Computational Geometry of Contour Extraction*

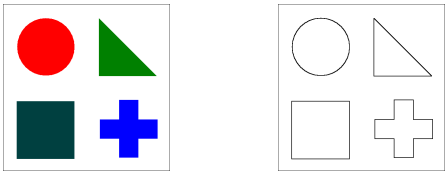
Pedro J. Tejada† Xiaojun Qi‡ Minghui Jiang§

Abstract

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.

1 Introduction

Contours are the boundary lines of geometric shapes within digital images; see Figure 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 [5, p. 1135]. This problem is especially difficult for images with complex shapes and with noise.



(a) (b)

Figure 1: (a) A digital image of shapes. (b) Contours.

Traditional methods for finding contours can be classified by scope depending on whether they do local, regional, or global processing [7, pp. 725–738]. Local methods analyze a small neighborhood about 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 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. We present a method for extracting contours from digital images using techniques from computational geometry. 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.

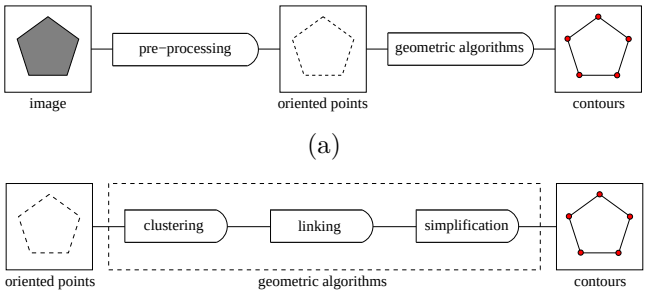


Figure 2: Overview of our method. (a) Two stages. (b) Geometric algorithms for the second stage.

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 Figure 2.

*Supported in part by NSF grant DBI-0743670 and an ADVANCE grant from Utah State University.

†Department of Computer Science, Utah State University, p.tejada@aggiemail.usu.edu. This work is part of Pedro J. Tejada's Master's thesis [20] supervised by Minghui Jiang.

‡Department of Computer Science, Utah State University, xiaojun.qi@usu.edu

§Department of Computer Science, Utah State University, mjiang@cc.usu.edu

2 Input Conversion

At the pre-processing stage, a Sobel edge detector [7] is used to determine possible contour pixels, which are then transformed into oriented points. The edge detector outputs a set of *edge pixels* where the intensity of the image changes abruptly. Each edge pixel has a *magnitude* indicating how good or strong is the edge at the pixel location, and a *direction* indicated by an angle. Then 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.

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 [22]. We use a clustering based algorithm to reduce the number of points in order to reduce the processing time of the following steps and improve the results of the linking step, which might find multiple lines where there should be a single contour, if points are close together.

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 $d_{\max}$, which is chosen to be a constant times the minimum distance $d_{\min}$ between two points in the input.

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. Merging points p_i and p_j into a new point p_k is done as follows:

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

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.

Implementation. 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 [1, 13] 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 [2].

4 Point Linking

Finding contours from image regions or edge pixels can be done by simple *contour tracing* algorithms such as Moore's algorithm [16], which traces the boundaries by starting from a known contour pixel and repeatedly moving to adjacent contour pixels until a stopping condition is met. Another possibility is to use edge linking algorithms, which link edge pixels if they are within a small neighborhood and have similar magnitude or direction [18, 19, 17, 23]. Similar to these algorithms, our algorithm also links points based on proximity and orientation.

Algorithm. Our algorithm is in spirit similar to Prim's algorithm for minimum spanning tree [6]. To generate a path, it starts from a single segment then greedily extends the path in both directions until maximal.

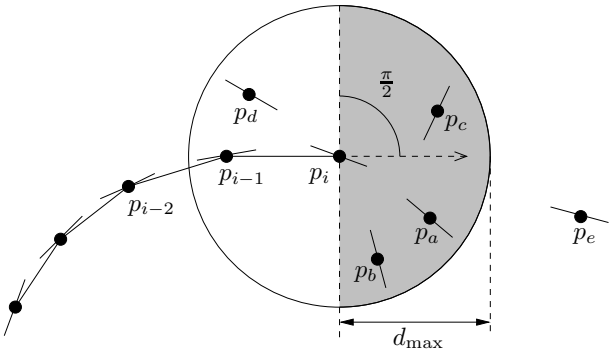


Figure 3: Extending the path $P = (\ldots, p_{i-2}, p_{i-1}, p_i)$ at the end point p_i. Only points in the gray area can be linked. The maximum distance allowed to link points is $d_{\max}$.

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 maximum. Then the path is extended at both ends by repeatedly adding points until there are no more candidates or the added point was already part of some path. 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 satisfying the following conditions: (i) the distance from p_x to p_i is at most $d_{\max}$, and (ii) the turn angle from $p_{i-1}p_i$ to p_ip_x is at most $\pi/2$. We refer to Figure 3 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 given conditions.

Weight function. 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, it 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 they are greater than or equal to $d_{\max}$ or $\alpha_{\max}$.

Implementation. 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 some range searching technique all these pairs can be found in linear time and sorted in $O(n \log n)$ time. Then, 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 $O(n \log n)$.

5 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. By doing that it is 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.

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 [5]: 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, $r \le k \le s$, is less than some predetermined error tolerance ε according to some error criterion.

Algorithm. Several algorithms have been proposed for approximating polygonal chains [21, 12, 15, 8, 3, 9], 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 [5]: for each p_k, $r \le k \le s$, the minimum distance from p_k to $q_j q_{j+1}$ is less than ε.

The *detour* [4] 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 $p_i p_j$:

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

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

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

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

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

Base case: For all i, $K(i) = i$.

Recurrence: For all i and j such that $1 \le j < i$ and $d(j, i) \le d_T(j, i)$, $K(i) = \min\{K(i), K(j) + 1\}$.

Implementation. The dynamic programming algorithm clearly runs in $O(n^2)$ time, where n is the number of vertices in the original chain, since the detour of any segment $p_i p_j$ can be computed in constant time after pre-computing the lengths of the sub-chains $p_1 \ldots p_i$ in linear time. Then recovering the best approximation is done in linear time using backtracking.

6 Experiments

To evaluate our method we have done tests with a variety of synthetic and natural images. See Figure 4 for two example results.

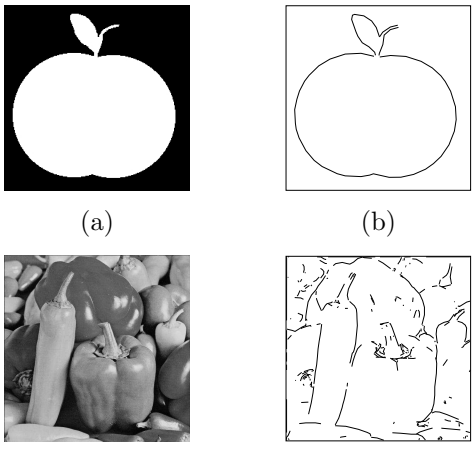


Figure 4: (a) Binary image `apple-11.jpg`. (b) Binary image contours: 61 points and 4 paths. (c) Natural image `peppers.jpg`. (d) Natural image contours: 2182 points and 214 paths.

Evaluating accuracy. To test the accuracy of our method, we use two data sets. The first data set is a set of random pictures, each containing a few randomly generated shapes (such as line segments and ellipses)

among 10,000 noise points. The second data set includes the binary images from the MPEG7 CE Shape–1 Part B database[1] [14] with noise injected: 5% of the pixels of each image (randomly selected with replacement) are set to random values.

We use the Hausdorff distance, with images rescaled to fit inside a unit square, as the performance measure to evaluate our results. It is widely used in pattern recognition [10, 11], but it is not commonly used to evaluate contour extraction performance because it is quite sensitive to noise. Nevertheless, it is good for our purposes since it can be applied to continuous lines instead of discrete pixels for which one can count pixels detected correctly (true positives) or incorrectly (false positives).

For the set of random shapes, the average Hausdorff distance is 0.0182 ± 0.0326, and the average number of paths is 12.68. For the set of binary images, the average Hausdorff distance is 0.0339 ± 0.0508, and the average number of paths is 9.57.

Evaluating compression. To evaluate the amount of compression obtained by our method we use some commonly used test images from the USC-SIPI Image Database[2] and some images created by ourselves. For each image we compare the number of points extracted by the edge detector with the number of points after the clustering and simplification steps. On average the number of points is reduced to 9.20 ± 2.76 percent.

7 Conclusion

Our experimental results show that our method can effectively extract contours from digital images with a moderate amount of noise. 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 that can be obtained by using a pixel-based representation.

References

[1] 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*, pages 220–230, 1976.

[2] S.N. Bespamyatnikh. An optimal algorithm for closest pair maintenance. In *Proceedings of the 11th Annual Symposium on Computational Geometry*, pages 152–161, 1995.

[3] 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*, 6:59–77, 1996.

[4] 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*, 27:123–134, 2004.

[5] J.E. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry*, 2nd edition. CRC Press, 2004.

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

[7] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*, 3rd edition. Pearson Prentice Hall, 2008.

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

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

[10] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, 1993.

[11] D. Huttenlocher and C. Olson. Automatic target recognition by matching oriented edge pixels. *IEEE Transactions on Image Processing* 6:103–113, 1997.

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

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

[14] 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*, pages 424–429, 2000.

[15] A. Melkman and J. O'Rourke. On polygonal chain approximation. In G.T. Toussaint, editor, *Computational Morphology*, pages 87–95, North-Holland, Amsterdam, 1988.

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

[17] R. Nevatia and K.R. Babu. Linear feature extraction and description. *Computer Graphics and Image Processing*, 3:257–269, 1980.

[18] L.G. Roberts. Machine perception of three dimensional solids. In J.T. Tippett et al., editors, *Optical and Electro-Optical Information Processing*, MIT Press, 1965.

[19] G.S. Robinson. Detection and coding of edges using directional masks. In *Proceedings SPIE Conference on Advances in Image Transmission Techniques*, pages 117–125, August 1976.

[20] P.J. Tejada. *A Computational Geometry Approach to Digital Image Contour Extraction*. Master's Thesis, Utah State University, 2009.

[21] 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.

[22] G.T. Toussaint. Computational geometry and computer vision. *Contemporary Mathematics*, 119:213–224, 1991.

[23] S.E. Umbaugh. *Computer Imaging: Digital Image Analysis and Processing*. CRC Press, 2005.

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

[2] `sipi.usc.edu/database/index.html`.