOW_SIZE, WINDOW_SIZE);
    if (ppm_input_file != NULL)
        load_gl_textures(ppm_input_file);

    glutMainLoop();
    return 0;
}
```

Appendix C

Format Manipulation Source Code

C.1 jpg2ppm.manifest

```
Main-Class: jpg2ppm
```

C.2 jpg2ppm.java

```
import java.awt.color.ColorSpace;
import java.awt.image.*;
import java.io.*;
import javax.swing.*;
import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageDecoder;

/**
 * <p>Converts .jpg images to .ppm images.</p>
 *
 * @author Pedro Tejada and Minghui Jiang
 * @version Wed Mar 25 10:45:58 MDT 2009
 */
class jpg2ppm {
    public static void main(String[] args) throws IOException {
        String inputFile = null, outputFile = null;
        boolean gray = false;
        int i = 0;

        if (i < args.length)
            inputFile = args[i++];
        if (i < args.length && !args[i].startsWith("-"))
            outputFile = args[i++];
        for (; i < args.length; i++) {
            if (args[i].equals("-g"))
                gray = true;
            else
                printHelpAndExit();
        }

        if (inputFile != null && outputFile != null)
            convert(inputFile, outputFile, gray);
        else if (inputFile != null && outputFile == null)
            view(inputFile, gray);
        else
            printHelpAndExit();
    }
```

```
    static void printHelpAndExit() {
        System.out.println("parameters:\n"
                + "\tinput_file_name [output_file_name] [options]");
        System.out.println("options:\n"
                + "\t-g\tconvert to grayscale");
        System.exit(1);
    }

    static BufferedImage getImage(String file) throws IOException {
        BufferedImage image = null;

        if (file.endsWith(".jpg") || file.endsWith(".jpeg")) {
            InputStream is = new BufferedInputStream(new FileInputStream(file));
            JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(is);
            image = decoder.decodeAsBufferedImage();
        } else if (file.endsWith(".ppm"))
            image = decodeImage(file);
        return image;
    }

    static void makeGray(BufferedImage image) {
        ColorConvertOp op = new ColorConvertOp(
                ColorSpace.getInstance(ColorSpace.CS_GRAY), null);
        op.filter(image, image);
    }

    static void view(String file, boolean gray) throws IOException {
        BufferedImage image = getImage(file);
        JFrame frame = new JFrame(file);
        if (gray)
            makeGray(image);
        frame.getContentPane().add(new JLabel(new ImageIcon(image)));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }

    static void convert(String inputFile, String outputFile,
            boolean gray) throws IOException {
        BufferedImage image = getImage(inputFile);
        if (gray)
            makeGray(image);
        encodeImage(image, outputFile);
    }

    static void encodeImage(BufferedImage image,
            String file) throws IOException {
        PrintWriter writer = new PrintWriter(
                new BufferedWriter(new FileWriter(file)));
        writer.println("P3");   // signature
        writer.println("# created by jpg2ppm");
        writer.println(image.getWidth() + " " + image.getHeight() + " 255");

        for (int y = 0; y < image.getHeight(); y++)
            for (int x = 0; x < image.getWidth(); x++)
```

```
                for (int rgb = 0; rgb <= 2; rgb++) {
                    writer.print(image.getRaster()
                        .getSample(x, y, rgb % image.getRaster().getNumBands()));
                    if (rgb < 2)
                        writer.print(" ");
                    else
100                     writer.println();
                }
            writer.close();
        }

        static BufferedImage decodeImage(String file) throws IOException {
            StreamTokenizer parser = new StreamTokenizer(new BufferedReader(
                        new InputStreamReader(new FileInputStream(file))));
            parser.commentChar('#');

110         parser.nextToken(); // signature
            if (parser.ttype != StreamTokenizer.TT_WORD && parser.sval != "P3")
                throw new IOException("Invalid file format");

            parser.nextToken();
            int width = (int) parser.nval;

            parser.nextToken();
            int height = (int) parser.nval;

120         parser.nextToken(); // max-value

            BufferedImage image = new BufferedImage(
                    width, height, BufferedImage.TYPE_INT_RGB);
            for (int y = 0; y < height; y++)
                for (int x = 0; x < width; x++)
                    for (int rgb = 0; rgb <= 2 ; rgb++) {
                        parser.nextToken();
                        if (parser.ttype == StreamTokenizer.TT_EOF)
                            throw new EOFException("Unexpected end of file");
130                     if (parser.ttype != StreamTokenizer.TT_NUMBER)
                            throw new IOException("Invalid file format");
                        image.getRaster().setSample(x, y, rgb, (int) parser.nval);
                    }
            return image;
        }
    }
```

C.3 pth2fig.c

```
/*
 *  pth2fig.c - convert .pth to .fig
 *      draw each path as an open interporated spline curve
 *      http://xfig.org/userman/frm_drawing.html
 *
 *  Pedro Tejada
 *  Sat Mar 14 01:36:34 MDT 2009
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

#define IMG_SIZE (1200 * 5)
#define PNT_SIZE (IMG_SIZE / 256)

int spline = 0;         /* paths format: 0 = polyline, 1 = spline */
int show_points = 0;    /* output points */

s_point *points;
s_path **paths;
int n_paths, n_points;

void output_points_line(FILE *file, s_path *p) {
    int j;

    fprintf(file, "\t");
    for (j = 0; j < p->length; j++) {
        int x = (int) (points[p->indices[j]].x * IMG_SIZE);
        int y = (int) (IMG_SIZE - points[p->indices[j]].y * IMG_SIZE);

        fprintf(file, " %d %d", x, y);
    }
    fprintf(file, "\n");
}

void output_polyline(FILE *file, s_path *p) {

    /* definition line */
    fprintf(file, "2 1 0 1 0 7 50 -1 -1 0.000 0 0 0 0 0 %d\n", p->length);

    output_points_line(file, p);
}

void output_spline(FILE *file, s_path *p) {
    int j;

    /* definition line */
    fprintf(file, "3 2 0 1 0 7 50 -1 -1 0.000 0 0 0 %d\n", p->length);

    output_points_line(file, p);

    /* control points line */
    fprintf(file, "\t");
```

```
        for (j = 0; j < p->length; j++)
            if (j == 0 || j == p->length - 1)
                fprintf(file, " 0.000");
            else
                fprintf(file, " -1.000");
        fprintf(file, "\n");
    }

    void output_points_(FILE *file) {
        int i;

        for (i = 0; i < n_points; i++) {
            int x = (int) (points[i].x * IMG_SIZE);
            int y = (int) (IMG_SIZE - points[i].y * IMG_SIZE);

            fprintf(file, "1 3 0 1 4 4 50 -1 20 0.000 1 0.0000");
            fprintf(file, " %d %d", x, y); /* center */
            fprintf(file, " %d %d", PNT_SIZE / 2, PNT_SIZE / 2); /* radii */
            fprintf(file, " %d %d", x, y); /* first point */
            fprintf(file, " %d %d", x + PNT_SIZE / 2, y); /* second point */
            fprintf(file, "\n");
        }
    }

    void output_bounding_box(FILE *file) {
        fprintf(file, "2 2 0 1 0 7 50 -1 -1 0.000 0 0 -1 0 0 5\n");
        fprintf(file, "\t");
        fprintf(file, " %d %d", 0, 0);
        fprintf(file, " %d %d", IMG_SIZE, 0);
        fprintf(file, " %d %d", IMG_SIZE, IMG_SIZE);
        fprintf(file, " %d %d", 0, IMG_SIZE);
        fprintf(file, " %d %d", 0, 0);
        fprintf(file, "\n");
    }

    void output_fig(char *file_name, s_point *points, s_path **paths,
            int n_points, int n_paths) {
        FILE *file;
        int i;

        if ((file = fopen(file_name, "w")) == NULL) {
            fprintf(stderr, "output_fig: fopen(%s) failed\n", file_name);
            exit(1);
        }

        /* header */
        fprintf(file, "#FIG 3.2\n");
        fprintf(file, "Landscape\n");
        fprintf(file, "Center\n");
        fprintf(file, "Inches\n");
        fprintf(file, "Letter\n");
        fprintf(file, "100.00\n");
        fprintf(file, "Single\n");
        fprintf(file, "-2\n");
        fprintf(file, "1200 2\n");
```

```
        output_bounding_box(file);
        for (i = 0; i < n_paths; i++)
            if (spline)
                output_spline(file, paths[i]);
            else
                output_polyline(file, paths[i]);
        if (show_points)
            output_points_(file);
120     fclose(file);
    }

    /*
     *  Initialization
     */

    void print_help_and_exit(int argc, char *argv[]) {
        fprintf(stderr, "usage: %s points_file_name paths_file_name "
                "fig_file_name [options]\n" "options:\n"
130             "\t-s\t<splines>\n"
                "\t-p\t<output points>\n",
            argv[0]);
        exit(1);
    }

    int main(int argc, char *argv[]) {
        int i;

        if (argc < 4)
140         print_help_and_exit(argc, argv);
        for (i = 4; i < argc; i++)
            if (!strcmp(argv[i], "-s"))
                spline = 1;
            else if (!strcmp(argv[i], "-p"))
                show_points = 1;
            else
                print_help_and_exit(argc, argv);

        points = input_points(argv[1], &n_points);
150     paths = input_paths(argv[2], &n_paths, &n_points);
        output_fig(argv[3], points, paths, n_points, n_paths);
        return 0;
    }
```

Appendix D

Evaluation Source Code

D.1 noise.c

```
/*
 *  noise.c - add noise to .ppm image (salt and pepper)
 *                      or .pnt file (random points)
 *
 *  Pedro Tejada
 *  Sat Jun 27 22:10:16 MDT 2009
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "lib.h"

#define PPM 1
#define PNT 2

int file_type = -1;     /* PPM or PNT */
int seed;               /* random number generator seed */
double density = 0.05;  /* used to determine number of pixels for .ppm image */
int n = 1000;           /* number of noise points for .pnt file */

void print_help_and_exit(int argc, char *argv[]) {
    fprintf(stderr, "usage: %s input_file output_file [options]\n"
            "options:\n"
            "\t-ppm\t<ppm_file_type>\n"
            "\t-pnt\t<pnt_file_type>\n"
            "\t-s\t<random_seed>\n"
            "\t-d\t<noise_density> (for .ppm image)\n"
            "\t-n\t<number_of_points> (for .pnt file)\n",
        argv[0]);
    exit(1);
}

int main(int argc, char *argv[]) {
    int i, n;

    seed = (int) time(NULL);
    if (argc < 3)
        print_help_and_exit(argc, argv);
    for (i = 3; i < argc; i++)
        if (!strcmp(argv[i], "-ppm")) {
            if (file_type == PNT)
```

```
                print_help_and_exit(argc, argv);
            file_type = PPM;
        } else if (!strcmp(argv[i], "-pnt")) {
            if (file_type == PPM)
                print_help_and_exit(argc, argv);
            file_type = PNT;
        } else if (!strcmp(argv[i], "-s")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                seed = atoi(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-d")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                density = atof(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-n")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                n = atoi(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else
            print_help_and_exit(argc, argv);
    if (file_type != PPM && file_type != PNT)
        print_help_and_exit(argc, argv);

    if (file_type == PPM) {
        s_image *img;
        int x, y;

        img = load_ppm(argv[1]);
        n = (int) (density * img->h * img->w);
        printf("%d noise pixels\n", n);

        srand(seed);
        for (i = 0; i < n; i++) {
            x = (int) ((img->w - 1) * RAND(1.0));
            y = (int) ((img->h - 1) * RAND(1.0));
            img->p[x][y].r = (int) RAND(MAX_VALUE);
            img->p[x][y].g = (int) RAND(MAX_VALUE);
            img->p[x][y].b = (int) RAND(MAX_VALUE);
        }
        save_ppm(img, argv[2]);
        return 0;
    } else if (file_type == PNT) {
        s_point *points;
        int n_points;
        s_point_array *pnt_array;

        points = input_points(argv[1], &n_points);
        printf("%d noise points\n", n);

        pnt_array = new_point_array(0);
        extend_point_array(pnt_array, points, n_points);
        srand(seed);
        for (i = 0; i < n; i++)
```

```
100             extend_point_array1(pnt_array,
                    new_point(RAND(1.0), RAND(1.0), RAND(1.0), RAND(1.0)));
            output_points(argv[2], pnt_array->points, pnt_array->n);
        }
        return 0;
    }
```

D.2 discretize.c

```
/*
 *  discretize.c - compare results
 *
 *  Pedro Tejada
 *  Mon Mar 16 01:29:17 MDT 2009
 */

#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

double delta = 0.0009765625; /* max space between points (= 1 / 1024) */

s_point *points;
int n_points;

s_path **paths;
int n_paths;

s_point *discretize(int *np) {
    s_point_array *pnt_array;
    int i, j, k;

    pnt_array = new_point_array(0);

    /* copy existing points */
    extend_point_array(pnt_array, points, n_points);

    /* discretize segments */
    for (i = 0; i < n_paths; i++) {
        for (j = 0; j < paths[i]->length - 1; j++) {
            s_point *p = &points[paths[i]->indices[j]];
            s_point *q = &points[paths[i]->indices[j + 1]];
            double length = dist(p, q);
            double dx, dy;  /* distances between points in line */
            double a = atan2((q->y - p->y), (q->x - p->x)) / M_PI;
            double x, y;
            int n = 2;

            if (delta < length)
                n += ceil(length / delta) - 1;
            dx = (q->x - p->x) / (n - 1);
            dy = (q->y - p->y) / (n - 1);

            for (k = 1; k < n-1; k++) {
                x = p->x + k * dx;
                y = p->y + k * dy;
                extend_point_array1(pnt_array, new_point(x, y, a, 1.0));
            }
        }
    }
```

```
    *np = pnt_array->n;
    return pnt_array->points;
}

int main(int argc, char *argv[]) {
    s_point *points_;
    int n_points_;

    if (argc != 4) {
        fprintf(stderr, "usage: %s pnt_input_file pth_input_file"
            " pnt_output_file\n", argv[0]);
        exit(1);
    }

    points = input_points(argv[1], &n_points);
    paths = input_paths(argv[2], &n_paths, &n_points);
    printf("%d points\n", n_points);
    printf("%d paths\n", n_paths);

    points_ = discretize(&n_points_);
    output_points(argv[3], points_, n_points_);
    printf("%d points\n", n_points_);
    return 0;
}
```

D.3 compare.c

```
/*
 *  compare.c - compare .pnt files using hausdorff distance
 *
 *  Pedro Tejada
 *  Thu May 28 20:50:40 MDT 2009
 */

#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

s_point *points1;
int n_points1;

s_point *points2;
int n_points2;

double hausdorff(s_point *points1, int n_points1, s_point *points2, int n_points2) {
    double h = 0.0;
    int i, j;

    if (n_points1 == 0 || n_points2 == 0)
        return DBL_MAX;

    for (i = 0; i < n_points1; i++) {
        double shortest = DBL_MAX;

        for (j = 0; j < n_points2; j++)
            shortest = fmin(shortest, dist(&points1[i], &points2[j]));
        h = fmax(shortest, h);
    }
    return h;
}

int main(int argc, char *argv[]) {
    double h12, h21, h;

    if (argc != 3) {
        fprintf(stderr, "usage: %s pnt_input_file_1 pnt_input_file_2\n", argv[0]);
        exit(1);
    }
    points1 = input_points(argv[1], &n_points1);
    points2 = input_points(argv[2], &n_points2);
    h12 = hausdorff(points1, n_points1, points2, n_points2);
    h21 = hausdorff(points2, n_points2, points1, n_points1);
    h = fmax(h12, h21);
    if (h == DBL_MAX)
        printf("%s\t%s\t%s\t%s\t%s\n", "inf", "inf", "inf", argv[1], argv[2]);
    else
        printf("%g\t%g\t%g\t%s\t%s\n", h12, h21, h, argv[1], argv[2]);
    return 0;
}
```

D.4 stats.c

```
/*
 *  stats.c - average and standard deviation
 *
 *  Joel Gillespie, Minghui Jiang and Pedro Tejada
 *  Mon Mar  2 01:16:34 MST 2009
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

double *data;

int main() {
    double average = 0.0;
    double stdev = 0.0;
    int i, size, n = 0;

    size = 2;
    if ((data = malloc(size * sizeof(double))) == NULL) {
        fprintf(stderr, "main: malloc failed\n");
        exit(1);
    }
    while (scanf("%lf", &data[n]) != EOF) {
        n++;
        if (n == size) {
            double *t;

            size *= 2;
            if ((t = malloc(size * sizeof(double))) == NULL) {
                fprintf(stderr, "main: malloc failed\n");
                exit(1);
            }
            for (i = 0; i < n; i++)
                t[i] = data[i];
            free(data);
            data = t;
        }
    }

    for (i = 0; i < n; i++)
        average += data[i];
    average /= n;
    if (n > 1) {
        for (i = 0; i < n; i++) {
            double diff = data[i] - average;

            stdev += diff * diff;
        }
        stdev = sqrt(stdev / (n - 1));
    } else
        stdev = 0;
    printf("%.4f\t%.4f\n", average, stdev);
    return 0;
}
```

D.5 randomtests.definition

```
>> SET01 20
LINE 1
>> SET02 20
LINE 2
>> SET03 20
LINE 3
>> SET04 20
LINE 5
>> SET05 20
LINE 10
>> SET06 20
ELLIPSE 1
>> SET07 20
ELLIPSE 2
>> SET08 20
ELLIPSE 3
>> SET09 20
ELLIPSE 5
>> SET10 20
ELLIPSE 10
>> SET11 20
LINE 1
ELLIPSE 1
>> SET12 20
LINE 2
ELLIPSE 2
>> SET13 20
LINE 3
ELLIPSE 3
>> SET14 20
LINE 5
ELLIPSE 5
>> SET15 20
LINE 10
ELLIPSE 10
```

D.6 filter.awk

```
#
#   filter.awk - filter pseudobase.fasta
#
#   Pedro Tejada and Minghui Jiang
#   Tue Jul  7 03:07:55 MDT 2009
#

/^>>/ {
    testset = $2
}

/^> / {
    if (!testset)
        next

    testcase = $3
    close(command)
    command = "cat > " testset "_" testcase ".test1.case"
}

! /^>/ {
    if (!testset && !testcase)
        next

    print $0 | command
}
```

D.7 tcgen.c

```
/*
 *  tcgen.c - generate test cases from definition file
 *
 *  Pedro Tejada
 *  Fri Mar 27 23:43:15 MDT 2009
 */

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "lib.h"

#define S_LINE      "LINE"
#define S_ELLIPSE   "ELLIPSE"

#define I_LINE      0
#define I_ELLIPSE   1

void generate_line(FILE *out) {
    double x1 = RAND(1.0);
    double y1 = RAND(1.0);
    double x2 = RAND(1.0);
    double y2 = RAND(1.0);

    fprintf(out, S_LINE " %g %g %g %g\n", x1, y1, x2, y2);
}

void generate_ellipse(FILE *out) {
    double x = RAND(1.0);
    double y = RAND(1.0);
    double a = RAND(0.5); /* semi-axis */
    double b = RAND(0.5); /* semi-axis */

    fprintf(out, S_ELLIPSE " %g %g %g %g\n", x, y, a, b);
}

void generate_set_cases(FILE *in, FILE *out) {
    char test_set_name[64];
    int n_test_cases;
    int obj_counts[2];
    char c;
    int i, j, k;

    /* Basic set information */
    if ((c = getc(in)) != '>' || (c = getc(in)) != '>') {
        fprintf(stderr, "generate_set_cases: file has wrong format 1\n");
        exit(1);
    }
    if (fscanf(in, "%s %d\n", test_set_name, &n_test_cases) != 2) {
        fprintf(stderr, "generate_set_cases: file has wrong format 2\n");
        exit(1);
    }
    fprintf(out, ">> %s %d\n", test_set_name, n_test_cases);
```

```
    /* Read test case description */
    obj_counts[0] = obj_counts[1] = 0;
    while ((c = getc(in)) != EOF && ungetc(c, in) && c != '>') {
        char obj_type[64];
        int obj_count;

        if (fscanf(in, "%s %d\n", obj_type, &obj_count) != 2) {
            fprintf(stderr, "generate_set_cases: file has wrong format\n");
            exit(1);
        }
        if (!strcmp(obj_type, S_LINE))
            obj_counts[I_LINE] += obj_count;
        else if (!strcmp(obj_type, S_ELLIPSE))
            obj_counts[I_ELLIPSE] += obj_count;
        else {
            fprintf(stderr, "generate_set_cases: file has wrong format\n");
            exit(1);
        }
    }

    /* Generate test cases */
    for (i = 0; i < n_test_cases; i++) {
        fprintf(out, "> CASE %d\n", (i + 1));
        for (j = 0; j < 2; j++)
            for (k = 0; k < obj_counts[j]; k++)
                switch (j) {
                    case I_LINE:
                        generate_line(out);
                        break;
                    case I_ELLIPSE:
                        generate_ellipse(out);
                        break;
                }
    }
}

void generate_test_cases(char *input_file, char *output_file, int seed) {
    FILE *in, *out;
    char c;

    srand(seed);
    if ((in = fopen(input_file, "r")) == NULL) {
        fprintf(stderr, "generate_test_cases: fopen(%s) failed\n", input_file);
        exit(1);
    }
    if ((out = fopen(output_file, "w")) == NULL) {
        fprintf(stderr, "generate_test_cases: fopen(%s) failed\n", input_file);
        exit(1);
    }
    while ((c = getc(in)) != EOF && ungetc(c, in))
        generate_set_cases(in, out);
    fclose(in);
    fclose(out);
}
```

```
void print_help_and_exit(int argc, char *argv[]) {
    fprintf(stderr, "usage:  %s [options]\n" "options:\n"
            "\t-i\t<ppm_input_filename>\n"
            "\t-o\t<pnt_output_filename>\n"
            "\t-s\t<random_seed>\n",
        argv[0]);
    exit(0);
}

int main(int argc, char *argv[]) {
    char *input_filename = NULL;
    char *output_filename = NULL;
    int seed;
    int i;

    seed = (int) time(NULL);
    for (i = 1; i < argc; i++)
        if (!strcmp(argv[i], "-i")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                input_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-o")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                output_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-s")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                seed = atoi(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else
            print_help_and_exit(argc, argv);
    if (!input_filename || !output_filename)
        print_help_and_exit(argc, argv);

    generate_test_cases(input_filename, output_filename, seed);
    return 0;
}
```

D.8 tcapx.c

```
/*
 *  tcapx.c - generate point set approximation of test case
 *
 *  Pedro Tejada
 *  Mon Apr  6 09:57:23 MDT 2009
 */

#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "lib.h"

#define S_LINE      "LINE"
#define S_ELLIPSE   "ELLIPSE"

double dist_xy(double x1, double y1, double x2, double y2) {
    double dx = x1 - x2;
    double dy = y1 - y2;

    return sqrt(dx * dx + dy * dy);
}

int n_points_line(double x1, double y1, double x2, double y2, double delta) {
    double d = dist_xy(x1, y1, x2, y2);
    int n = 2;

    if (delta < d)
        n += ceil(d / delta) - 1;
    return n;
}

s_point *line(double x1, double y1, double x2, double y2, int n) {
    s_point *points;
    double dx, dy;  /* distances between points in line */
    double x, y, a;
    int i;

    if ((points = malloc(sizeof(s_point) * n)) == NULL) {
        fprintf(stderr, "line: malloc failed\n");
        exit(1);
    }
    a = atan((y2 - y1) / (x2 - x1));
    if ((a = fmod(a, M_PI) / M_PI) < 0.0)
        a += 1.0;
    dx = (x2 - x1) / (n - 1);
    dy = (y2 - y1) / (n - 1);

    /* first point */
    points[0] = *new_point(x1, y1, a, 1.0);

    /* middle points */
    for (i = 1; i < n-1; i++) {
```

```
        x = x1 + i * dx;
        y = y1 + i * dy;
        points[i] = *new_point(x, y, a, 1.0);
    }

    /* last point */
    points[n-1] = *new_point(x2, y2, a, 1.0);
    return points;
}

int n_points_ellipse(double x, double y, double a, double b, double delta) {

    /* approximation of circumference */
    double c = M_PI * (3 * (a + b) - sqrt((3 * a + b) * (a + 3 * b)));
    return 2 * ceil(c / delta);
}

/*
 * See:
 * http://en.wikipedia.org/wiki/Ellipse
 * http://mathworld.wolfram.com/Ellipse.html
 */
s_point *ellipse(double x, double y, double a, double b, int n) {
    s_point *points;
    double alpha;   /* angle arround center */
    int i_point = 0;
    int i;

    if ((points = malloc(sizeof(s_point) * n)) == NULL) {
        fprintf(stderr, "ellipse: malloc failed\n");
        exit(1);
    }
    for (i = 0; i < n; i++) {
        double px, py, pa;  /* point coordinates and direction */
        double xt, yt;

        alpha = (double) i * (2.0 * M_PI) / n;
        px = x + a * cos(alpha);
        py = y + b * sin(alpha);

        /* determine slope of tangent line at point */
        xt = - (a * sin(alpha));
        xt /= sqrt(b * b * cos(alpha) * cos(alpha)
                    + a * a * sin(alpha) * sin(alpha));
        yt = (b * cos(alpha));
        yt /= sqrt(b * b * cos(alpha) * cos(alpha)
                    + a * a * sin(alpha) * sin(alpha));
        pa = atan(yt / xt);
        if ((pa = fmod(pa, M_PI) / M_PI) < 0.0)
            pa += 1.0;

        points[i_point++] = *new_point(px, py, pa, 1.0);
    }
    return points;
}
```

```
int out_of_range(s_point *point) {
    return point->x < 0.0 || point->x > 1.0
            || point->y < 0.0 || point->y > 1.0;
}

void remove_out_of_range(s_point *points, int *n_points) {
    int i, k;

    k = 0;
    for (i = 0; i < *n_points; i++)
        if (!out_of_range(&points[i])) {
            if (k < i)
                points[k] = points[i];
            k++;
        }
    *n_points = k;
}

void generate_approx(char *input_file, char *output_file, int img_size) {
    FILE *in;
    s_point_array *pnt_array;
    double delta = 1.0 / img_size; /* max space between points */

    if ((in = fopen(input_file, "r")) == NULL) {
        fprintf(stderr, "generate_approx: fopen(%s) failed\n", input_file);
        exit(1);
    }
    pnt_array = new_point_array(0);

    while (!feof(in)) {
        char obj_type[64];
        s_point *points_;
        int n_points_;

        if (fscanf(in, "%s", obj_type) != 1) {
            fprintf(stderr, "generate_approx: file has wrong format\n");
            exit(1);
        }
        if (!strcmp(obj_type, S_LINE)) {
            double x1, y1, x2, y2;

            if (fscanf(in, "%lf %lf %lf %lf\n", &x1, &y1, &x2, &y2) != 4) {
                fprintf(stderr, "generate_approx: file has wrong format\n");
                exit(1);
            }
            n_points_ = n_points_line(x1, y1, x2, y2, delta);
            points_ = line(x1, y1, x2, y2, n_points_);
            extend_point_array(pnt_array, points_, n_points_);

        } else if (!strcmp(obj_type, S_ELLIPSE)) {
            double x, y, a, b;

            if (fscanf(in, "%lf %lf %lf %lf\n", &x, &y, &a, &b) != 4) {
                fprintf(stderr, "generate_approx: file has wrong format\n");
                exit(1);
            }
```

```
            n_points_ = n_points_ellipse(x, y, a, b, delta);
            points_ = ellipse(x, y, a, b, n_points_);
            extend_point_array(pnt_array, points_, n_points_);

        } else {
            fprintf(stderr, "generate_approx: file has wrong format\n");
            exit(1);
        }
    }
    fclose(in);
    remove_out_of_range(pnt_array->points, &pnt_array->n);
    output_points(output_file, pnt_array->points, pnt_array->n);
    printf("%d points\n", pnt_array->n);
}

void print_help_and_exit(int argc, char *argv[]) {
    fprintf(stderr, "usage:  %s [options]\n" "options:\n"
            "\t-i\t<ppm_input_filename>\n"
            "\t-o\t<pnt_output_filename>\n"
            "\t-s\t<image_size>\n",
        argv[0]);
    exit(0);
}

int main(int argc, char *argv[]) {
    char *input_filename = NULL;
    char *output_filename = NULL;
    int img_size = 512;
    int i;

    for (i = 1; i < argc; i++)
        if (!strcmp(argv[i], "-i")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                input_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-o")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                output_filename = argv[++i];
            else
                print_help_and_exit(argc, argv);
        } else if (!strcmp(argv[i], "-s")) {
            if (i + 1 < argc && argv[i + 1][0] != '-')
                img_size = atoi(argv[++i]);
            else
                print_help_and_exit(argc, argv);
        } else
            print_help_and_exit(argc, argv);
    if (!input_filename || !output_filename)
        print_help_and_exit(argc, argv);

    generate_approx(input_filename, output_filename, img_size);
    return 0;
}
```

D.9 bimcpnt.c

```
/*
 *  bimcpnt.c - extract contour points from binary image
 *
 *  Pedro Tejada
 *  Sun Jun 28 01:23:20 MDT 2009
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "lib.h"

#define CM  3
#define CN  3

/* structure element */
int SE[CM][CN] = {{0, 1, 0},
                  {1, 1, 1},
                  {0, 1, 0}};

int check_pixel(s_image *img, int x, int y) {
    int i, j;

    if (img->p[x][y].r < MAX_VALUE / 2)
        return 0;   /* black pixel */
    for (i = 0; i < CM; i++)
        for (j = 0; j < CN; j++) {
            int x_ = x + (CM / 2 - i);
            int y_ = y + (CN / 2 - j);
            int pix = img->p[x_][y_].r >= MAX_VALUE / 2; /* white pixel */

            if (SE[i][j] && !pix)
                return 1;
        }
    return 0;
}

s_point *extract_points_(s_image *img, int *n_points) {
    s_point *points;
    int np;
    int i, j;

    if ((points = malloc(sizeof(s_point) * img->w * img->h)) == NULL) {
        fprintf(stderr, "extract_points: malloc failed\n");
        exit(1);
    }
    np = 0;
    for (i = 1; i < img->w - 1; i++)
        for (j = 1; j < img->h - 1; j++) {
            if (check_pixel(img, i, j)) {
                s_point *p = &points[np++];

                p->x = get_point_x(img->w, img->h, i);
                p->y = get_point_y(img->w, img->h, j);
                p->a = 0.0;
```

```
                p->w = 1.0;
            }
        }
    *n_points = np;
    return points;
}

int main(int argc, char *argv[]) {
    s_point *points;
    int n_points;
    s_image *img;

    if (argc != 3) {
        fprintf(stderr, "usage: %s ppm_file_name pnt_file_name\n", argv[0]);
        exit(1);
    }
    img = load_ppm(argv[1]);
    points = extract_points_(img, &n_points);
    output_points(argv[2], points, n_points);
    printf("%d points\n", n_points);
    return 0;
}
```

Appendix E

Makefiles

E.1 Makefile.generic

```
#
#   Makefile.generic - platform-independent Makefile
#
#   Pedro Tejada and Minghui Jiang
#   Tue Jul  7 02:01:52 MDT 2009
#

SRC1 = cluster.c lib.c lib.h link.c pth2fig.c show.c simplify.c sobel.c
SRC2 = bimcpnt.c compare.c discretize.c noise.c tcapx.c tcgen.c stats.c
SRC3 = filter.awk gif2jpg.m jpg2ppm.java jpg2ppm.manifest \
        randomtests.definition
MAK = Makefile.Cygwin Makefile.Linux Makefile.MacOSX Makefile.generic
JPG = cameraman.jpg jetplane.jpg lena.jpg livingroom.jpg peppers.jpg \
        rose.jpg text.jpg walkbridge.jpg woman_blonde.jpg

HDR = lib.h
OBJ = lib.o
JAR = jpg2ppm.jar
EXE1 = cluster.exe link.exe pth2fig.exe show.exe simplify.exe sobel.exe
EXE2 = bimcpnt.exe compare.exe discretize.exe noise.exe \
        tcgen.exe tcapx.exe stats.exe
ALL = $(JAR) $(EXE1) $(EXE2)

.PHONY: all clean clobber spotless

all: $(ALL)
%.clean:
    rm -f $**.ppm $**.pnt $**.pth $**.fig $**.eps $**.pdf
clean: *.clean
clobber: clean
    rm -f $(ALL) *.class *.o all.zip
spotless: clobber test.spotless

lib.o: lib.h
%.o: %.c $(HDR)
    cc -o $@ -c $< $(CFLAGS)
%.exe: %.c $(OBJ)
    cc -o $@ $^ $(CFLAGS) $(LDFLAGS)
show.exe: show.c $(OBJ)
    cc -o $@ $^ $(CFLAGS) $(LDFLAGS) $(GLFLAGS)
%.jar: %.java
    javac $<
    jar cmf $*.manifest $@ *.class
```

```
    rm -f *.class

##
#   Contour extraction and simplification
#       .ppm        (grayscale .ppm image from .jpg)
#       .pnt        (sobel points from .ppm)
#       .c.pnt      (clustered points: .pnt)
#       .pth        (linked points: pth)
#       .s.pnt      (simplified points: .pnt)
#       .s.pth      (simplified paths: .pth)
##

# default param values for jpg2ppm.c
J2PG_PARAM = -g
ifeq ($(J2PG),0)
    J2PG_PARAM =
endif
%.ppm: %.jpg
    java -jar jpg2ppm.jar $< $@ $(J2PG_PARAM)

# default param values for sobel.c
ST = -1.0
SN = 20000
SF = 0
%.pnt: %.ppm
    ./sobel.exe -i $< -o $@ -n $(SN) -t $(ST) -f $(SF)

# default param values for cluster.c
CD = -1.0
CR = 2.0
%.c.pnt: %.pnt
    ./cluster.exe -i $< -o $@ -d $(CD) -r $(CR)

# default param values for link.c
LD = -1.0
LR = 3.0
LA = 35.0
%.s.pth: %.s.pnt ;
%.pth: %.pnt
    ./link.exe -i $< -o $@ -d $(LD) -r $(LR) -a $(LA)

# default param values for simplify.c
ZD = -1.0
ZF = 0.5
ZML = 0.0
ZRI = 1
%.s.pnt: %.pth
    ./simplify.exe -i1 $*.pnt -i2 $*.pth \
        -o1 $*.s.pnt -o2 $*.s.pth \
        -d $(ZD) -df $(ZF) -ml $(ZML) -ri $(ZRI)

##
#   File format
#       .fig    (.fig from .pth)
#       .eps    (.eps from .fig)
#       .pdf    (.pdf from .eps)
```

```
##

# default param values for pth2fig.c
FS_PARAM =
ifeq ($(FS),1)
    FS_PARAM = -s
endif
FP_PARAM =
ifeq ($(FP),1)
    FP_PARAM = -p
endif
%.fig: %.pth
    ./pth2fig.exe $*.pnt $*.pth $*.fig $(FS_PARAM) $(FP_PARAM)
%.eps: %.fig
    fig2dev -L eps $< $@
%.pdf: %.eps
    epstopdf $<

##
#   Display
#       .jpg.show   (.jpg image)
#       .ppm.show   (.ppm image)
#       .pnt.show   (sobel points: .pnt)
#       .pth.show   (paths: .pth)
#       .c.show     (clustered points: .pnt)
#       .l.show     (linked points: .pth)
#       .s.show     (simplified paths: .pth)
#       .show       (default for points: .pnt)
##

IMG_PARAM =
ifdef IMG
    IMG_PARAM = -i $(IMG)
endif

PNT2_PARAM =
ifdef PNT2
    PNT2_PARAM = -p2 $(PNT2)
endif

%.jpg.show: %.jpg
    java -jar jpg2ppm.jar $<
%.ppm.show: %.ppm
    java -jar jpg2ppm.jar $<
%.pnt.show: %.pnt
    ./show.exe -p $< $(IMG_PARAM) $(PNT2_PARAM)
%.pth.show: %.pnt %.pth
    ./show.exe -p $< -P $*.pth $(IMG_PARAM) $(PNT2_PARAM)
%.c.show: %.c.pnt
    ./show.exe -p $< $(IMG_PARAM) $(PNT2_PARAM)
%.l.show: %.pth
    ./show.exe -p $*.pnt -P $*.pth $(IMG_PARAM) $(PNT2_PARAM)
%.s.show: %.s.pth
    ./show.exe -p $*.s.pnt -P $*.s.pth $(IMG_PARAM) $(PNT2_PARAM)
%.show: %.pnt
    ./show.exe -p $< $(IMG_PARAM) $(PNT2_PARAM)
```

```
##
#   Tests
#       test.clean test.clobber test.spotless
#       .d.pnt      (discretize points)
#       .noise.ppm  (add noise to .ppm image)
#       .noise.pnt  (add noise to .ppm file)
#       .compare    (compare expected and discretized results)
##

.PHONY: test.clean test.clobber test.spotless

SEED = 5050     # default seed for random number generator
IMG_SIZE = 512  # default image size
ND = 0.05       # default noise density
NN = 1000       # default noise points

test.clean: test1.clean test2.clean test3.clean ;
test.clobber: test1.clobber test2.clobber test3.clobber ;
test.spotless: test1.spotless test2.spotless test3.spotless ;

%.d.pnt: %.pnt %.pth
    ./discretize.exe $*.pnt $*.pth $@

%.noise.ppm: %.ppm
    ./noise.exe $< $@ -ppm -s $(SEED) -d $(ND)

%.noise.pnt: %.pnt
    ./noise.exe $< $@ -pnt -s $(SEED) -n $(NN)

%.compare:
    ./compare.exe $*.res.d.pnt $*.exp.pnt

##
#   Test 1: random shapes
#       test1.clean     (remove test files, leave zipped files)
#       test1.clobber   (remove test directory)
#       test1.spotless  (remove all test files)
#       test1.cases     (generate test cases)
#       .test1.exp.pnt  (expected file: .pnt)
#       .test1.src.pnt  (source file: .pnt)
#       .test1.res.pnt  (result files: .pnt, .pth)
#       .test1          (run test for specific file)
#       test1.expected  (generate all expected files)
#       test1.source    (generate all source files)
#       test1.result    (generate all result files)
#       test1.d.result  (generate all discretized results)
#       test1.txt       (run tests on all files: output to .txt)
#       %.test1.cnt     (count number of paths)
#       test1.cnt       (count number of paths)
#       test1.sts       (extract statistical information)
#       test1           (run tests)
#       test1.all       (run all tests)
##

.PHONY: test1.clean test1.clobber test1.spotless
```

```
.PHONY: test1.cases.all test1.cases
.PHONY: test1.expected test1.source test1.result test1.d.result
.PHONY: test1.txt test1.cnt test1.sts test1 test1.all

# test definition file, test cases file, test directory, and results directory
TEST1_DEF = randomtests.definition
TEST1_TST = test1.cases.all
TEST1_DIR = test1
TEST1_RES = results

test1.clean:
    rm -f *.test1.case
    ls -l -d $(TEST1_DIR)/* | grep -v \.zip$ | awk '{system("rm -fr " $$9)}'
test1.clobber:
    rm -fr $(TEST1_DIR)
test1.spotless: test1.clobber
    rm -f $(TEST1_TST) test1*.txt test1*.cnt test1*.sts

$(TEST1_TST): $(TEST1_DEF)
    ./tcgen.exe -i $< -o $(TEST1_TST) -s $(SEED)

test1.cases: $(TEST1_TST)
    awk -f filter.awk $<

%.test1.exp.pnt: %.test1.case
    ./tcapx.exe -i $< -o $@ -s $(IMG_SIZE)

%.test1.src.pnt: %.test1.exp.noise.pnt
    mv $< $@

%.test1.res.pnt: %.test1.src.c.s.pth
    mv $*.test1.src.c.s.pnt $*.test1.res.pnt
    mv $*.test1.src.c.s.pth $*.test1.res.pth

%.test1:
    make $*.test1.exp.pnt
    make $*.test1.src.pnt
    make $*.test1.res.pnt
    make $*.test1.res.d.pnt
    make $*.test1.compare

test1.expected:
    ls $(TEST1_DIR)/*.test1.case | sed -e 's/test1.case/test1.exp.pnt/' \
        | awk '{system("make -s " $$0)}'

test1.source:
    ls $(TEST1_DIR)/*.test1.case | sed -e 's/test1.case/test1.src.pnt/' \
        | awk '{system("make -s " $$0)}'

test1.result:
    ls $(TEST1_DIR)/*.test1.case | sed -e 's/test1.case/test1.res.pnt/' \
        | awk '{system("make -s " $$0)}'

test1.d.result:
    ls $(TEST1_DIR)/*.test1.case | sed -e 's/test1.case/test1.res.d.pnt/' \
        | awk '{system("make -s " $$0)}'
```

```
test1.txt: test1.expected test1.source test1.result test1.d.result
    rm -f $@
    echo -n "" > $@
    ls $(TEST1_DIR)/*.test1.case | sed -e 's/test1.case/test1.compare/' \
        | awk '{system("make -s " $$0)}' >> $@

%.test1.cnt: %.test1.res.pth
    awk '{if (NR == 1) print $$1 " $*"}' $<

test1.cnt:
    rm -f $@
    echo -n "" > $@
    ls $(TEST1_DIR)/*.test1.res.pth | sed -e 's/test1.res.pth/test1.cnt/' \
        | awk '{system("make -s " $$0)}' >> $@
    awk '{print $$1"\n"}' $@ | ./stats.exe

test1.sts: stats.exe
    rm -f $@
    echo -n "Total pictures:         " > $@
    cat test1.txt | wc -l >> $@
    echo -n "With paths detected:    " >> $@
    awk '{if ($$3 < 2.0) print $$3" "}' test1.txt | wc -w >> $@
    echo -n "With no paths detected: " >> $@
    awk '{if ($$3 > 2.0) print $$3"\n"}' test1.txt | wc -w >> $@
    echo "Hausdorf distances:" >> $@
    awk '{if ($$3 < 2.0) print $$1"\n"}' test1.txt | ./stats.exe >> $@
    awk '{if ($$3 < 2.0) print $$2"\n"}' test1.txt | ./stats.exe >> $@
    awk '{if ($$3 < 2.0) print $$3"\n"}' test1.txt | ./stats.exe >> $@
    echo "Average number of paths:" >> $@
    awk '{if ($$3 < 2.0) print $$1"\n"}' test1.cnt | ./stats.exe >> $@
    cat $@

test1: test1.txt test1.cnt test1.sts
    mkdir $(TEST1_DIR)/$(TEST1_RES)
    mv $(TEST1_DIR)/*.src.pnt $(TEST1_DIR)/$(TEST1_RES)
    mv $(TEST1_DIR)/*.res.pnt $(TEST1_DIR)/$(TEST1_RES)
    mv $(TEST1_DIR)/*.res.pth $(TEST1_DIR)/$(TEST1_RES)
    mv $(TEST1_DIR)/*.res.d.pnt $(TEST1_DIR)/$(TEST1_RES)
    mv test1.txt test1_$(TEST1_RES).txt
    mv test1.cnt test1_$(TEST1_RES).cnt
    mv test1.sts test1_$(TEST1_RES).sts
    zip -r $(TEST1_DIR)/test1_$(TEST1_RES).zip $(TEST1_DIR)/$(TEST1_RES)

test1.all: test1.spotless test1.cases
    mkdir $(TEST1_DIR)
    mv *.test1.case $(TEST1_DIR)
    cp $(TEST1_DEF) $(TEST1_TST) $(TEST1_DIR)
    make -s test1.expected
    make -s test1 TEST1_RES=0_0 NN=0 ZML=0.00 CD=0.002
    make -s test1 TEST1_RES=0_5 NN=0 ZML=0.05 CD=0.002
    make -s test1 TEST1_RES=5_0 NN=5000 ZML=0.00 CD=0.002
    make -s test1 TEST1_RES=5_5 NN=5000 ZML=0.05 CD=0.002
    make -s test1 TEST1_RES=10_0 NN=10000 ZML=0.00 CD=0.002
    make -s test1 TEST1_RES=10_5 NN=10000 ZML=0.05 CD=0.002
    mkdir $(TEST1_DIR)/testcases
```

```
    mv $(TEST1_DIR)/$(TEST1_DEF) $(TEST1_DIR)/$(TEST1_TST) $(TEST1_DIR)/testcases
    mv $(TEST1_DIR)/*.test1.case $(TEST1_DIR)/testcases
    zip -r $(TEST1_DIR)/test1_testcases.zip $(TEST1_DIR)/testcases
    mkdir $(TEST1_DIR)/exp
    mv $(TEST1_DIR)/*.exp.pnt $(TEST1_DIR)/exp
    zip -r $(TEST1_DIR)/test1_exp.zip $(TEST1_DIR)/exp
    cp test1_*.txt test1_*.cnt test1_*.sts $(TEST1_DIR)
    zip -r $(TEST1_DIR)/test1_results.zip \
        $(TEST1_DIR)/test1_*.txt $(TEST1_DIR)/test1_*.cnt $(TEST1_DIR)/test1_*.sts

##
#   Test 2: binary images
#       test2.clean     (remove test files, leave zipped files)
#       test2.clobber   (remove test directory)
#       test2.spotless  (remove all test files)
#       .test2.exp.pnt  (expected file: .pnt)
#       .test2.src.ppm  (source file: .ppm)
#       .test2.res.pth  (result files: .pnt, .pth)
#       .test2          (run test for specific file)
#       test2.jpg       (copy all jpg files to test directory)
#       test2.ppm       (generate all ppm files)
#       test2.expected  (generate all expected files)
#       test2.source    (generate all source files)
#       test2.result    (generate all result files)
#       test2.d.result  (generate all discretized results)
#       test2.txt       (run tests on all files: output to .txt)
#       %.test2.cnt     (count number of paths)
#       test2.cnt       (count number of paths)
#       test2.sts       (extract statistical information)
#       test2           (run tests)
#       test2.all       (run all tests)
##

.PHONY: test2.clean test2.clobber test2.spotless
.PHONY: test2.ppm test2.expected test2.source test2.result test2.d.result
.PHONY: test2.txt test2.cnt test2.sts test2 test2.all

# dataset, test, and results directories
TEST2_DAT = mpeg7
TEST2_DIR = test2
TEST2_RES = results

test2.clean:
    ls -l -d $(TEST2_DIR)/* | grep -v \.zip$ | awk '{system("rm -fr " $$9)}'
test2.clobber:
    rm -fr $(TEST2_DIR)
test2.spotless: test2.clobber
    rm -f test2*.txt test2*.cnt test2*.sts

%.test2.exp.pnt: %.ppm
    ./bimcpnt.exe $< $@

%.test2.src.ppm: %.noise.ppm
    mv $< $@

%.test2.res.pth: %.test2.src.c.s.pth
```

```
380     mv $*.test2.src.c.s.pnt $*.test2.res.pnt
        mv $*.test2.src.c.s.pth $*.test2.res.pth

    %.test2:
        make $*.test2.exp.pnt
        make $*.test2.src.ppm
        make $*.test2.res.pth
        make $*.test2.res.d.pnt
        make $*.test2.compare

390 test2.jpg:
        cp $(TEST2_DAT)/*.jpg $(TEST2_DIR)

    test2.ppm:
        ls $(TEST2_DIR)/*.jpg | sed -e 's/jpg/ppm/' \
            | awk '{system("make -s " $$0)}'

    test2.expected:
        ls $(TEST2_DIR)/*.jpg | sed -e 's/jpg/test2.exp.pnt/' \
            | awk '{system("make -s " $$0)}'
400
    test2.source:
        ls $(TEST2_DIR)/*.jpg | sed -e 's/jpg/test2.src.ppm/' \
            | awk '{system("make -s " $$0)}'

    test2.result:
        ls $(TEST2_DIR)/*.jpg | sed -e 's/jpg/test2.res.pth/' \
            | awk '{system("make -s " $$0)}'

    test2.d.result:
410     ls $(TEST2_DIR)/*.jpg | sed -e 's/jpg/test2.res.d.pnt/' \
            | awk '{system("make -s " $$0)}'

    test2.txt: test2.source test2.result test2.d.result
        rm -f $@
        echo -n "" > $@
        ls $(TEST2_DIR)/*.jpg | sed -e 's/jpg/test2.compare/' \
            | awk '{system("make -s " $$0)}' >> $@

    %.test2.cnt: %.test2.res.pth
420     awk '{if (NR==1) print $$1 " $*"}' $<

    test2.cnt:
        rm -f $@
        echo -n "" > $@
        ls $(TEST2_DIR)/*.test2.res.pth | sed -e 's/test2.res.pth/test2.cnt/' \
            | awk '{system("make -s " $$0)}' >> $@
        awk '{print $$1"\n"}' $@ | ./stats.exe

    test2.sts: stats.exe
430     rm -f $@
        echo -n "Total pictures:         " > $@
        cat test2.txt | wc -l >> $@
        echo -n "With paths detected:    " >> $@
        awk '{if ($$3 < 2.0) print $$3" "}' test2.txt | wc -w >> $@
        echo -n "With no paths detected: " >> $@
```

```
        awk '{if ($$3 > 2.0) print $$3"\n"}' test2.txt | wc -w >> $@
        echo "Hausdorf distances:" >> $@
        awk '{if ($$3 < 2.0) print $$1"\n"}' test2.txt | ./stats.exe >> $@
        awk '{if ($$3 < 2.0) print $$2"\n"}' test2.txt | ./stats.exe >> $@
        awk '{if ($$3 < 2.0) print $$3"\n"}' test2.txt | ./stats.exe >> $@
        echo "Average number of paths:" >> $@
        awk '{if ($$3 < 2.0) print $$1"\n"}' test2.cnt | ./stats.exe >> $@
        cat $@

    test2: test2.txt test2.cnt test2.sts
        mkdir $(TEST2_DIR)/$(TEST2_RES)
        mv $(TEST2_DIR)/*.src.ppm $(TEST2_DIR)/$(TEST2_RES)
        mv $(TEST2_DIR)/*.res.pnt $(TEST2_DIR)/$(TEST2_RES)
        mv $(TEST2_DIR)/*.res.pth $(TEST2_DIR)/$(TEST2_RES)
        mv $(TEST2_DIR)/*.res.d.pnt $(TEST2_DIR)/$(TEST2_RES)
        mv test2.txt test2_$(TEST2_RES).txt
        mv test2.cnt test2_$(TEST2_RES).cnt
        mv test2.sts test2_$(TEST2_RES).sts
        zip -r $(TEST2_DIR)/test2_$(TEST2_RES).zip $(TEST2_DIR)/$(TEST2_RES)

    test2.all: test2.spotless
        mkdir $(TEST2_DIR)
        make -s test2.jpg
        make -s test2.ppm
        make -s test2.expected
        make -s test2 TEST2_RES=0_0 ND=0.00 ZML=0.00
        make -s test2 TEST2_RES=0_5 ND=0.00 ZML=0.05
        make -s test2 TEST2_RES=2_0 ND=0.02 ZML=0.00
        make -s test2 TEST2_RES=2_5 ND=0.02 ZML=0.05
        make -s test2 TEST2_RES=5_0 ND=0.05 ZML=0.00
        make -s test2 TEST2_RES=5_5 ND=0.05 ZML=0.05
        mkdir $(TEST2_DIR)/jpg
        mv $(TEST2_DIR)/*.jpg $(TEST2_DIR)/jpg
        zip -r $(TEST2_DIR)/test2_jpg.zip $(TEST2_DIR)/jpg
        mkdir $(TEST2_DIR)/ppm
        mv $(TEST2_DIR)/*.ppm $(TEST2_DIR)/ppm
        zip -r $(TEST2_DIR)/test2_ppm.zip $(TEST2_DIR)/ppm
        mkdir $(TEST2_DIR)/exp
        mv $(TEST2_DIR)/*.exp.pnt $(TEST2_DIR)/exp
        zip -r $(TEST2_DIR)/test2_exp.zip $(TEST2_DIR)/exp
        cp test2_*.txt test2_*.cnt test2_*.sts $(TEST2_DIR)
        zip -r $(TEST2_DIR)/test2_results.zip \
            $(TEST2_DIR)/test2_*.txt $(TEST2_DIR)/test2_*.cnt $(TEST2_DIR)/test2_*.sts

    ##
    #   Test 3: natural images
    #       test3.clean     (remove test files, leave zipped files)
    #       test3.clobber   (remove test directory)
    #       test3.spotless  (remove all test files)
    #       .test3          (run test for specific file)
    #       test3.txt       (run tests on all files: output to .txt)
    #       test3.sts       (extract statistical information)
    #       test3           (run all tests)
    #       test3.all       (run all tests)
    ##
```

```
.PHONY: test3.clean test3.clobber test3.spotless
.PHONY: test3.txt test3.sts test3 test3.all

TEST3_DIR = test3

test3.clean:
    ls -l -d $(TEST3_DIR)/* | grep -v \.zip$ | awk '{system("rm -fr " $$9)}'
test3.clobber:
    rm -fr $(TEST3_DIR)
test3.spotless: test3.clobber
    rm -f test3.txt test3.sts

%.test3:
    rm -f $@
    echo -n "" > $@
    make -s $*.ppm
    make -s $*.pnt
    make -s $*.c.pnt
    make -s $*.c.s.pnt
    make -s $*.c.s.pth
    awk '{if (NR > 1 && $$1 != "#") {printf $$1"x"$$2 " " ; exit 0}}' $*.ppm >> $@
    awk '{if (NR == 1) printf $$1 " "}' $*.pnt >> $@
    awk '{if (NR == 1) printf $$1 " "}' $*.c.pnt >> $@
    awk '{if (NR == 1) printf $$2 " " $$1 " "}' $*.c.s.pth >> $@
    echo $*.jpg >> $@

test3.txt:
    rm -f $@
    echo -n "" > $@
    # Input:  (1) image size, (2) sobel points, (3) cluster points,
    #         (4) simp points, (5) simp paths, (6) file name
    # Output: (1) image size, (2) sobel points, (3) cluster points, (4) cluster %,
    #         (5) simp points, (6) simp %, (7) simp paths, (8) file name
    ls $(TEST3_DIR)/*.test3 | awk '{system("cat " $$1)}' \
        | awk '{printf $$1 "\t" $$2 "\t" $$3 "\t" $$2/$$3 "\t" $$4 \
            "\t" $$2/$$4 "\t" $$5 "\t" $$6 "\n"}' >> $@

test3.sts: stats.exe
    rm -f $@
    echo -n "Sobel points  : " > $@
    awk '{print $$2"\n"}' test3.txt | ./stats.exe >> $@
    echo -n "Cluster points: " >> $@
    awk '{print $$3"\n"}' test3.txt | ./stats.exe >> $@
    echo -n "Cluster comp. : " >> $@
    awk '{print $$4"\n"}' test3.txt | ./stats.exe >> $@
    echo -n "Simp. points  : " >> $@
    awk '{print $$5"\n"}' test3.txt | ./stats.exe >> $@
    echo -n "Simp. comp.   : " >> $@
    awk '{print $$6"\n"}' test3.txt | ./stats.exe >> $@
    echo -n "Simp. paths   : " >> $@
    awk '{print $$7"\n"}' test3.txt | ./stats.exe >> $@

test3:
    ls $(TEST3_DIR)/*.jpg | sed -e 's/jpg/test3/' | awk '{system("make -s " $$0)}'
    make -s test3.txt
    make -s test3.sts
```

```
test3.all: test3.spotless
    mkdir $(TEST3_DIR)
    cp cameraman.jpg jetplane.jpg lena.jpg livingroom.jpg peppers.jpg \
        rose.jpg text.jpg walkbridge.jpg woman_blonde.jpg $(TEST3_DIR)
    make -s test3 SN=-1 ST=0.2
    cp test3.txt test3.sts $(TEST3_DIR)
    zip -r $(TEST3_DIR)/test3.zip $(TEST3_DIR)/*.jpg $(TEST3_DIR)/*.ppm \
            $(TEST3_DIR)/*.pnt $(TEST3_DIR)/*.pth
    zip -r $(TEST3_DIR)/test3_results.zip \
        $(TEST3_DIR)/test3.txt $(TEST3_DIR)/test3.sts

##
#   Software package and source code
#       all.zip code.pdf
##

all.zip: $(SRC1) $(SRC2) $(SRC3) $(MAK) $(JPG)
    zip -r $@ $^

%.java.ps: %.java
    enscript -MLetter -2r -Ec -T4 --header='$$n||$$%' -p $@ $< || true
%.manifest.ps: %.manifest
    enscript -MLetter -2r -Ec -T4 --header='$$n||$$%' -p $@ $< || true
%.h.ps: %.h
    enscript -MLetter -2r -Ec -T4 --header='$$n||$$%' -p $@ $< || true
%.c.ps: %.c
    enscript -MLetter -2r -Ec -T4 --header='$$n||$$%' -p $@ $< || true
%.awk.ps: %.awk
    enscript -MLetter -2r -Eawk -T4 --header='$$n||$$%' -p $@ $< || true
%.m.ps: %.awk
    enscript -MLetter -2r -Eawk -T4 --header='$$n||$$%' -p $@ $< || true
%.ps: %
    enscript -MLetter -2r -T4 --header='$$n||$$%' -p $@ $< || true
%.pdf: %.ps
    ps2pdf $<

code.pdf: Makefile.generic.pdf Makefile.Cygwin.pdf \
            Makefile.Linux.pdf Makefile.MacOSX.pdf \
            jpg2ppm.java.pdf jpg2ppm.manifest.pdf \
            bimcpnt.c.pdf cluster.c.pdf compare.c.pdf discretize.c.pdf \
            lib.c.pdf lib.h.pdf \
            link.c.pdf noise.c.pdf pth2fig.c.pdf show.c.pdf simplify.c.pdf \
            sobel.c.pdf stats.c.pdf tcgen.c.pdf tcapx.c.pdf \
            filter.awk.pdf gif2jpg.m.pdf randomtests.definition.pdf
    pdftk $^ output $@

##
#   Movie: process multiple images to create a movie
#       .movie.cnt  (find contours)
#       .movie.eps  (convert contours to .eps format)
#       .movie.jpg  (convert .eps to .jpg)
#       .movie      (generate .jpg's for every frame)
##

.PHONY: movie.clean
```

```
movie.clean:
    rm -f *.eps *.movie.jpg

%.movie.cnt:
    ls $**.jpg | sed -e 's/jpg/ppm/' | awk '{system("make -s " $$0)}'
    ls $**.jpg | sed -e 's/jpg/pnt/' | awk '{system("make -s " $$0)}'
    ls $**.jpg | sed -e 's/jpg/c.pnt/' | awk '{system("make -s " $$0)}'
    ls $**.jpg | sed -e 's/jpg/c.s.pnt/' | awk '{system("make -s " $$0)}'

%.movie.eps:
    ls $**.jpg | sed -e 's/jpg/c.s.eps/' | awk '{system("make -s " $$0)}'
    ls $**.c.s.eps | sed -e 's/.c.s.eps//' \
        | awk '{system("mv " $$0 ".c.s.eps" " " $$0 ".eps")}'

%.movie.jpg:
    ls $**.eps | sed -e 's/\.eps//' | \
        awk '{system("gs -sDEVICE=jpeg -dJPEGQ=100 -dNOPAUSE -dBATCH -dSAFER \
            -r300 -sOutputFile=" $$0 ".movie.jpg" " " $$0 ".eps")}'
    ls $**.movie.jpg | awk '{system("mogrify -trim " $$0)}'
    ls $**.movie.jpg | awk '{system("mogrify -resize 512x512 " $$0)}'

%.movie: %.movie.cnt %.movie.eps %.movie.jpg ;
```

E.2 Makefile.Cygwin

```
CFLAGS = -O3 -pedantic -Wall
GLFLAGS = -I/usr/include/w32api -lglut32 -lglu32 -lopengl32 -L/usr/lib/w32api
include Makefile.generic
```

E.3 Makefile.Linux

```
CFLAGS = -O3 -Wall
LDFLAGS = -lm
GLFLAGS = -lglut -lGLU -lGL
include Makefile.generic
```

E.4 Makefile.MacOSX

```
CFLAGS = -O3 -ansi -pedantic -Wall
GLFLAGS = -framework GLUT -framework OpenGL -framework Cocoa
include Makefile.generic
```