A Computational-Geometry Approach to Digital Image Contour Extraction*

Minghui Jiang, Xiaojun Qi, and Pedro J. Tejada

Department of Computer Science, Utah State University, Logan, Utah 84322-4205, USA
mjiang@cc.usu.edu, xiaojun.qi@usu.edu,
p.tejada@aggiemail.usu.edu

Abstract. We present a simple method based on computational-geometry for extracting contours from digital images. Unlike traditional image processing methods, our proposed method first extracts a set of oriented feature points from the input images, then applies a sequence of geometric techniques, including clustering, linking, and simplification, to find contours among these points. Extensive experimental results on synthetic and natural images show that our method can effectively extract contours from both clean and noisy images. Experiments on the Berkeley Segmentation Dataset also show that our proposed computational-geometry method can be linked with any state-of-the-art pixel-based contour extraction algorithm to remove noise and close gaps without severely dropping the contour accuracy. Moreover, contours extracted by our method have a much more compact representation than contours obtained by traditional pixel-based methods. Such a compact representation allows more efficient extraction of shape features in subsequent computer vision and pattern recognition tasks.

Keywords: contour extraction, image processing, computational geometry, point linking.

1 Introduction

Image contours refer to lines and boundaries of interesting regions within digital images, including object boundaries and boundaries of regions defined by sudden changes of brightness, color or texture. Image contour extraction is crucial for analyzing the contents of an image and is therefore paramount in many computer vision and pattern recognition applications [18], including image segmentation [1,35], object detection [3], and object recognition [3,50]. Arbeláez et al. [1] proposed an image segmentation method for transforming the output of any contour detector into a hierarchical region tree, thus reducing the problem of image segmentation to that of finding contours. Shotton et al. [50] proposed a method for recognizing objects from several contour fragments. Bai et al. [3] proposed to detect and recognize contour parts using shape similarity. However, highly accurate image contour extraction algorithms are typically computationally intensive. Furthermore, extracting contours from images with

* Supported in part by NSF grant DBI-0743670 and an ADVANCE grant from Utah State University. A preliminary version of this paper appeared in the *Proceedings of the 21st Canadian Conference on Computational Geometry (CCCG'09)* [53].

M.L. Gavrilova and C.J.K. Tan (Eds.): Trans. on Comput. Sci. XIII, LNCS 6750, pp. 13–43, 2011.

complex shapes, with textures, or with substantial amount of noise is especially difficult due to discontinuities in lines and region boundaries. In this paper, we present a simple contour extraction method based on computational-geometry techniques.

Image contour extraction is an active research area, as suggested by recent papers [1,8,16,38,45,60]. In general, contour extraction techniques can be classified into two main categories: region-based, and line-based. Region-based techniques use similar brightness, color or texture properties to segment the image into regions and extract contours directly from the segmented region boundaries. Line-based techniques use high contrast of luminance, color, or texture to find lines or boundaries. Region-based techniques include three kinds of segmentation techniques [23] (e.g., region-growing-based, region-splitting-based, and region-merging-based segmentation techniques), and techniques based on graph theory and cuts [15,57]. Line-based techniques include active contour techniques [29], edge detection-based techniques [6,38,49], and edge grouping-based techniques. Active contour techniques adapt an initial estimate of a closed contour until the sum of some external and internal energy functions are minimized. Edge grouping-based methods use line approximation algorithms to connect noise-removed edge pixels into segments and further group these segments to obtain object boundaries [12,51]. Interested readers may refer to the survey by Papari and Petkov [42] for more details on line-based contour extraction methods.

Our work is closely related to the edge grouping-based contour extraction techniques [12,13,22,47,48,51,52,56,59]. These edge grouping-based contour extraction techniques can be further classified into three categories: local, regional, and global-based [20, pp. 725–738]. Local methods (e.g., gradient-based methods [26,40] and statistical-based methods [36]) analyze a small neighborhood around each pixel and link adjacent pixels which satisfy some criteria. Regional methods (e.g., edge linking-based algorithms [41,46]) connect pixels that are previously known to be part of the same region or contour. They further apply polygonal fitting to find contour approximations. Global methods (e.g., Hough transform [9]) find sets of pixels lying on curves of specific shape without using any prior knowledge. However, all these three categories of methods have their own shortcomings.

Specifically, local methods ignore valuable global information about the geometric proximity of pixels and tend to find locally optimal solutions; regional methods require prior knowledge about the regional membership of each pixel to find closed boundaries; and global methods work well to find certain types of shapes but fail to generalize to any arbitrary shape. In addition, most edge grouping-based methods find either closed or open boundaries but not both. Furthermore, most edge grouping-based methods are sensitive to noise resulting from the edge detectors since they mainly consider boundary information such as proximity, closure, and continuity and disregard region information such as homogeneity and appearance of structures.

To address the noise issues, two kinds of solutions have been proposed. The first kind [22,25,47,48,58,59] is to apply measures such as length, gaps, smoothness, and closure to assign a saliency value to each edge pixel, and apply a thresholding approach to filter out noisy pixels and keep contour pixels. The second kind [51] is to combine both boundary and region information to remove noisy pixels and find closed boundaries.

Recently, researchers have begun to use computational-geometry techniques to solve problems of *digital-geometry*[1] [2,30]. It is well known that computational-geometry techniques work in a continuous domain and digital-geometry techniques work with discrete points or pixels on the integer coordinates. As a result, computational-geometry techniques are more general than digital-geometry techniques and they are sometimes more efficient than methods that check all the pixels in the region. For example, finding all the edge pixels in a region could be done more efficiently with computational-geometry range searching than by checking all the pixels in it. Furthermore, even though image pixels are on a grid, natural image features have a much larger scale; thus, considering the position and orientation of edges as continuous values is a better model than using discrete values. In view of the advantages of computational-geometry, we present a simple method based on computational-geometry for extracting contours from digital images with complex shapes and with substantial amount of noise.

Our computational-geometry method is simpler than digital-geometry methods, because it is based on simple general rules instead of many special cases considered by digital-geometry methods. In addition, our proposed technique is unique and different from traditional image processing techniques from the following perspectives: (i) It works with a set of oriented feature points instead of edge pixels from the input images. (ii) It works with geometric primitives (points and line segments) at each step instead of obtaining a geometric representation of the contours as a post-processing step. (iii) It finds contours in the presence of substantial amount of noise and gets rid of all the points that are not part of any contour. Our major contributions are as follows:

1. Proposing a simple computational-geometry method for contour extraction, which requires no prior knowledge about the regional membership of pixels and is not restricted to any particular shapes.
2. Developing a suite of simple and general geometric algorithms that can work on any set of oriented points obtained from digital images and other sources, where each geometric algorithm can be independently applied to achieve desired goals in other tasks.
3. Predetermining default parameter values that work well for extracting contours from digital images based on the valid assumption that the closest pair of edge pixels is likely separated by a small distance (one or two pixels).

To evaluate the performance of our proposed method, we compared it with some methods evaluated by Williams and Thornber [59] and also performed various tests on a large set of synthetic and natural images, and measured the accuracy, connectivity, and level of compression of the extracted contours. Experiments show that our method can effectively extract contours from noisy images and it can achieve a high level of compression for the extracted contours.

The remainder of the paper is organized as follows: Section 2 presents the proposed computational-geometry method for contour extraction. Section 3 explains our choice of default parameter values and the performance measures used to evaluate our results.

[1] "Digital geometry is the study of geometric or topologic properties of sets of pixels of voxels" [30]. It is therefore closely related to the fields of digital image processing and computer graphics.

Section 4 quantitatively evaluates the performance of the proposed method with extensive experiments that show its effectiveness. Section 5 draws the conclusion and presents the direction for future work.

2 Computational Geometric Algorithms for Contour Extraction

In this section, we explain the steps of our proposed method. It consists of two stages, as shown in Fig. 1(a): input conversion and geometric algorithms. In the first stage, we extract a set of oriented points from the input image, and in the second stage, we find contours among these points using geometric algorithms. The second stage is the most important and has three steps, as shown in Fig. 1(b): for point clustering, we filter points using a clustering technique; for point linking, we link points, based on proximity and orientation, into paths representing the contours; and for path simplification, we simplify paths by reducing their number of points. In the following subsections, we explain each stage in detail.

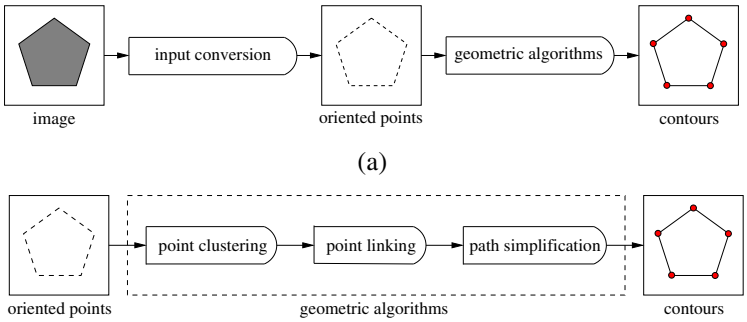


Fig. 1. Block diagram of our method. (a) Two stages. (b) Geometric algorithms for the second stage.

2.1 Input Conversion

In this paper, we focus on the geometric algorithms of the second stage, which are independent of the method used by the input conversion stage. We use the Sobel edge detector [20] in the input conversion stage, to extract a set of edge pixels, due to its simplicity. Other edge detectors (e.g., the Canny edge detector [6]) or methods to detect edges based on brightness and texture [38] could also be used, as long as they can extract a set of edge pixels along with their orientations.

The input conversion stage transforms the problem of finding contours into a geometric problem, and it is very simple. First, we use the Sobel edge detector [49] to find a set of *edge pixels*, which correspond to possible contour pixels. Here, each edge pixel has a *magnitude* indicating the edge strength at the pixel location, and a *direction* indicating the edge angle relative to the horizontal line. Then, we transform each edge pixel into an *oriented* point p_i located at the center (x_i, y_i) of the edge pixel, whose orientation α_i is given by the edge direction, and whose weight w_i is initialized with the

edge magnitude. Finally, we normalize both spatial coordinates and orientation angles to $[0, 1]$, invert the y-axis to have the origin at the lower left corner, and center the area of the original image inside the unit square. Specifically, we convert any angle α greater than π to $\alpha \bmod \pi$ and map the angle $\alpha = \pi$ to 1. To ease discussion, we use angles in radians or degrees when describing the geometric algorithms later on.

2.2 Geometric Algorithms: Point Clustering

Clustering techniques have been widely used in image processing, where they are among the most powerful approaches to image segmentation [54], and they are also used as preprocessing steps for pattern recognition tasks. We use a clustering-based algorithm to reduce the number of points obtained from the input conversion stage. This algorithm keeps the distribution and orientations of the reduced point set close to those of the original set, and it is used to achieve the following goals: (1) reduce the processing time of the following steps; (2) improve the results of the next step (point linking), which can find cleaner contours if the number of very close points is reduced.

Algorithm. We reduce the number of oriented points using a simple iterative greedy algorithm. This greedy algorithm repeatedly merges the closest pair of points into a new point until the distance between the closest pair reaches a threshold $d_{c\text{-max}}$. Fig. 2 illustrates a typical arrangement of points before clustering and the result after clustering, where the distance between the closest pair of points is at least $d_{c\text{-max}}$.

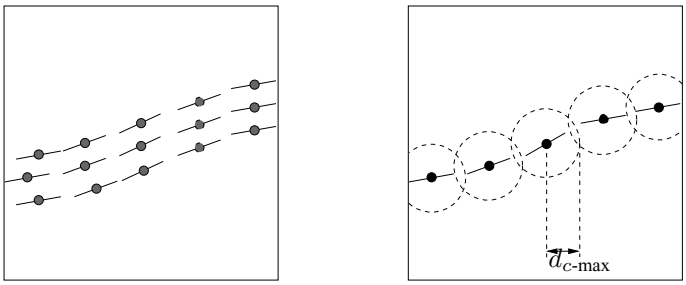


Fig. 2. (a) Typical arrangement of points before clustering. (b) Points after clustering.

When a pair of points is merged into one, the values of the new point are weighted averages of the values of the original points. Specifically, merging points p_i and p_j into a new point p_k is done as follows:

$$x_k = \frac{x_i w_i + x_j w_j}{w_i + w_j}, \quad y_k = \frac{y_i w_i + y_j w_j}{w_i + w_j}, \quad \alpha_k = \frac{\alpha'_i w_i + \alpha'_j w_j}{w_i + w_j}, \quad w_k = w_i + w_j; \tag{1}$$

where

$$\alpha'_i = \begin{cases} \alpha_i + \pi \text{ if } |\alpha_i - \alpha_j| > \frac{\pi}{2}, \alpha_i < \alpha_j \\ \alpha_i \quad \text{otherwise,} \end{cases} \tag{2}$$

$$\alpha'_j = \begin{cases} \alpha_j + \pi \text{ if } |\alpha_i - \alpha_j| > \frac{\pi}{2}, \alpha_j < \alpha_i \\ \alpha_j \quad \text{otherwise,} \end{cases} \tag{3}$$

are chosen so that the orientation of p_k is close to the orientations of both p_i and p_j. Note that we merge points based only on their distance, but the orientation of the new point is the weighted average of the orientations of the merged points. We did experiments with other functions that used both the spatial distance and the orientations of the points and got similar results. Therefore, we decided to use only the distance to be able to find points that could be merged faster, for example, by finding the closest pair or by using range searching.

Implementation. We use a simple array-based implementation of the clustering algorithm to show that the proposed geometric techniques work well to achieve the aforementioned goals. However, we also describe an alternative faster implementation. We base our analysis on the following observation:

Observation 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 a point p in S, if $d_{max} = c \cdot d_{min}$ for a small constant c, the number of points k in S within distance d_{max} to p is bounded by a constant.*

Since d_{min} is the closest pair distance, the circles of radius $d_{\text{min}}/2$ centered at every point are non-overlapping. Therefore, the number of points k within distance d_{max} of a point p is limited by the maximum number of circles of radius $d_{\text{min}}/2$ that can fit inside a circle of radius $d_{\text{max}} + d_{\text{min}}/2$. Based on the areas of these two circles, we find that $k \leq (2c+1)^2$.

Array-based implementation. Our array-based implementation works as follows. First, find all m candidate pairs of points separated by a distance of at most $d_{c\text{-max}}$ and put them in a pairs array. This is done by checking all pair of points in $O(n^2)$ time. Next, merge pairs of points from the pairs array in $O(n+m)$ time by the following steps: traverse the pairs array to find the closest pair in $O(m)$ time; merge the pair in constant time; remove from the pairs array all pairs containing any of the two merged points in $O(m)$ time; update the points array in constant time by removing the two merged points and adding the newly created point; and add all pairs that contain the newly created point and are separated by a distance of at most $d_{c\text{-max}}$ to the pairs array in $O(n)$ time by traversing the updated points array.

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

Proposition 1. *Given a set of n points with closest pair distance $d_{c\text{-min}}$, the array-based implementation of the clustering algorithm runs in $O(n^3)$ time. If $d_{c\text{-max}} = c \cdot d_{c\text{-min}}$ for a small constant c, it runs in $O(n^2)$ time.*

Improved running time. The array-based implementation must traverse the pairs array to find the closest pair, and traverse the 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 which consider only nearby points. The algorithm can be implemented to run in $O(n^2 \log n)$ time by finding the closest pair in $O(n \log n)$ time [4] at every merging step, or in $O(n \log n)$ time by using a data structure that maintains the closest pair in $O(\log n)$ time per insertion and deletion [5].

Proposition 2. *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.*

2.3 Geometric Algorithms: Point Linking

Two categories of algorithms have been widely used to find contours from a set of edge or region boundary pixels. The first category uses *contour tracing* algorithms [39,44] to trace the contours by starting from a known contour pixel and repeatedly moving to adjacent contour pixels until a stopping condition is met (usually returning to the starting pixel). Contours extracted by these algorithms are stored as a sequence of pixels encoded as a chain-code [17]. However, algorithms under this category work relatively well only on clean boundaries without gaps and the encoding can take a lot of space. The second category uses *edge linking* algorithms [41,46] which link edge pixels if they are within a small neighborhood and have a similar magnitude or direction. Contours extracted by these algorithms are sometimes stored using a geometric representation, and an additional step is needed to obtain this representation. However, algorithms under this category have to examine all the edge pixels within a chosen neighborhood window to find the best pixels to link, which is not efficient if the window is large. Alternatively, they can use a small window, but only if they use an additional step to fill edge gaps before linking.

To address the shortcomings of the above two categories of algorithms, we propose a simple point linking algorithm to find contours as a set of paths represented as polylines, i.e., sequences of line segments between the given points. Similar to edge linking algorithms, our point linking algorithm links points based on proximity and orientation. However, it is based on simple rules, it does not require the boundaries to be clean or connected, and it yields equivalent results of multiple steps required by other pixel-based linking algorithms, e.g., filling gaps between edges, linking pixels, and approximating the contours with line segments.

Algorithm. Our algorithm is in spirit similar to Prim's algorithm for minimum spanning tree (MST) [19, pp. 366–369] in the sense that it extends a path (grows a cluster) until it cannot be extended anymore. However, our algorithm does not find a MST. Instead, it finds multiple paths based on simple rules for choosing appropriate segments. A path is obtained by starting from a single segment and then greedily extending it in both directions.

The initial segment of a new path is a between a pair of isolated points (p_i, p_j), whose distance is within the threshold distance $d_{\ell\text{-max}}$ and whose weight $w(p_i, p_j)$ is the

maximum among the weights of all the candidate pairs. In our algorithm, the weight of a pair of points (p_i, p_j) measures how good is the line segment between p_i and p_j for representing contours. That is, a higher weight indicates a better choice of segment to represent contours. Specifically, the weight of a pair of points (p_i, p_j) depends on the following values: the distance between the points p_i and p_j (i.e., $|p_i p_j|$), the difference between the orientation of point p_i and the orientation of segment $p_i p_j$ (i.e., a_{ij}^i), the difference between the orientation of point p_j and the orientation of segment $p_i p_j$ (i.e., a_{ij}^j), the threshold distance $d_{\ell\text{-max}}$, and the threshold orientation difference $\alpha_{\ell\text{-max}}$. It is computed as follows:

$$w(p_i, p_j) = \min\{w_d(p_i, p_j), w_{\alpha_i}(p_i, p_j), w_{\alpha_j}(p_i, p_j)\}, \tag{4}$$

where

$$w_d(p_i, p_j) = \begin{cases} 1 - \frac{|p_i p_j|}{d_{\ell\text{-max}}} & \text{if } |p_i p_j| < d_{\ell\text{-max}} \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

$$w_{\alpha_x}(p_i, p_j) = \begin{cases} 1 - \frac{\alpha_{ij}^x}{\alpha_{\ell\text{-max}}} & \text{if } \alpha_{ij}^x < \alpha_{\ell\text{-max}} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

It is clear that the weight of the distance $w_d(p_i, p_j)$ and the weight of the two orientation differences, i.e., $w_{\alpha_i}(p_i, p_j)$ and $w_{\alpha_j}(p_i, p_j)$, are in the range $[0, 1]$, and their values are determined by the linear and monotonically decreasing functions shown in Fig. 3. Note that this function returns the value of the worst part of the segment between two points, which can be the distance or the difference between the orientation of the segment and the orientation of one of the points. Therefore, if the points are too far away, they will not be linked even if the orientations are very similar; and if the orientations are very different, they will not be linked even if they are very close.

After the initial segment is chosen, the path is extended at both ends by repeatedly adding the most appropriate point. For example, 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 that satisfies the following conditions: (1) the distance from p_i to p_x is at most $d_{\ell\text{-max}}$;

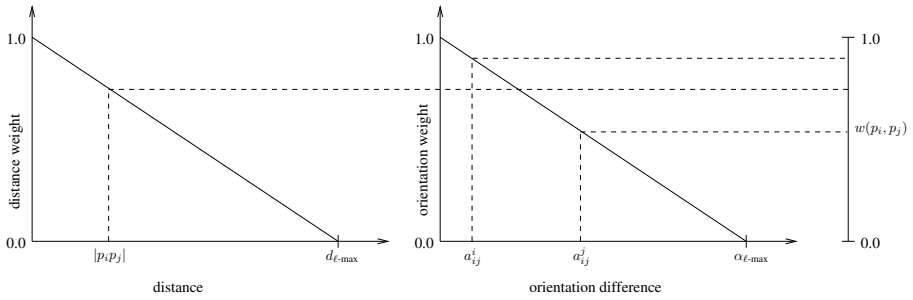


Fig. 3. Link weight function for a pair of points (p_i, p_j). A value is assigned to the distance $|p_i p_j|$, and each of the orientation differences a_{ij}^i and a_{ij}^j, respectively. The weight $w(p_i, p_j)$ of the pair is the minimum of those three values.

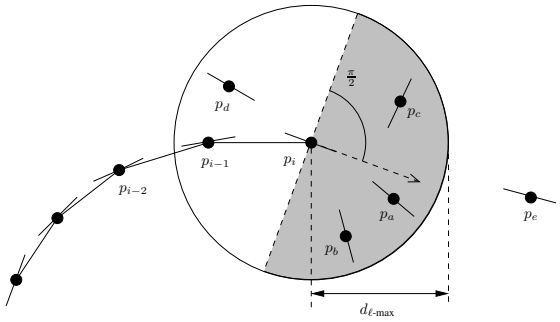


Fig. 4. Extending the path $P = (\ldots, p_{i-2}, p_{i-1}, p_i)$ at the end point p_i. Only the three candidate points in the gray area can be linked when the maximum allowable linking distance is set to be $d_{\ell\text{-max}}$.

(2) the turn angle from p_i to p_x, relative to the orientation of p_i, is at most $\pi/2$; and (3) the weight of the pair (p_i, p_x) is the maximum among all pairs satisfying the first two conditions. Fig. 4 shows an example where the point with the best weight would be selected from among p_a, p_b, and p_c, since p_d and p_e do not satisfy the first two conditions. The algorithm stops extending a path when there are no more candidate points or the added point is already part of an existing path (i.e., the path intersects another path).

Implementation. We use a simple array-based implementation of the linking algorithm to show that the proposed geometric techniques work well. However, we also describe an alternative faster implementation.

Array-based implementation. Our array-based implementation works as follows. First, find all m possible pairs that can be linked in $O(n^2)$ time, put them in a pairs array, and sort them by weight in $O(m \log m)$ time. Then, find the paths by repeating the following steps: First, find the initial segment for the next path by traversing the pairs array until a pair of isolated point is found. To determine if points are part of a path, we store the degree of each point (i.e., the number of linked segments connected to a point) in a separate array. Then, extend the path in one direction until it cannot be extended any more, and repeat the same process of extending the path in the other direction. When extending a path, we find the next point in $O(m)$ time by checking all pairs in the pairs array, and adding as the new end point, the other point of a pair that includes the current end point and satisfies the three linking conditions.

Since at most two segments are added for each point, the total number of segments for all paths output by the point linking algorithm is $O(n)$. Therefore, the total running time is $O(nm)$, which can be $O(n^3)$ if m is $O(n^2)$. However, Observation 1 implies that if $d_{\ell\text{-max}}$ is small, m is $O(n)$, and the algorithm runs in $O(n^2)$ time.

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

Improved running time. The running time of the algorithm can be improved by using a range searching technique, such as bucketing or range trees. We choose $d_{\ell\text{-max}} = c \cdot d_{\ell\text{-min}}$ for a small constant c, where $d_{\ell\text{-min}}$ is the closest pair distance. As a result, the number of points within distance $d_{\ell\text{-max}}$ 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 of these pairs can be found in linear time and sorted in $O(n \log n)$ time. Similarly, by using range searching, it takes constant time to find the initial pair for a path and the best point to extend it. Therefore, the algorithm can be implemented to run in $O(n \log n)$ time.

Proposition 4. *Given a set of n points with closest pair distance $d_{\ell\text{-min}}$, and $d_{\ell\text{-max}} = c \cdot d_{\ell\text{-min}}$ for a small constant c, the linking algorithm can be implemented to run in $O(n \log n)$ time by using a range searching technique.*

2.4 Geometric Algorithms: Path Simplification

The paths obtained by the linking step are often more complex than necessary and thus it is desirable to simplify them by reducing the number of points they have without introducing any perceivable visual distortion. This reduction is reasonable and feasible based on the observation that many consecutive points of the same contour are either collinear or close to the same straight line. For example, 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 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. By reducing the number of points, the simplification algorithm achieves the following goals: (1) reduce the space required to store contours, (2) reduce small inconsistencies resulting from noise, and (3) improve the efficiency of any post-processing techniques on the contours.

Since paths are represented by polylines, the problem of simplifying paths is the same as the geometric problem of *polygonal chain approximation* or *simplification*, defined as follows [18]: 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) $m \leq n$ (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 $\varepsilon(r, s)$

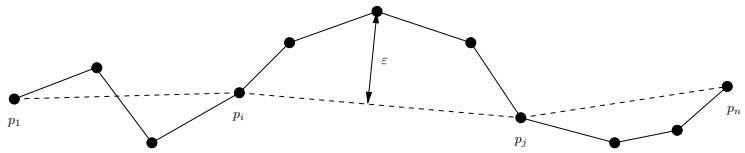


Fig. 5. A polygonal chain with eleven vertices (solid line) and an approximation with four vertices (dashed line). The sub-chain $p_i \ldots p_j$ is approximated by the segment $p_i p_j$. The error of segment $p_i p_j$, under the segment criterion, is ε.

between q_jq_{j+1} and each p_k, $r \leq k \leq s$, is smaller than some predetermined error tolerance ε. Several error criteria have been used for the approximation problem, each with its own algorithmic issues. We use the *segment* criterion, which defines the error $\varepsilon(r, s)$ of a segment as the maximum of the distances from the vertices p_k to the segment q_jq_{j+1}, and consider the error of chain Q as the maximum of the errors of each of its segments: $\varepsilon(P, Q) = \max_{1\leq j<m}\{\varepsilon(q_j, q_{j+1})\}$. Fig. 5 shows a polygonal chain and an approximation.

Algorithm. Several algorithms [7,14,24] have been proposed for approximating polygonal chains, with most optimal algorithms taking $\Omega(n^2)$ time to find approximations in $\mathbb{R}^2$. We propose a dynamic programming algorithm to simplify polygonal chains based on the segment criterion. To simplify contours, we run this algorithm on every path and then remove *isolated points* (i.e., the points do not belong to any path) and paths under a certain length that might exist due to noise in the original image.

The *detour* [10] of a chain P on the pair of points (p_i, p_j) is defined as the total length $|p_i \ldots p_j|$ of the sub-chain $p_i \ldots p_j$ divided by the length of the segment p_ip_j. That is,

$$d(i, j) = \frac{|p_i \ldots p_j|}{|p_ip_j|}. \tag{7}$$

Our algorithm uses the following property of the detour to determine whether a segment of the approximation has an error within the desired tolerance: Given a segment p_ip_j and an error tolerance ε, there is a bound

$$d_T(i, j) = \frac{\sqrt{4\varepsilon^2 + |p_ip_j|^2}}{|p_ip_j|} \tag{8}$$

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

The derivation of this property is illustrated in Fig. 6. The sub-chain $p_i \ldots p_k \ldots p_j$ (solid line) is approximated by the segment p_ip_j, with p_k being the farthest point of the sub-chain to the segment. For an error tolerance ε, 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 $|p_ip_j|$ and 2ε, and two semicircles of radii ε with centers at p_i and p_j. For 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 the error is within the tolerance

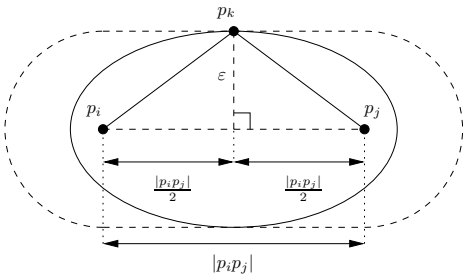


Fig. 6. Maximum detour for a specified error tolerance ε

ε is determined by the ellipse (solid line) with foci p_i and p_j that is tangent to the boundary of the tolerance region. Otherwise, if the detour is larger, some points of the sub-chain $p_i \dots p_j$ could be outside of the tolerance region. Specifically, this ellipse is tangent to the tolerance region when the sub-chains $p_i \dots p_k$ and $p_k \dots p_j$ are equal to the segments p_ip_k and p_kp_j, respectively, and $|p_ip_k| = |p_kp_j|$. Therefore, the maximum detour is

$$d_T(i,j) = \frac{2\sqrt{\varepsilon^2 + \left(\frac{|p_ip_j|}{2}\right)^2}}{|p_ip_j|} = \frac{\sqrt{4\varepsilon^2 + |p_ip_j|^2}}{|p_ip_j|}. \tag{9}$$

Dynamic programming. Let $K(i)$ denote the minimum number of points of the best known approximation of the sub-chain $p_1 \dots p_i$. Our proposed dynamic programming algorithm works as follows:

Base case: For all i,

$$K(i) = i.$$

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

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

The base case initializes $K(i)$ to use all points. That is, the best known approximation of each sub-chain $p_1 \dots p_i$ is initially the same chain with no points removed. Then, to determine $K(i)$, the recurrence step finds the minimum number of points required by any of the best known approximations having p_j as the next to last vertex, and takes the minimum of those plus one as the best approximation for the sub-chain $p_1 \dots p_i$. The condition $d(j,i) \le d_T(j,i)$ ensures that only valid segments p_jp_i, i.e., those with an error within the specified tolerance, are considered. However, the algorithm may not find the the minimum number of points possible for the specified tolerance because some segments with an error within the tolerance may be discarded by the detour test. Nevertheless, experiments show that it produces good approximations with considerable reduction in the number of points.

To recover the simplified chain, we use a table $P(i)$ to store the point choices for the best approximation. The choice table $P(i)$ stores the index of the next to last point p_j of the best approximation of the sub-chain $p_1, \dots, p_j, p_i$. We initialize $P(i)$ to $i-1$ and set $P(i)$ to j every time $K(i)$ is updated to $K(j)+1$ during the computation of $K(i)$. After the computation of $K(i)$ and $P(i)$, we recover the best approximation from $P(i)$. We start with the last point p_n and backtrack from $P(n)$ until the first vertex p_1 is reached. Every time $P(i)$ is visited, we add point $k = P(i)$ to the approximation chain and move to $P(k)$.

Implementation. The implementation of the dynamic programming algorithm is straightforward, and with a constant time to compute the detour, it clearly runs in $O(n^2)$ time. To compute the detour in constant time, we first compute the lengths $L(i)$ of all sub-chains $p_1 \dots p_i$ by $L(i-1) + |p_{i-1}p_i|$, as a preprocessing step. Then the detour for segment p_ip_j is simply

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

Since the backtracking procedure takes linear time to recover the best approximation, the overall running time of the algorithm is $O(n^2)$. After simplification, the filtering steps to remove isolated points and short paths take linear time on the total number of points.

Proposition 5. *Given a polygonal chain P with n vertices, the dynamic programming algorithm finds an approximation chain Q in $O(n^2)$ time, for the segment criterion.*

3 Default Parameters and Performance Measures

In this section, we explain our choices of default parameter values and the rational for these choices. We also explain the three performance measures, namely, accuracy, connectivity, and compression, which are used to quantitatively evaluate the performance of the proposed algorithm.

3.1 Default Parameter Values

Since every step of the process requires some parameters, it is useful to have good default parameter values that can achieve decent contour extraction results for a majority of images. We use the closest pair distance d_{min} and a range parameter r to automatically select appropriate values for some of the parameters. Below, we describe the choice of each parameter in detail.

Input Conversion (magnitude threshold). We use $\mu + \frac{1-\mu}{3}$ as the magnitude threshold for the Sobel edge detector, where μ is the normalized average magnitude. This choice ensures that a relatively large number of strong edge pixels is extracted.

Point Clustering (maximum distance to merge points). We use $d_{c\text{-max}} = r_c \cdot d_{c\text{-min}}$ as the maximum distance to merge points, where the clustering range r_c is set to be 2. Since the closest pair distance is most likely the distance between two points corresponding to adjacent edge pixels, this choice ensures that any two points corresponding to adjacent pixels in the eight neighborhood window centered at one of the pixels can be merged. We do not set $d_{c\text{-max}}$ to be large in order to avoid losing small details.

Point Linking (maximum distance and orientation difference to link points). We use $d_{\ell\text{-max}} = r_\ell \cdot d_{\ell\text{-min}}$ as the maximum distance to link points, where the linking range r_ℓ is set to be 3. The rational for this choice is that small values of $d_{\ell\text{-max}}$ can cause breaks in the contours because points are not linked, and large values of $d_{\ell\text{-max}}$ can cause false contours to be detected. Our choice is a compromise between these two extremes and allows the linking algorithm to find contours when there are small discontinuities or some intermediate points have inconsistent orientations resulting from noise. We use $\alpha_{\ell\text{-max}} = 35^\circ$ as the maximum orientation difference to link points because it is important to use a value that is not too small, to link points of curved contours, yet not too large, to prevent linking nearby points of different contours.

Path Simplification (error tolerance and filtering thresholds). We use $\varepsilon = \frac{1}{2} d_{\min}$ as the error tolerance. This value ensures that the visual difference between the original and the simplified contours is hardly perceivable for most images, because the distance between oriented points is small before simplification. After simplification, the filtering steps can remove isolated points and paths under a certain length that might exist due to noise in the original image. By default, we remove any isolated points and keep all paths.

3.2 Performance Measures

Evaluating accuracy. Most methods for evaluating contour extraction algorithms use ground truths [21,37,55], which correspond to the real contours, and count the number of *true positives* (TP) as the contour pixels correctly detected, *false negatives* (FN) as the contour pixels missed, and *false positives* (FP) as the contour pixels detected incorrectly. Then they use traditional statistical measures such as precision and recall to evaluate the results.[2] However, these measures work with discrete binary data and cannot be applied directly to evaluate the performance of our method, which works with continuous lines. For this reason, we use the Hausdorff distance [27,28] as the performance measure to evaluate the accuracy (i.e., quality) of our results. Given two sets of points X and Y, the Hausdorff distance is defined as follows:

$$H(X,Y) = \max\{h(X,Y), h(Y,X)\}, \tag{10}$$

where $h(X,Y)$ is the *directed* Hausdorff distance between the two sets of points X and Y, and is defined as

$$h(X,Y) = \sup_{x \epsilon X} \inf_{y \epsilon Y} |xy|, \tag{11}$$

where $|xy|$ is the distance between x and y. The directed Hausdorff distance is the maximum distance from a point in X to the nearest point in Y. A small directed Hausdorff distance indicates that all points in X are close to some point in Y. The Hausdorff distance is the maximum distance from any point in one of the sets, X or Y, to a point in the other set. A small Hausdorff distance indicates a high degree of similarity between two sets of points.

In the context of evaluating the quality of extracted contours, we let G be the ground truth of an image and C be a set of extracted contours of the same image. Then $h(C,G)$, $h(G,C)$, and $H(C,G)$, can be used as alternative measures to the selectivity, sensitivity, and accuracy, as used in other methods, respectively. If $h(C,G)$ is small, all parts of 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, the extracted contours are both real and complete.

Evaluating connectivity. Most segmentation and contour extraction algorithms produce pixel-based output. Therefore, evaluation methods are also pixel-based [21,37] and disregard information related to pixel connectivity. However, we believe that connectivity of extracted contours is important since contours are not useful if they are accurate but

[2] Precision $= \mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$, Recall $= \mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$.

not well connected. In other words, it is difficult to obtain shape information if the extracted contours are broken into many small parts (i.e., they have low connectivity). As a result, we evaluate the connectivity of the extracted contours by counting the number of paths. In general, the number of paths used to represent a shape can vary and the connectivity required is application dependent. For example, a rectangle could be represented by a single path covering the whole perimeter, or by four paths corresponding to each side. However, a small number of paths likely indicates a better connectivity.

Evaluating compression. Most methods for evaluating contour extraction do not evaluate the amount of compression. We believe that a compact contour representation not only saves storage space but also facilitates computer vision and pattern recognition tasks. As a result, it is important to evaluate the compactness of the extracted contours. We determine the amount of compression as the ratio of the number of edge pixels extracted by the edge detector to the number of points after the clustering and simplification steps. This ratio gives a measure of the level of compression at each step. A larger ratio means a better compression and a more compact contour representation.

4 Experimental Results

To evaluate the performance of our method, we performed tests with a variety of synthetic and natural images. First, we compared our system with other peer systems under the same settings. Second, we conducted extensive experiments on synthetic images, including the binary images from the MPEG7 CE Shape–1 Part B database, whose contours are known as ground truth, to measure accuracy and connectivity of the extracted contours. Third, we conducted experiments on natural images to measure the compression level attained by our method. Fourth, we evaluated our method using different parameter values, to show the effectiveness of our default parameters values. Finally, we conducted experiments on the Berkeley Segmentation Dataset [37] using two of the top pixel-based contour extraction algorithms as the input conversion stage of our proposed computational-geometry method. These experiments were used to evaluate the accuracy of the contours after using our proposed algorithms to postprocess the contours from any pixel-based contour extraction algorithm.

Most experiments were performed on two Dell OptiPlex GX620 computers with 2.8–3.0 GHz Pentium D Processors, 2 GB RAM, running Windows XP Professional Service Pack 3 and Cygwin 1.5.25(0.156/4/2). The experiments on the Berkeley Segmentation Dataset where performed on a Dell OptiPlex GX620 computer with a 2.8 GHz Pentium D Processor, 1 GB RAM, running Linux Ubuntu release 10.10 and Matlab R2010(b).

4.1 Comparison with Peer Systems

Williams and Thornber [59] evaluated several measures of perceptual saliency [22,25,47,48,58,59] and compared their ability to segment contour pixels from a set of oriented edge pixels with noise. These methods determine the saliency of an edge based on properties of the contours going through it, such as length, gaps, smoothness, or closure, and then segment the edges using a threshold for the saliency value. Unlike these methods, our method does not assign a saliency value to each edge. Instead, it

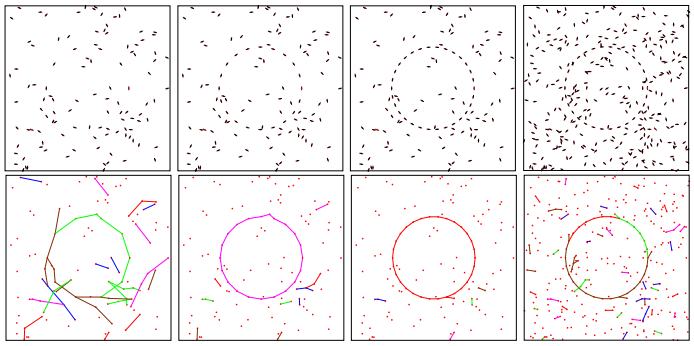


Fig. 7. Circle patterns with additional random noise. The top row shows the oriented points and the bottom row shows our linking results. (a) Ten-point circle with 100 noise points. (b) Twenty-point circle with 100 noise points. (c) Thirty-point circle with 100 noise points. (d) Thirty-point circle with 300 noise points.

finds contours with the best segments between points and gets rid of all the points that are not part of any contour.

We compare the results of our algorithms with some of the results published by Williams and Thornber. To ensure fair comparison, we generated point sets with the same specifications used in their experiments and used them as input for our point linking algorithm. That is, we generated three circles with evenly spaced sample points (10, 20, and 30 points), diameter equal to half of the size of the image, and 100 random noise points. We did not use the point clustering algorithm in this experiment because of the low sampling rate. Also, because the input is not an image, we could not use our assumptions to determine default parameter values. In this case, we used empirically determined values for the maximum link distance.

The results show that our method can obtain good contour extraction results. Fig. 7 shows the results of our algorithm for all three circles. The thirty-point circle (Fig. 7(c)) is detected correctly, and the twenty-point circle (Fig. 7(b)) is detected with small inaccuracies. This is better than the results obtained by several of the methods evaluated by Williams and Thornber. On the other hand, our result for the ten-point circle (Fig. 7(a)) is not very good, while the best method they evaluated can segment it. However, it is hard for a human subject to see the ten-point circle due to the substantial amount of noise and the low sampling rate. Furthermore, our method can successfully detect contours for the thirty-point circle (Fig. 7(d)) with additional 300 random noise edge points. In this regard, we believe that our algorithm achieves decent contour extraction results, since we can detect the contours in all but the hardest case, but we can detect contours with significantly higher amount of noise in other cases. Our results are consistent with human observations after removing small detected paths.

4.2 Experimental Results on Images with Ground Truth

We used two data sets, whose expected results were known, to evaluate the accuracy and connectivity of the contours extracted by our method. The first data set is a set of

Table 1. Contour extraction results (avg. $\pm$ stdv.) for random shapes. (i) Without removing short segments (top). (ii) After removing segments shorter than 0.05 (bottom).

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 ± 19.12
10000	0.4942 ± 0.2211	0.0020 ± 0.0027	0.4942 ± 0.2211	629.60 ± 11.62
0	0.0012 ± 0.0002	0.0148 ± 0.0323	0.0148 ± 0.0322	12.01 ± 16.20
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

300 pictures, evenly divided into 15 groups of 20 pictures, with the pictures of each group containing a different combination of randomly generated shapes (line segments and ellipses). These pictures are formed by point sets and are not pixel images. However, we consider them to be 512×512 and choose parameter values accordingly. The second data set includes the 1400 binary images from the MPEG7 CE Shape–1 Part B database[3], which is a collection of images of simple to moderately complex shapes commonly used for the evaluation of shape descriptors and object recognition and classification algorithms [33]. These binary images have a size ranging from about 50×50 to 1100×500. We added different amounts of noise to the images in each data set and tested our method on these noisy images.

Random shapes. For the random shapes data set, we added 5000 and 10000 noise points to the original clean pictures, which are approximately 2% and 4% of the number of pixels in a 512×512 image. Because the points are not equally spaced in a grid, we set the maximum distance to merge points (e.g. $d_{c\text{-max}}$) to be 0.002 (approximately one pixel away for a 512×512 image). The top part of Table 1 shows the results for the 300 pictures with random shapes, without removing short segments: the average and standard deviation of the directed Hausdorff distance from the extracted contours to the expected results $h(C,G)$, the directed Hausdorff distance from the expected results to the extracted contours $h(G,C)$, the Hausdorff distance $H(C,G)$, and the number of paths. It clearly indicates the effects of noise on the output. Due to the Hausdorff distance's sensitivity, there is a significant increase in the Hausdorff distance even for the least amount of noise. That is because, if a single segment far away from the ground truth is detected, the Hausdorff distance becomes large. However, we observe that $h(C,G)$ is high and $H(G,C)$ is low. This indicates that we detect correct contours without missing any parts, but also detect other false contours (corresponding to short segments) resulting from noise. Consequently, the number of paths gradually increases as the amount of noise increases.

To remove false contours, we removed segments with length less than 0.05. The final accuracy and connectivity results are summarized in the bottom part of Table 1. It clearly shows that noise has little effects on the output after removing short segments. That is, both the Hausdorff distance and the number of paths increase very little when the amount of noise increases. Furthermore, the small Hausdorff distance values

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

Table 2. Contour extraction results (avg. $\pm$ stdv.) for binary images. (i) Without removing short segments (top). (ii) After removing segments shorter than 0.05 (bottom).

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
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

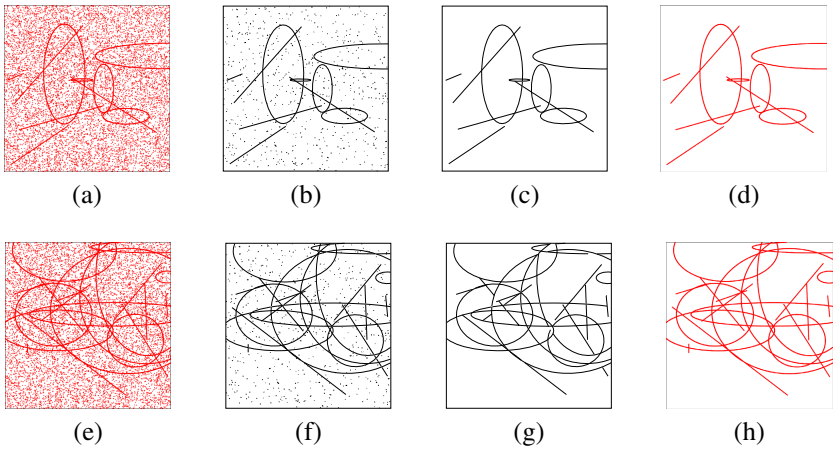


Fig. 8. Contour extraction results for pictures with random shapes and 10000 noise points. It is clear that results shown in (c) and (g) are similar to expected results shown in (d) and (h). (a) Sample picture `SET14_7` with 5 lines and 5 ellipses. (b) Extracted contours of `SET14_7` without removing short segments. (c) Extracted contours of `SET14_7` after removing segments shorter than 0.05 (178 points, 17 paths, $H(C,G) = 0.0180$). (d) Ground truth for `SET14_7`. (e) Sample picture `SET15_7` with 10 lines and 10 ellipses. (f) Extracted contours of `SET15_7` without removing short segments. (g) Extracted contours of `SET15_7` after removing segments shorter than 0.05 (435 points, 56 paths, $H(C,G) = 0.0306$). (h) Ground truth for `SET15_7`.

indicate that most contours are detected correctly and false contours are no longer present. The small number of paths is an indication of good connectivity. Specifically, we have an average of 5.6 shapes per picture for all test cases, which leads to about 2 paths per extracted contour. Based on all these observations, we can safely conclude that our method is resilient to noise.

Fig. 8 demonstrates contour extraction results for two sample pictures with 10000 noise points. To show the noise effects, we respectively display contour extraction results without removing segments and with segments shorter than 0.05 removed. The cleaned contour extraction results confirm that most contours are extracted correctly. However, the connectivity is affected at intersections since lines are sometimes broken at intersection points.

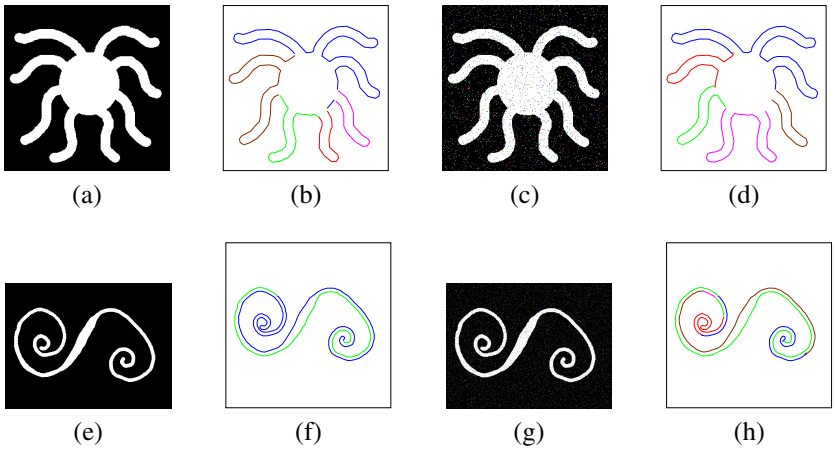


Fig. 9. Contour extraction results for binary images. (a) Sample binary image `octopus-16` (256×256). (b) Extracted contours of `octopus-16` without removing short segments (172 points, 6 paths, $H(C, G) = 0.0155$). (c) `octopus-16` image with noise (5%). (d) Extracted contours of noisy `octopus-16` after removing segments shorter than 0.05 (176 points, 5 paths, $H(C, G) = 0.0396$). (e) Sample binary image `spring-6` (640×480). (f) Extracted contours of `spring-6` without removing short segments (322 points, 2 paths, $H(C, G) = 0.0217$). (g) `spring-6` image with noise (5%). (h) Extracted contours of noisy `spring-6` after removing segments shorter than 0.05 (356 points, 7 paths, $H(C, G) = 0.0209$).

Binary images. For the binary images data set, the amount of noise was determined by a noise density: the number of noise pixels is equal to the noise density multiplied by the total number of pixels in an image. In this case, the contour extraction results are similar to those obtained with the first data set. Table 2 summarizes the results for the 1400 binary images having different amounts of noise without removing short segments (top), and with segments shorter than 0.05 removed (bottom). The small Hausdorff distance values obtained after removing short segments indicate that most contours are detected correctly even with a high amount of noise. The Hausdorff distance values are larger than the ones obtained for the first data set mainly due to the removal of small segments or isolated points that are sometimes part of the ground truth of the images. Specifically, at the noise density of 5%, approximately 1000 images have $H(C, G) \leq 0.05$, 1200 images have $H(C, G) \leq 0.10$, and 1300 images have $H(C, G) \leq 0.15$. For many of the images with $H(C, G) > 0.15$, we detect most contours but miss small segments or points in the ground truth. Again, the small number of paths shown in the bottom part of Table 2 is an indication of good connectivity.

Fig. 9 demonstrates contour extraction results for two sample binary images with 0% and 5% noise density. For clean binary images, the figure displays contour extraction results without removing short segments. For noisy binary images, the figure displays contour extraction results after removing segments shorter than 0.05. To facilitate viewing different paths, we also display paths in different colors for both results. This figure clearly confirms that most contours are correctly extracted. However, it can be seen that some sharp corners are not detected and contours are broken at such corners. This is

Table 3. Compression results

Images	Points (edge detector)	Points (after simplification)	Paths
High textured	21127.80 $\pm$ 12070.91	3679.90 $\pm$ 2273.74 (5.95 $\pm$ 0.69)	1412.70 $\pm$ 915.64
Low/mid textured	11770.40 $\pm$ 6877.48	1758.70 $\pm$ 1221.60 (7.60 $\pm$ 2.10)	611.50 $\pm$ 506.56

mainly because the orientation change is large at such corners and our point linking algorithm does not connect points with very dissimilar orientations. These broken corners also make the number of paths higher.

4.3 Experimental Results on Natural Images without Ground Truth

We used natural images from three sources to evaluate the performance of our method in terms of compression. These three sources include the USC-SIPI Image Database[4], the ImageProcessingPlace.com[5], and our own image database. The data sets from the USC-SIPI Image Database and the ImageProcessingPlace.com have 44 and 17 images, respectively. Some of these images have several versions, both in color and gray scale, or in different resolutions. We chose 18 images of size 512×512 and two images of our own for our experiments. We further classified these 20 images into two groups: 10 highly textured images and 10 low to medium textured images. For each image, we determined the number of points after the clustering and simplification steps.

Table 3 statistically summarizes the compression results for the chosen 20 images. Specifically, it summarizes the average number of points adding and subtracting its standard deviation for 10 highly textured images and 10 low or medium textured images when using a Sobel threshold of 0.2 with non-maxima suppression to thin the edges and after applying our three computational-geometry steps. It also summarizes the point reduction ratios (compression ratios) in parenthesis after the three computational geometrical steps. The average of the number of final paths adding and subtracting its standard deviation for 10 highly textured images and 10 low or medium textured images is listed in the last column of the table. We can clearly see that the amount of compression is considerable. The number of points is always reduced at least 5 times, for both groups of textured images.

Fig. 10 illustrates some example results.

4.4 Experimental Results Using Different Parameters Values

We used default parameter values for most experiments. However, a range of values could also give good and comparable results on average, but no values will give much better results. We illustrate this by evaluating the changes of the contour extraction results using different parameter values for the clustering range r_c, the linking range r_ℓ, and the linking angle (i.e., the maximum orientation difference to link oriented points)

[4] Miscellaneous volume, downloaded on September 1, 2009 from `http://sipi.usc.edu/database/index.html`

[5] "Standard" test images set, downloaded on September 1, 2009 from `http://www.imageprocessingplace.com/root_files_V3/image_databases.htm`

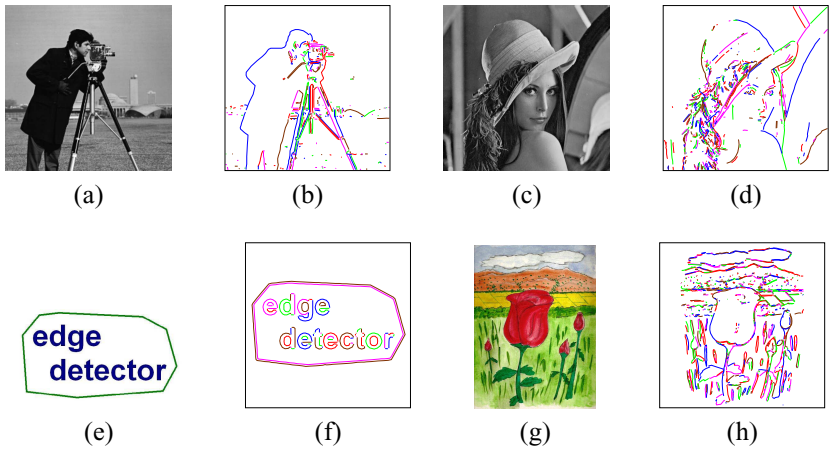


Fig. 10. Contour extraction results for sample natural images using default values for all parameters. Paths are shown in different colors. (a) Sample image `cameraman.jpg`. (b) Extracted contours of `cameraman.jpg`: 980 points and 292 paths obtained from 7361 original points (7.51 reduction ratio). (c) Sample image `lena.jpg`. (d) Extracted contours of `lena.jpg`: 1509 points and 488 paths obtained from 9977 original points (6.61 reduction ratio). (e) Sample image `text.jpg`. (f) Extracted contours of `text.jpg`: 548 points and 74 paths obtained from 5555 original points (10.14 reduction ratio). (g) Sample image `rose.jpg`. (h) Extracted contours of `rose.jpg`: 2724 points and 756 paths obtained from 19436 original points (7.14 reduction ratio).

$\alpha_{\ell\text{-max}}$. We evaluated the results of our method when each parameter value is varied and the other parameters are set to default values. For each test, we used the 1400 clean images from the binary image data set to compute the Hausdorff distance and the number of paths. The average values are plotted in Fig. 11 and the results are discussed below.

Clustering Range. The Hausdorff distance increases when the clustering range increases, because a larger clustering range makes the point clustering algorithm merge more points. Consequently, the original point set is approximated by a smaller point set and the extracted paths are less accurate. On the other hand, the number of paths decreases from a very large value to a relatively small value once the clustering range is slightly greater than 1, since no points are merged if the range is less than 1. As a result, when there is no clustering, many points are close together and multiple lines are detected for a single contour. Fig. 11(a) and (b) clearly demonstrates that our default value of 2, slightly after the big reduction in the number of paths, is a good choice. Larger values reduce the accuracy without improving the connectivity.

Linking Range. When the linking range is less than or equal to 1, no paths are extracted. When the linking range is slightly larger than 1, a few paths are extracted. Therefore, contours are not complete and the Hausdorff distance is large. When the linking range is slightly increased, more paths are detected and the Hausdorff distance is reduced.

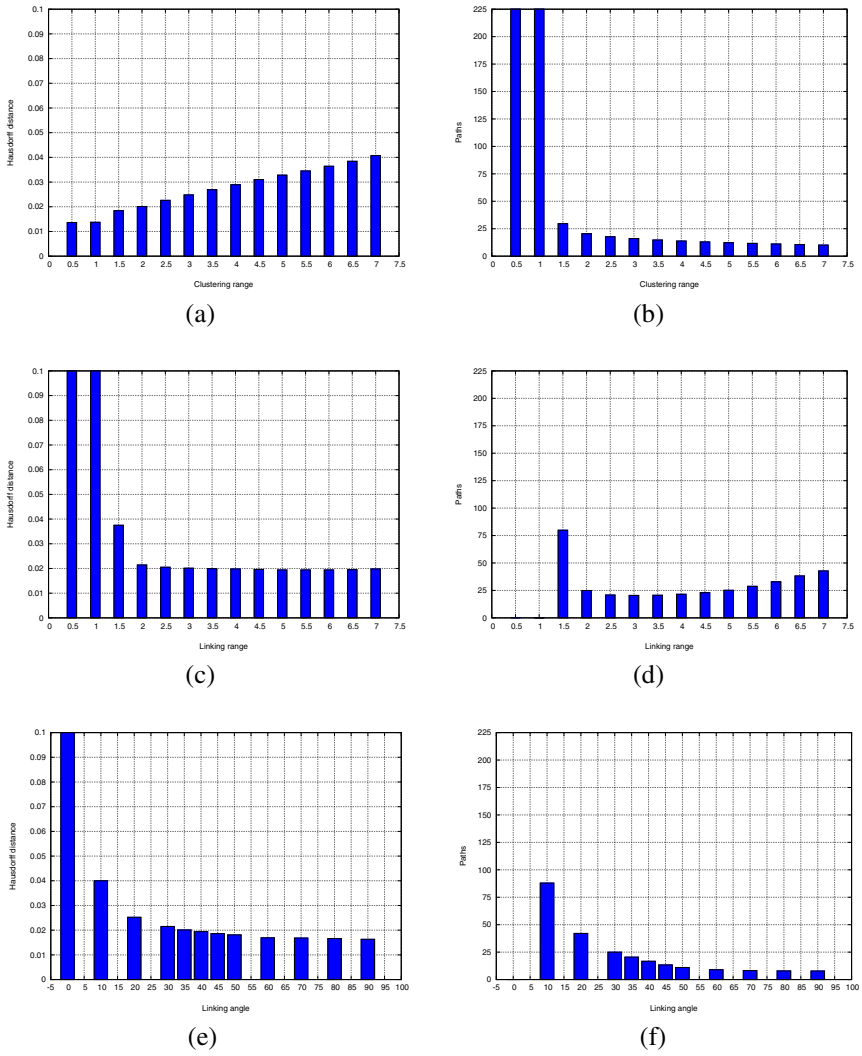


Fig. 11. Sensitivity of the Hausdorff distance and the number of paths on different parameters. (a) The Hausdorff distance for the clustering range. (b) The number of paths for the clustering range. (c) The Hausdorff distance for the linking range. (d) The number of paths for the linking range. (e) The Hausdorff distance for the linking angle. (f) The number of paths for the linking angle.

However, the number of paths remains high because many points cannot be linked. That is, only multiple short paths are detected (broken paths). When the linking range is further increased, more points are linked and both the Hausdorff distance and the number of paths decrease considerably. This trend continues until the linking range is too large and the number of paths starts to increase due to detected false contours.

Fig. 11(c) and (d) clearly demonstrate that our default value of 3, slightly after the big reduction in the Hausdorff distance and the number of paths, is a good choice. A range of values gives similar results, but no significant improvement is obtained with other values.

Linking Angle. When the linking angle is too small (e.g. close to zero), no paths are extracted. When the linking angle is slightly increased, a few points are linked and many other points remain isolated. That is, only very straight parts of contours are detected. As a result, the Hausdorff distance and the number of paths remain relatively large. However, when the linking angle is further increased, more points are linked and both the Hausdorff distance and the number of paths decrease considerably. Fig. 11(e) and (f) clearly demonstrates that our default value of 35°, slightly after the big reduction in the Hausdorff distance and the number of paths, is a good choice. Both performance measures continue to decrease after 35°. This is expected because more paths are linked. However, we do not use a larger angle value in order to avoid linking close paths that are not part of the same contour, such as close parallel lines. In these cases, the Hausdorff distance would not increase considerably since the paths are close to each other.

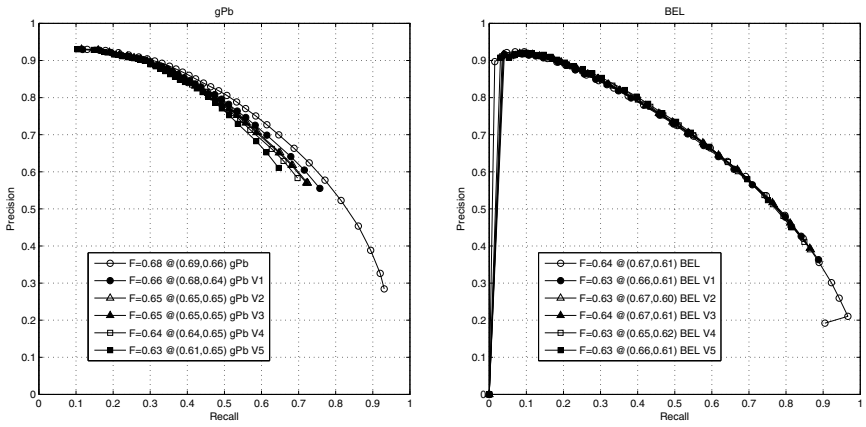
In summary, the point clustering algorithm is necessary to ensure that a reasonable number of paths (not too many) are extracted. The point linking algorithm needs to use a sufficiently long distance and a sufficiently large angle to be able to link most points. However, these two values cannot be too large, which may lead to more linked points and more false contours. Nevertheless, there is a wide range of values for the parameters that seem to give good results after their corresponding minimum values, and our default values are in that range.

4.5 Experimental Results on Berkeley Segmentation Dataset

We performed tests on the Berkeley Segmentation Dataset and Benchmark [37], which is widely used to evaluate the accuracy of contour extraction algorithms (we used the 100 test images in the gray scale dataset). Other datasets include the South Florida and the Sowerby datasets [32]. We used the same F-measure determined by the Berkeley benchmark to evaluate the accuracy of the contours. This F-measure is computed by $\frac{2PR}{P+R}$ where P means precision and R means recall.

For each image in the dataset, the benchmark requires a gray scale image where the intensity of each pixel represents the likelihood (or probability) that the pixel is a contour pixel in the dataset image. This image is thresholded to obtain a binary image that is compared with the ground truth, and precision and recall are computed for several threshold values to obtain a precision-recall curve. In our case, to generate this image, one possibility is to find contours for different threshold values of the Sobel edge detector, discretize the contours into pixels, and assign to each pixel the maximum threshold value for which it is output by the algorithms. However, the results for this benchmark will depend on the method used for preprocessing, and the Sobel edge detector is not the best method.

Suppose that we get some input points, using a Sobel edge detector or some other preprocessing method. Then we can only improve the results by (i) filling gaps and (ii) filtering noise. Our algorithms can fill small gaps and they can get rid of some noise, but



(a) (b)

Fig. 12. Precision-recall curves. (a) Five variations of our proposed computational-geometry algorithm and the `gPb` algorithm on the Berkeley gray scale dataset. (b) Five variations of our proposed computational-geometry algorithm and the `BEL` algorithm on the Berkeley gray scale dataset.

we cannot expect a considerable improvement with this, especially if the preprocessing is already quite good. Furthermore, other properties of our algorithms could affect the result in a negative way. For example, simplifying the paths introduces localization errors since simplified contours are close to the extracted contours, but not exactly in the same place.

Nevertheless, we show that if we use different pixel-based algorithms as preprocessing to obtain a set of input points for our geometric algorithms, we obtain results similar to the output of the pixel-based algorithms without postprocessing. This means that our algorithms are able to link most contours and our geometric algorithms do not affect the accuracy of the detected contours very much.

In our experiments, we used two of the top pixel-based contour extraction algorithms, namely, Global Probability of Boundary (`gPb`) [38] and Boosted Edge Learning (`BEL`) [8], as our input conversion step to produce the input for our proposed computational-geometry algorithms. Since the output of `gPb` or `BEL` is a gray scale image, we applied thirty evenly spaced threshold values on the output image to obtain thirty thresholded binary images containing edge pixels at different strength. The orientations of the edge pixels in each thresholded binary image were obtained by applying the Sobel mask on the corresponding pixels of the original image. For each of the two top pixel-based contour algorithms, we performed the following variations of our proposed computational-geometry algorithms on each of the thirty resultant thresholded binary images:

- Variation 1 (`V1`): clustering, and linking; without removing short segments or isolated points.
- Variation 2 (`V2`): clustering, and linking; without removing short segments but removing isolated points.

- Variation 3 (V3): clustering, and linking and simplification; without removing short segments but removing isolated points.
- Variation 4 (V4): clustering, and linking and simplification; removing short segments (length less than 1% of the image size) and isolated points.
- Variation 5 (V5): clustering, and linking and simplification; removing short segments (length less than 2% of the image size) and isolated points.

Fig. 12(a) and Fig. 12(b) show the precision-recall curves of these five variations of our proposed computational-geometry algorithms applied on the output from gPb and BEL, respectively. To facilitate comparison, we also plot the precision-recall curve of gPb and BEL in Fig. 12(a) and Fig. 12(b), respectively. Fig. 12(a) shows the contour accuracy of the five variations of our computational-geometry algorithms is slightly lower than the accuracy of the gPb algorithm. This is expected since our algorithms can get rid of some pixels that cannot be linked. The contour accuracy of the five variations also decreases a bit since more filtering is applied on the subsequent variations. Fig. 12(b) shows the contour accuracy of the five variations of our computational-geometry algorithms is comparable with the accuracy of the BEL algorithm. This is mainly due to the thick edges in the output of the BEL algorithm, which makes it harder to miss contours.

Fig. 13 shows four sets of sample results on three Berkeley benchmark images, together with their ground truths. We observe that the output after applying our proposed computational-geometry algorithms (V3) on the gPb output has less noise than the output directly from gPb. For the left and the center images, the thresholded outputs after applying our proposed computational-geometry algorithms (V3) on the gPb output are very similar to the thresholded output directly from gPb. However, the thresholded output after applying our proposed computational-geometry algorithms (V3) on the gPb output of the right image has broken contours (e.g. around the tree). This is caused by our linking algorithm, which cannot link points that do not have similar orientations, as is the case around some textured regions.

Fig. 14 shows another two sets of sample results on three Berkeley benchmark images. Fig. 14(b) illustrates the output from our proposed computational-geometry algorithms (V3) on the gPb output obtained with the best global threshold determined by the benchmark. Fig. 14(c) shows the geometric contours obtained by connecting all paths. Here, all the final contour points are marked in red color and all the paths are connected by blue lines. The pixel output in Fig. 14(b) is obtained by converting these paths to pixels, but it is clear that the connected paths are simpler.

Our experiments demonstrate that our proposed computational-geometry algorithms can be linked with any state-of-the-art pixel-based contour extraction algorithm (e.g., gPb and BEL) to remove noise and close gaps without severely dropping the contour accuracy. That is, our algorithm maintains a comparable contour accuracy as the linked pixel-based contour extraction algorithm, but using a more natural and compact representation. Since contours do not have to be complete for successful object recognition, as supported by the findings of Shotton et al. [50] and Bai et al. [3], the effectiveness of our contour representation should be further validated with experiments on real applications.

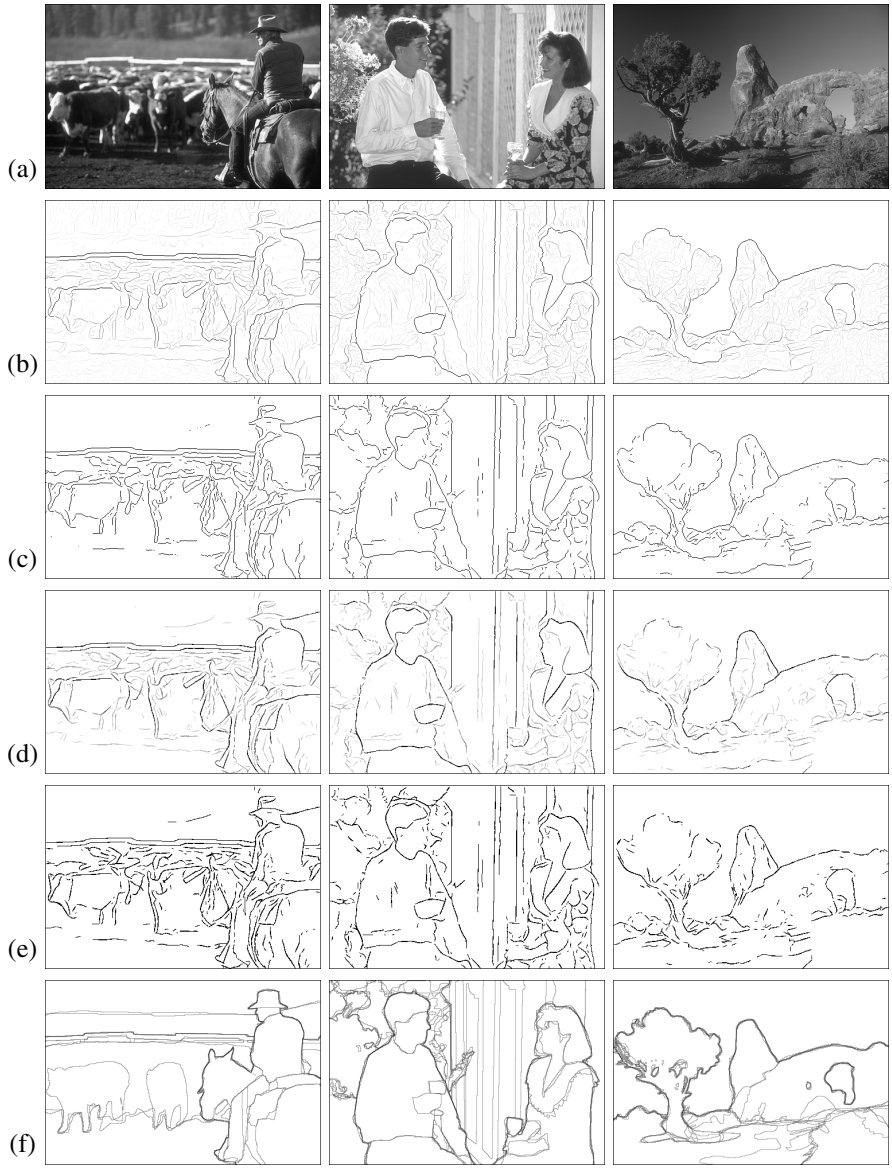


Fig. 13. Sample contours for three Berkeley images. (a) Original image. (b) `gPb` output. (c) Thresholded `gPb` output using the best global threshold determined by the benchmark. (d) `V3` output with `gPb` preprocessing. (e) Thresholded `V3` output with `gPb` preprocessing using the best global threshold determined by the benchmark. (f) Ground truth.

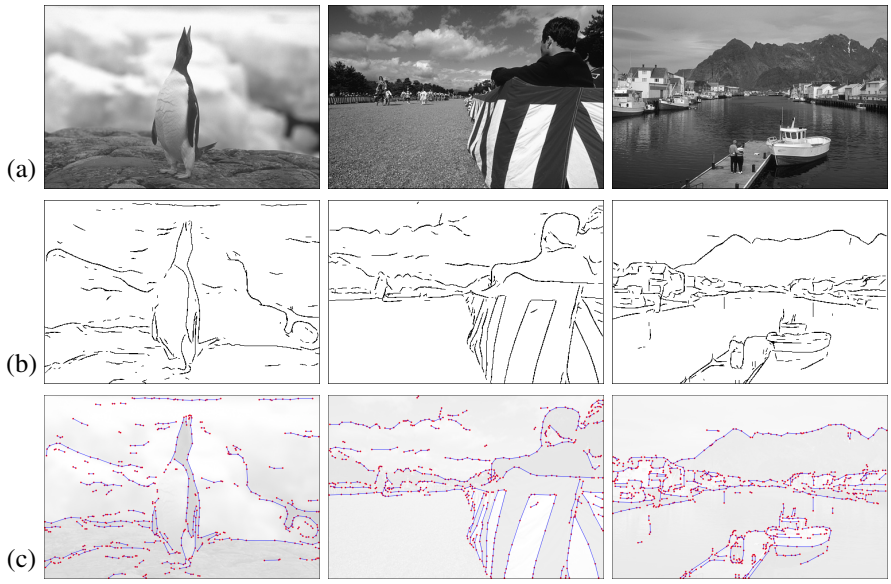


Fig. 14. Sample contours for another three Berkeley images. (a) Original image. (b) Thresholded `V3` output with `gPb` preprocessing using the best global threshold determined by the benchmark. (c) Geometric contours for the `V3` output in (b).

5 Conclusion and Future Work

The extensive experimental results show that our proposed method can effectively extract contours even from images with considerable noise. The contours extracted by our method are more compact than the ones obtained by a pixel-based representation due to our usage of a geometric representation. Our method is also more general than traditional pixel-based image processing methods. As a result, it can be used with any set of points, not necessarily with integer coordinates. It requires no prior knowledge about the regional membership of pixels, and it is not restricted to any particular type of shapes. However, our method has some disadvantages. First, it cannot detect small details because the spacing between points is increased after the point clustering step. Second, it does not detect sharp corners because the point linking algorithm connects points with similar orientations. Consequently, the connectivity is slightly affected.

More sophisticated methods can produce better results for specific applications. However, we show how simple geometric algorithms can be used to extract contours from digital images. Our results are promising and we expect that extensions to these simple algorithms can make them more effective and useful for specific applications such as detecting closed contours or junctions. Our future work includes evaluating the efficiency of the fastest implementations of the algorithms, detecting sharp corners, using arcs, circles, or splines to represent contours, and reducing the sensitivity to noise by exploiting better weight functions.

6 Availability and Supplemental Material

Our programs have been tested on three major platforms: Microsoft Windows (Cygwin), Linux, and Mac OS X. The complete source code, documentation, data set, and experimental results can be downloaded from the companion website of this paper at `http://www.cs.usu.edu/~mjiang/contour/`.

Acknowledgment: We thank Dr. Nicholas Flann for valuable comments.

References

1. Arbeláez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 33, 898–916 (2011)
2. Asano, T., Brimkov, V.E., Barneva, R.P.: Some theoretical challenges in digital geometry: A perspective. Discrete Applied Mathematics 157, 3362–3371 (2009)
3. Bai, X., Yang, X., Latecki, L.J.: Detection and recognition of contour parts based on shape similarity. Pattern Recognition 41, 2189–2199 (2008)
4. Bentley, J.L., Shamos, M.I.: Divide-and-conquer in multidimensional space. In: Proceedings of the 8th Annual ACM Symposium on Theory of Computing, pp. 220–230 (1976)
5. Bespamyatnikh, S.N.: An optimal algorithm for closest pair maintenance. In: Proceedings of the 11th Annual Symposium on Computational Geometry, pp. 152–161 (1995)
6. Canny, J.: A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 8, 679–698 (1986)
7. Chan, W.S., Chin, F.: Approximation of polygonal curves with minimum numbers of line segments or minimum error. International Journal of Computational Geometry and Applications 6, 59–77 (1996)
8. Dollár, P., Tu, Z., Belongie, S.: Supervised learning of edges and object boundaries. In: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), pp. 1964–1971 (2006)
9. Duda, R.O., Hart, P.E.: Use of the Hough transformation to detect lines and curves in pictures. Communications of the ACM 15, 11–15 (1972)
10. Ebbers-Baumann, A., Klein, R., Langetepe, E., Lingas, A.: A fast algorithm for approximating the detour of a polygonal chain. Computational Geometry: Theory and Applications 27, 123–134 (2004)
11. Elder, J.H., Goldberg, R.M.: Ecological statistics of Gestalt laws for the perceptual organization of contours. Journal of Vision 2, 324–353 (2002)
12. Elder, J.H., Krupnik, A., Johnston, L.A.: Contour grouping with prior models. IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 661–674 (2003)
13. Elder, J.H., Zucker, S.W.: Computing contour closure. In: Buxton, B.F., Cipolla, R. (eds.) ECCV 1996. LNCS, vol. 1064, pp. 399–412. Springer, Heidelberg (1996)
14. Eu, D., Toussaint, G.T.: On approximating polygonal curves in two and three dimensions. CVGIP: Graphical Models and Image Processing 56, 231–246 (1994)
15. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. International Journal of Computer Vision 59, 167–181 (2004)
16. Felzenszwalb, P., McAllester, D.: A min-cover approach for finding salient curves. In: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW 2006), pp. 185–185 (2006)

17. Freeman, H.: Computer processing of line drawings. ACM Computing Surveys 6, 57–97 (1974)
18. Goodman, J.E., O'Rourke, J.: Handbook of Discrete and Computational Geometry, 2 ed. CRC Press, Boca Raton (2004)
19. Goodrich, M.T., Tamassia, R.: Algorithm Design: Foundations, Analysis, and Internet Examples. John Wiley & Sons, Chichester (2002)
20. Gonzalez, R.C., Woods, R.E.: Digital Image Processing, 3rd edn. Pearson Prentice Hall (2008)
21. Grigorescu, C., Petkov, N., Westenberg, M.A.: Contour detection based on nonclassical receptive field inhibition. IEEE Transactions on Image Processing 12, 231–236 (2003)
22. Guy, G., Medioni, G.: Inferring global perceptual contours from local features. International Journal of Computer Vision 20, 113–133 (1996)
23. Haris, K., Efstratiadis, S.N., Maglaveras, N., Katsaggelos, A.K.: Hybrid image segmentation using watersheds and fast region merging. IEEE Transactions on Image Processing 7, 1684–1699 (1998)
24. Heckbert, P.S., Garland, M.: Survey of polygonal surface simplification algorithms, Technical Report, Carnegie Mellon University, School of Computer Science (1997)
25. Hérault, L., Horaud, R.: Figure-ground discrimination: a combinatorial optimization approach. IEEE Transactions on Pattern Analysis and Machine Intelligence 15, 899–914 (1993)
26. Hunt, G.C., Nelson, R.C.: Lineal feature extraction by parallel stick growing. In: Proceedings of the Third International Workshop on Parallel Algorithms for Irregularly Structured Problems, pp. 171–182 (1996)
27. Huttenlocher, D., Klanderman, G., Rucklidge, W.: Comparing images using the hausdorff distance. IEEE Transactions on Pattern Analysis and Machine Intelligence 15, 850–863 (1993)
28. Huttenlocher, D., Olson, C.: Automatic target recognition by matching oriented edge pixels. IEEE Transactions on Image Processing 6, 103–113 (1997)
29. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: Active contour models. International Journal of Computer Vision 1, 321–331 (1988)
30. Klette, R., Rosenfeld, A.: Digital Geometry: Geometric Methods for Digital Picture Analysis. Morgan Kaufmann, San Francisco (2004)
31. Koffka, K.: Principles of Gestalt Psychology. Harcourt, Brace & Company, New York (1935)
32. Konishi, S., Yuille, A.L., Coughlan, J.M., Zhu, S.C.: Statistical edge detection: learning and evaluating edge cues. IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 57–74 (2003)
33. Latecki, L.J., Lakämper, R., Eckhardt, U.: Shape descriptors for non-rigid shapes with a single closed contour. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 424–429 (2000)
34. Mahamud, S., Williams, L.R., Thornber, K.K., Xu, K.: Segmentation of multiple salient closed contours from real images. IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 433–444 (2003)
35. Malik, J., Belongie, S., Leung, T., Shi, J.: Contour and texture analysis for image segmentation. International Journal of Computer Vision 43, 7–27 (2001)
36. Mansouri, A., Malowany, A.S., Levine, M.D.: Line detection in digital pictures: A hypothesis prediction verification paradigm. Computer Vision, Graphics, and Image Processing 40, 95–114 (1987)
37. Martin, D., Fowlkes, C., Tal, D., Malik, J.: 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 2001), pp. 416–423 (2001)

38. Martin, D.R., Fowlkes, C., Malik, J.: Learning to detect natural image boundaries using local brightness, color, and texture cues. IEEE Transactions on Pattern Analysis and Machine Intelligence 26, 530–549 (2004)
39. Moore, G.A.: Automatic scanning and computer processes for the quantitative analysis of micrographs and equivalent subjects. Pattern Recognition: Pictorial Pattern Recognition 1, 275–326 (1969)
40. Nelson, R.C.: Finding line segments by stick growing. IEEE Transactions on Pattern Analysis and Machine Intelligence 16, 519–523 (1994)
41. Nevatia, R., Babu, K.R.: Linear feature extraction and description. Computer Graphics and Image Processing 3, 257–269 (1980)
42. Paparilow, G., Petkov, N.: Edge and line oriented contour detection: state of the art. Image and Vision Computing 29, 79–103 (2011)
43. Pelli, D.G., Majaj, N.J., Raizman, N., Christian, C.J., Kim, E., Palomares, M.C.: Grouping in object recognition: The role of a Gestalt law in letter identification. Cognitive Neuropsychology 26, 36–49 (2009)
44. Ren, M., Yang, J., Sun, H.: Tracing boundary contours in a binary image. Image and Vision Computing 20, 125–131 (2002)
45. Ren, X.: Multi-scale improves boundary detection in natural images. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part III. LNCS, vol. 5304, pp. 533–545. Springer, Heidelberg (2008)
46. Robinson, G.S.: Detection and coding of edges using directional masks. In: Proceedings SPIE Conference on Advances in Image Transmission Techniques, pp. 117–125 (1976)
47. Sarkar, S., Boyer, K.L.: Quantitative measures of change based on feature organization: eigenvalues and eigenvectors. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 1996), pp. 478–483 (1996)
48. Sha'ashua, A., Ullman, S.: Structural saliency: the detection of globally salient structures using a locally connected network. In: Second International Conference on Computer Vision (ICCV 1988), pp. 321–327 (1988)
49. Sobel, I.E.: Camera models and machine perception, Ph.D. dissertation, Stanford University, CA, USA (1970)
50. Shotton, J., Blake, A., Cipolla, R.: Multiscale categorical object recognition using contour fragments. IEEE Transactions on Pattern Analysis and Machine Intelligence 30, 1270–1281 (2008)
51. Stahl, J.S., Oliver, K., Wang, S.: Open boundary capable edge grouping with feature maps. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2008), pp. 1–8 (2008)
52. Stahl, J.S., Wang, S.: Convex grouping combining boundary and region information. In: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV 2005), pp. 946–953 (2005)
53. Tejada, P.J., Qi, X., Jiang, M.: Computational geometry of contour extraction. In: Proceedings of the 21st Canadian Conference on Computational Geometry (CCCG 2009), pp. 25–28 (2009)
54. Toussaint, G.T.: Computational geometry and computer vision. Contemporary Mathematics 119, 213–224 (1991)
55. Wang, S., Ge, F., Liu, T.: Evaluating edge detection through boundary detection. EURASIP Journal on Applied Signal Processing 2006, 1–15 (2006)
56. Wang, S., Kubota, T., Siskind, J.M., Wang, J.: Salient closed boundary extraction with ratio contour. IEEE Transactions on Pattern Analysis and Machine Intelligence 27, 546–561 (2005)
57. Wang, S., Siskind, J.M.: Image segmentation with ratio cut. IEEE Transactions on Pattern Analysis and Machine Intelligence 25, 675–690 (2003)

58. Williams, L.R., Jacobs, D.W.: Stochastic completion fields: a neural model of illusory contour shape and salience. In: Proceedings of the Fifth International Conference on Computer Vision (ICCV 1995), pp. 408–415 (1995)
59. Williams, L.R., Thornber, K.K.: A comparison of measures for detecting natural shapes in cluttered backgrounds. International Journal of Computer Vision 34, 81–96 (1999)
60. Zhu, Q., Song, G., Shi, J.: Untangling cycles for contour grouping. In: Proceedings of the 11th IEEE International Conference on Computer Vision (ICCV 2007), pp. 1–8 (2007)