Proc. International Conference on Document Analysis and Recognition, Seattle, WA, September 2001

High-Quality Polygonal Contour Approximation Based on Relaxation

Artem Mikheev, Luc Vincent and Vance Faber

LizardTech, Inc.
821 2nd Ave., 18th Floor, Seattle, WA 98104, USA

{amikheev,lvincent,vxf}@lizardtech.com

Abstract

An algorithm is proposed for extracting high quality polygonal contours from connected objects in binary images. This is can be useful for OCR algorithms working on scanned text. It is also particularly interesting for raster-to-vector conversion of engineering drawings or other types of line drawings, where simple contours (known as Freeman chains) are grossly inadequate. Our algorithm is based on a relaxation technique, in which an initial polygonal approximation is iteratively refined until stability. Experiments demonstrate that this method consistently produces high quality results: the polygonal approximations obtained have a minimal number of corner points and remain extremely close to the original Freeman chain code.

Keywords: contour tracing, chain codes, polylines, engineering drawings, relaxation.

1 Introduction, Problem Definition

For a number of tasks related to document image analysis, one is interested in extracting an accurate polygonal representation of objects in binary scans (See examples of such scans in Fig. 1). Such representations can form the basis for OCR algorithms. They are also key in a number of tasks related to the understanding of graphics and engineering drawings. In addition, this raster-to-vector conversion generates compact object representations that are very useful for geometrical transformations and shape comparisons, and can also be imported in CAD/CAM systems.

(a) (b)

Figure 1: (a) Scanned text. (b) Portion of a scanned line drawing.

A straightforward way of performing this conversion is to use a contour tracking method and generate Freeman chain codes for all the contours in the binary image [3]. This can be done using 4-connectivity or

8-connectivity (sometimes 6-connectivity), and therefore results in polygonal object representations where all vectors have unit length and can have respectively 4 or 8 possible orientations. Neighboring vectors with the same orientation can then be combined into longer vectors, thereby reducing the overall number of vectors in the polyline representation without losing any information. Let us call this the *basic polyline representation.* An example of a 4-connected basic polyline representation is shown in Fig. 2.

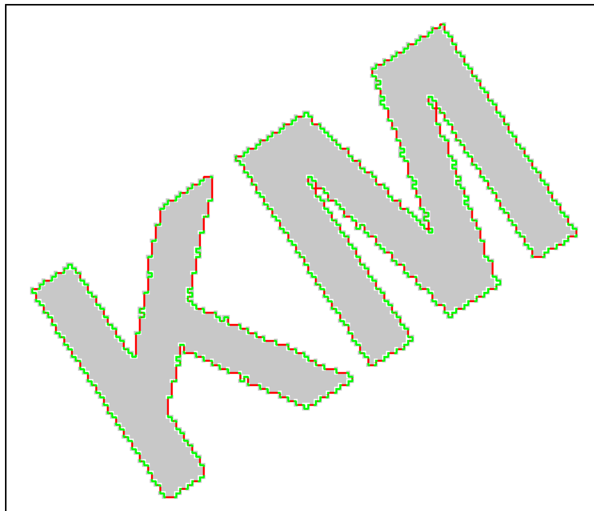

Figure 2: *Basic contour representation via polyline: the extremities of the vectors forming the polyline are shown with squares and simply represent the "corners" of the chain code representation.*

While this simple method preserves all the information contains in the original bitmap, it also has substantial drawbacks: resulting polylines are "jaggy", which is visually unpleasant. They are also inefficient for representing object boundaries not in one of the 4 or 8 main orientations, and typically generate an excessive number of vectors, or contour elements. This makes it difficult to use them for applications requiring a compact and flexible object representation.

The present paper proposes a new method for raster-to-vector conversion of material such as shown in Fig. 1. Our algorithm starts from a basic polyline representation (see Fig. 2). Points along this basic polyline are first grouped into clusters. This is done in such a way that for any cluster, the total deviation between cluster points and the line formed by joining the extremities of the cluster does not exceed a preset tolerance parameter ε. A relaxation process is then used to iteratively improve the location of the corner points defined as the extremities of each cluster. This algorithm is shown to be efficient, to produce esthetically pleasing results, and to preserve important structural details of the contour, such as smooth regions and sharp corners.

2 Review of Existing Methods

Since the seventies, a large number of methods have been proposed to extract polygonal approximation of raster objects. Some of these methods have been reviewed in recent papers by Rosin [8] and by Kadonaga et al [5]. For example, Cheng and Hu base their method on curvature approximation [1]. Fu et al. propose a method also based on curvature [4] while Douglas and Peucker use sequential contour subdivisions [2]. Pei and Horng's method does not use curvature either, relying instead on spatial shifting [7].

A quick literature survey shows that one of the most popular published methods remains the classic Pavlidis and Horowitz algorithm [6]. It is based on iterative split and merge steps, which progressively refine an initially trivial polygonal approximation. Basically, at any given step in the process, a polygon element (vector) is split in two if any of the contour pixels it approximates is farther than a maximal deviation δ. When this is the case, the vector is split in two at the point of maximal deviation. The process is iterated until all pixels deviate by no more than δ from the polygonal approximation. Along the way, neighboring polygon elements with the same orientation can also be merged.

The second most popular set of methods is based on curvature and basically aims to pick polygon corners as the contour pixels with maximal curvature.

The motivation for the work reported in this paper started from our observation that none of the published methods we experimented with worked satisfactorily. Curvature based methods worked fine with smooth data but tended to fall apart when applied to lower quality data. The Pavlidis and Horowitz method was generally more reliable, but often produced more corner points than we would have liked, for a given maximal deviation. In addition, all these methods constrained extracted corner points to be located on original contour pixels, which limits the accuracy that can be achieved. For all these reasons, we developed the new algorithm described in this paper.

3 How a Direct Solution Might Work

If all we are concerned with is that the maximal deviation between the original curve (chain code) and the computed polyline be bounded by parameter ε, we can propose a very straightforward method. From now on, this method will be referred to as *direct solution.* It works as follows:

Algorithm: Direct computation of polyline from chain-coded contour

- `Input: closed chain of length` N
- `Initialization:`
  - `stroke_start` $\leftarrow$ `first note of chain`
  - `stroke_end` $\leftarrow$ `stroke_start` $+1$
  - `resulting polyline` P `initialized to array of vectors of size 0`
- `While there remain chain elements to scan, do:`
  - `While all nodes between stroke_start and stroke_end` $+1$ `are within` ε `of the vector defined by (stroke_start,stroke_end` $+1$`), do:`
    `stroke_end++`
  - `(stroke_start,stroke_end) defines next vector of polyline` P
  - `stroke_start` $\leftarrow$ `stroke_end`
  - `stroke_end` $\leftarrow$ `stroke_end` $+1$

This algorithm is just about a simple as they come. As expected, and as illustrated by Fig. 3, it does not work very well in most cases: with small values of parameter ε, extracted polylines still have more vectors than one would like. On the other hand, larger values of ε can result in poor quality results, where corner points of the polyline are shifted compared to what the eye expects the correct position to be. However, as simplistic as it is, this method forms a good starting point for the algorithm described in this paper.

4 Proposed Algorithm

The algorithm proposed in this paper relies on the following simple observations:

- By only scanning the original contour in one direction (clockwise or counterclockwise), we introduce a systematic bias in corner point locations: all of them may be shited a few pixels compared to what their ideal location should be. This problem could be avoided, and a higher quality polyline would be obtained if we iteratively refined corner points by allowing them to shift compared to their original location. Our algorithm achieves this using a relaxation technique.

- Even if polyline corner points are positioned at the best possible location along the original contour, an even better approximation might be obtained by not constraining corner points to be located on image pixels. In other words, we should allow corner points to be located anywhere in the Euclidean space, not only at discrete locations defined by the grid. One can argue that this way, subpixel accuracy results may be produced.

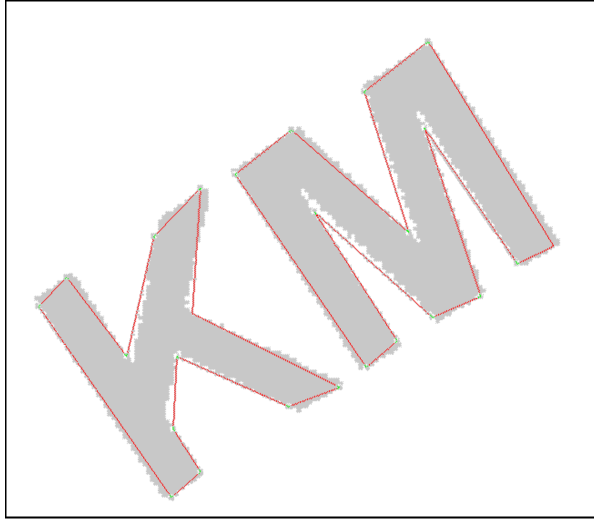

Figure 3: *Typical output of "direct solution" (with $\varepsilon = 3$): while the number of contour elements has been substantially reduced, the quality of the result is very low and unusable in practice.*

Let us now describe the algorithm in detail. First, we define a *cluster* of points of size m as a group of m consecutive points in a contour. A contour C of length n can therefore be divided into k non-overlapping clusters (γ_i), such that $\sum_{i=1}^{k} \text{size}(\gamma_i) = n$. Our algorithm is based on an initial clustering, followed by an iterative cluster relaxation. Like the direct solution described in Section 3, this algorithm relies on a parameter ε, characterizing the maximal deviation that can be tolerated between the resulting polyline and the original object contour. It can be decomposed into four major steps:

1. Upsampling and smoothing of initial contour
2. Extraction of initial cluster estimates
3. Iterative cluster relaxation
4. Computation of resulting polyline from clusters resulting from relaxation step.

These steps are described below.

4.1 Upsampling and contour smoothing

As often in many image analysis applications, a basic preprocessing of our data is beneficial. The idea here is to perform a slight smoothing of the original data, prior to computing initial cluster estimates. A simple convolution applied on the original contour pixels could be sufficient trick, but would produce floating point coordinates and slow down subsequent processing.

To prevent this, we first upsample the contour by a scale factor S (in our implementation, $S = 4$ was used), smooth the resulting contour and then round the resulting coordinates to the nearest integer. Denoting by $p_i = (x_i, y_i), i = 0$ to n the consecutive points of initial contour C, this simple smoothing operation is described as:

$$\text{For any } i \in [1; n], \quad p'_i = \text{round}(S \times (a \times p_{i-1} + b \times p_i + c \times p_{i+1})/(a + b + c)),$$

where, by convention, $p_0 = p_n$ and $p_{n+1} = p_1$. a, b, and c are three positive integers. As its name indicates, the 'round' operator rounds a floating point number to the nearest integer. In our experiments, a, b and c were respectively equal to 1, 4, and 1. Different smoothing results would be obtained with different values, and in some applications, one may even consider smoothing using a larger window (e.g., 5-point neigborhood instead of 3-point neighborhood), or iterating the basic smoothing a few times.

An example of smoothed contour produced by this method is shown in Fig. 4. This step is obviously not required for the present method. However, it proved useful in practice in that it enables the relaxation part of the algorithm to converge faster.

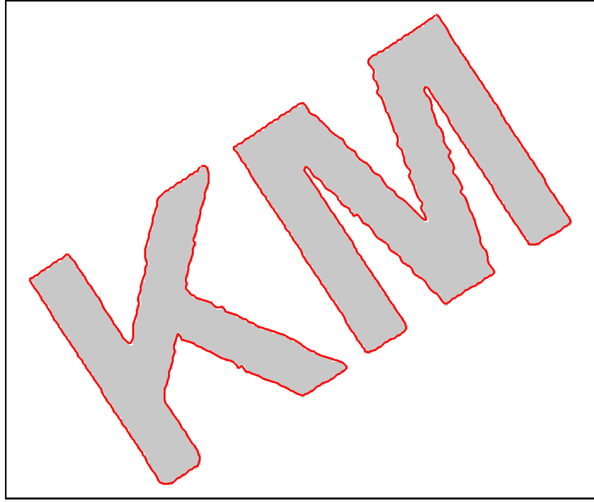

Figure 4: Result of initial upsampling and contour smoothing step

4.2 Construction of initial clusters of points

After smoothing, the next step is to determine initial clusters of points, given parameter ε. We do this by applying the direct method described in Section 3. This produces a decomposition of our initial (possibly smoothed) contour C into k non-overlapping clusters (γ_i), $i = 1$ to k.

The lines connecting the end points of each cluster are shown in Fig. 3. With each cluster γ_i, we now associate the best fitting straight line, in terms of mean square error. This straight line can be described by an equation like:

$$A_i x + B_i y + C_i = 0.$$

The following information is now associated with each cluster γ_i:

- Its size, or length $\text{size}(\gamma_i)$;
- An array AR_i of $\text{size}(\gamma_i)$ contour points, each given by their x, y coordinates;
- Three parameters A_i, B_i and C_i characterizing the best fitting line for this cluster;
- The mean square error E_i corresponding to best fitting line.

4.3 Iterative cluster relaxation

The core of the proposed algorithm is the iterative relaxation step. Its goal is to iteratively reassign pixels towards the extremities of each cluster to neighboring clusters when it improves the overall polygonal approximation. This process is iterated over all the clusters until stability is reached, that is, until the overall approximation quality no longer improves, or until a maximum number of iterations has been reached.

The relaxation procedure works as follows: for each iteration, we consider the pairs of neighboring clusters (γ_1, γ_2), (γ_2, γ_3), $\ldots$, (γ_{k-1}, γ_k), (γ_k, γ_1) one after the other. Each iteration is therefore decomposed into k primitive relaxation steps.

Lets us consider the adjacent clusters γ_i and γ_{i+1}. Let $L = \min(\text{size}(\gamma_i), \text{size}(\gamma_{i+1}))$, and let $q >= 2$. The relaxation process considers the L/q last points of cluster γ_i and the L/q first points of cluster γ_{i+1}. This is called the *relaxation interval.* In our experiments, we used $q = 4$. The $2 \times L/q$ cluster points p in the relaxation interval are then examined in sequence: if the distance between p and the line associated with cluster γ_i is smaller than the distance between p and the line associated with cluster γ_{i+1}, we increment a counter count_i, associated with γ_i. Otherwise, counter count_{i+1}, associated with γ_{i+1}, is incremented.

Once all the $2 \times L/q$ pixels have been visited, the value of each counter is considered. If $\text{count}_i > \text{count}_{i+1}$, then the $(\text{count}_i - \text{count}_{i+1})/2$ first points of cluster γ_{i+1} are reassigned to cluster γ_i, thereby shifting the position of the corner point defined as the junction between these two clusters. Similarly, if

$\text{count}_i < \text{count}_{i+1}$, then the $(\text{count}_{i+1} - \text{count}_i)/2$ last points of cluster γ_i are reassigned to cluster γ_{i+1}. If both counters end up with the same value, neither cluster is modified.

In more algorithmic terms, this process can be described as follows:

Algorithm: Primitive relaxation step for two neighboring clusters

- `Input:`
  - `neighboring clusters` γ_i `and` γ_{i+1}
  - `parameter` q
- `Initialization:`
  - $l \leftarrow \texttt{round}(\min(\texttt{size}(\gamma_i), \texttt{size}(\gamma_{i+1}))/q)$
  - `Relaxation interval` I: `list of` l `last pixels of` γ_i `and` l `first pixels of` γ_{i+1}.
  - $\texttt{count}_i \leftarrow 0$;
  - $\texttt{count}_{i+1} \leftarrow 0$;
- `For each pixel` $p \in I$:
  - `If` $\texttt{dist}(p, A_i x + B_i y + C_i = 0) < \texttt{dist}(p, A_{i+1} x + B_{i+1} y + C_{i+1} = 0)$ `then`
    $\texttt{count}_i \leftarrow \texttt{count}_i + 1$
    `else`
    $\texttt{count}_{i+1} \leftarrow \texttt{count}_{i+1} + 1$
- `If` $\texttt{count}_i > \texttt{count}_{i+1}$ `then`
  - `Increase` γ_i `by adding to it the` $(\texttt{count}_i - \texttt{count}_{i+1})/2$ `first pixels of` γ_{i+1}.
- `Else if` $\texttt{count}_{i+1} > \texttt{count}_i$ `then`
  - `Increase` γ_{i+1} `by adding to it the` $(\texttt{count}_{i+1} - \texttt{count}_i)/2$ `last pixels of cluster` γ_i.

A full iteration cycle consists of processing all the k pairs of neighboring clusters. At the end of each iteration, the overall mean square error is computed by taking the weighted average of the per-cluster mean square error E_i, where the weight corresponds to the number of pixels in each cluster. In other words, this corresponds to computing the mean square distance between each pixel and the line representing the cluster it belongs to.

The process stops when either:

- The pre-set maximal number of iterations has been reached.
- The overall mean square error described in the previous paragraph stops decreasing monotonically.

In our experiments, a maximum number of iterations of about 30 was usually more than enough.

4.4 Computation of final outline

At the end of this process, we are left with k clusters γ_i, each described as a line with equation $A_i x + B_i x + C_i = 0$. The final polyline is obtained by considering the k pairs of lines describing neighboring clusters, and computing the coordinates of their intersection. This results in a polyline described as a list of k corner points.

Note that the corner points obtained are generally not located on pixels of the original image. This is in fact one of the strengths of the proposed algorithm: by allowing corner points to be located "between" pixels, we essentially achieve sub-pixel accuracy in our polyline approximation.

Applying this algorithm to the "KM" image used before produces the polyline shown in Fig. 5, which demonstrates the quality of the results we are able to achieve with this method. Additional examples are shown in Figs. 6 and 7.

5 Discussion, Prospects

A new algorithm was proposed for computing high-quality polygonal contour approximations from chain-code contour representations. Being based on a relaxation process which involves a fair amount of computation, this algorithm is less efficient than other published methods. However, since the amount of data processed is typically very small (contour data is often rather sparse in bitmap images), this increase in

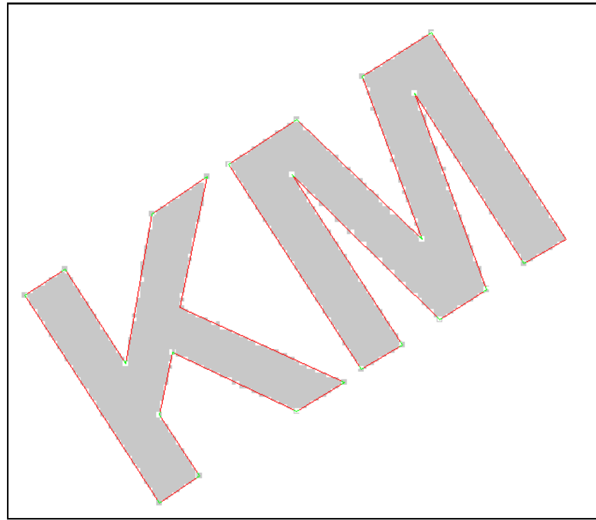

Figure 5: Final outline after relaxation procedure.

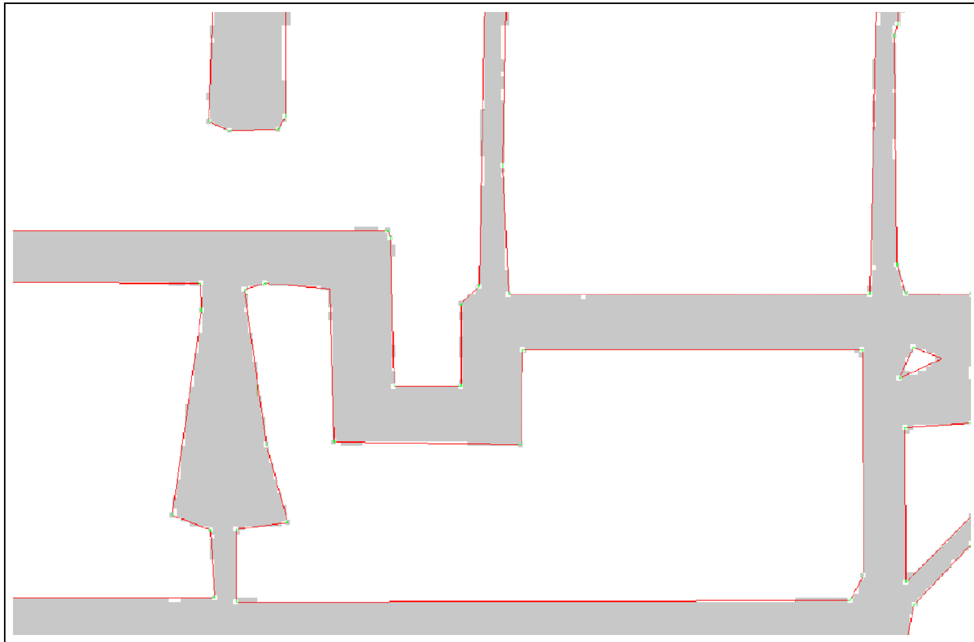

Figure 6: Result of our relaxation based polyline extraction on a line drawing scan.

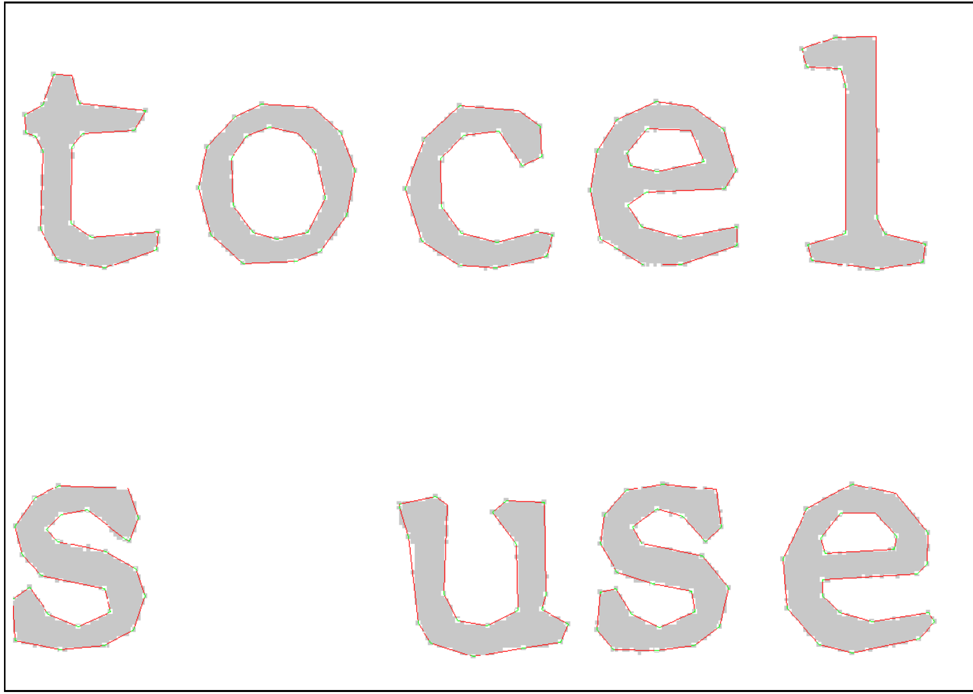

Figure 7: *Result on scanned text.*

computation time is usually negligible when compared to the time taken by susequent steps, such as OCR. Also, importantly, this algorithm consistently produces high quality results, where polygon corners are approximated with sub-pixel accuracy.

The algorithm depends on very few parameters: the maximal deviation ε, the maximal number of iterations that can be used during relaxation, and the initial smoothing parameters. It is therefore very easy to use and to tailor to any given application involving raster to vector conversion. In short, the algorithm proved simple and effective.

Further improvements are currently considered. For example, the size of the relaxation region could be improved to better account for cases where a small cluster is adjacent to a much larger one. This would enable the algorithm to converge faster. We are also investigating a cluster merging step—which could also be part of the relaxation—to better handle those rare cases where neighboring clusters can be described with a single line segment.

References

[1] Cheng and Hsu. “Parallel Algorithm for Corner Finding on Digital Curves”, *Pattern Recognition Letters,* Vol. 8, pp. 47–54, 1988.

[2] Douglas and Peucker. “Algorithms for the Reduction of Number of Points Required to Represent a Digitized Line or its Caricature”, *The Canadian Cartographer,* Vol. 10, pp. 111–122, 1973.

[3] H. Freeman. “On the Encoding of Arbitrary Geometric Configurations”, *IEEE Transactions on Computers,* Vol. C10, pp. 160–168, 1961.

[4] Fu, Yan and Huang. “A Curve Bend Function Method to Characterize Contour Shapes”, *Pattern Recognition,* Vol. 30(10), pp. 1661–1671, 1997.

[5] T. Kadonaga and K. Abe. “Comparison of Methods for Detecting Corner Points from Digital Curves”, R. Kasturi & K. Tombre, Eds.: Graphics Recognition Methods and Applications, Springer-Verlag, pp. 23–34, 1996.

[6] Pavlidis and Horowitz. “Segmentation of Plane Curves”, *IEEE Transactions on Computers,* Vol. 23, pp. 860–870, 1974.

[7] Pei and Horng. "Corner Point Detection Using Best Moving Average", *Pattern Recognition*, Vol. 27, pp. 1533–1537, 1994.

[8] Rosin. "Accessing the Behaviour of Polygonal Approximation Algorithms", *British Machine Vision Conference,* 1998.