Segmentation of Planar Curves and the Iterative Thresholding Technique *

Xingang Huang **Feng Zhao**
Dept. of Computer and Information Science
The Ohio State University
2015 Neil Avenue
Columbus, Ohio 43210
{huang,fz}@cis.ohio-state.edu

Abstract

We review different curve segmentation techniques and algorithms and identify the four main components in a curve segmentation algorithm. We abstract these main components as independent software modules so that different techniques can be combined to derive existing and new curve segmentation algorithms. We also develop the iterative thresholding technique to find appropriate threshold parameters for a large class of optimization algorithms. The technique is applied to the split and merge algorithm in curve segmentation and the experimental results show that the technique is capable of selecting appropraite threholds to explore the inherent properties and structures of data.

*The work is supported in part by NSF NYI grant CCR-9457802, ONR YI grant N00014-97-1-0599, and a Xerox grant to the Ohio State University.

1 Introduction

Planar Curve segmentation is a key problem in computer vision and pattern recognition. Segmentation results can be used to extract more primitive components, either salient points or curve segments, for many vision tasks such as salient feature detection, concise and accurate curve representation, and shape modeling and recognition.

The research on 2D curve segmentation has been active for many years with a huge literature accumulated. Most of these research work can be classified into two classes:

1. Salient point detection.
2. Edge approximation.

The first class includes all the work that label each point on a curve either as significant or insignificant according to some local properties. The significant points are believed to be information richer and more representative at least within the corresponding application context. Most of the commonly used properties are related to curvature, such as curvature extrema ([2, 24, 11, 1]) or zero curvature ([9, 17]). Other properties are also proposed such as the B/F-saliency by Fischler et al [8, 7]. These properties are usually noise sensitive and scale independent, and scale space techniques are often used to resolve these weaknesses [1, 32, 33, 23].

The work in the second class seeks not only to recover the salient points but also to approximate the resulted curve segments with some standard parameterized curves, such as straight lines [19, 21, 30, 18], circular arcs [15, 22] or their combinations with other high order curves [6, 16, 20, 25, 26, 27, 31]. The problem can be treated as a compression problem seeking to increase both the compression ratio (decrease the description length) and the approximation accuracy. These two goals are generally incompatible to each other and need to be carefully balanced in a specific application.

Among these two approaches, the edge approximation approach is more general. It seeks to fit every point on a curve instead of just a few of salient one so that global objective evaluation creteria can be established to evaluate the segmentation results. On the other hand, the salient point detection approach is simpler and can be used in applications of less requirements. It can also be used as a component in an edge approximation algorithm for locating segmentation points preliminarily. In this paper, we will only focus on the edge approximation approach.

In [7] Fischler et al. have demonstrated convincely that curve segmentation is not a generic, task independent problem. They also gave a detailed discussion about the main factors which may influence curve segmentation. We summarize these main factors briefly below in a slightly different organization:

- *Purpose*: What is the segmentation for? For example, the task of classifying an machined part definitely requires a different segmentation from the task of inspecting the defects in the same part.
- *Curve properties*: A class of curves need to be processed by an segmentation algorithm usually have some same properties. For example, some algorithms may need to deal with contours of man-made objects which have a large portion of straight line segments; Some may need to deal with natural objects which seldom have straight line segments.
- *Evaluation*: How to evaluate segmentation results? This includes the evaluation of approximation accuracy, compression ratio and their combination. Different

evaluation criteria will certainly lead to different "good" segmentation results.

Many curve segmentation algorithms and even more of their variations have been developed in curve segmentation. These algorithms have made their contributions on some factors above-mentioned. However, they usually also suffer from the lack of generality by unnecessarily committing to some fixed treatments on other factors. In this paper, we are interested in deriving curve segmentation algorithms by combining different techniques. More specifically, we want to address the following questions: What are the main components of a curve segmentation algorithm? What kind of different techniques exist for each component? How can these different techniques be combined to derive existing and new curve segmentation algorithms?

We will also develop the Iterative Thresholding technique for the exploration of the inherent properties of a dataset. This technique is general and we have experimented it on curve segmentation to find appropriate thresholds for the split and merge algorithm. The results are very encouraging.

The rest of this paper is organized as following: Section 2 analyzes the main components of a curve segmentation algorithm, the different techniques for each component, and how to combine them to derive existing and new segmentation algorithms; Section 3 develops the iterative thresholding technique; Section 4 presents the experimental results ; Section 5 concludes and summarizes the whole paper.

2 Main components in curve segmentation and their combinations

Based on our review of the current literature, a general curve segmentation algorithm can be decomposed into four main components: *approximation*, *evaluation*, *segmentation refinement* and *segmentation optimization*. We will discuss next these four component and the different techniques developed for them, and formalize them as independent software modules so that existing and new segmentation algorithms can be derived by flixibily combining different segmentation techniques.

2.1 Approximation

The approximation component deals with the problem of approximating a segment of curve using standard parametrized curve as well as how to compute the point-wise approximation error.

Curve segments can be approximated by straight lines, constant curvature curves (straight lines and circular arcs), conic sections, splines or other high order curves. Most applications studied require the continuity in the approximation, and more specifically, the approximating curve segments passing through the two endpoints of the approximated segment. With this constraint, the two endpoints of a segment can uniquely determine a straight line approximation. Generally, least square method is the most commonly used to determine the parameters for an approximating segment.

Different ways for computing point-wise approximation error usually exist. For example, the error for a straight line approximation can be computed based on functional distance or normal distance; the error for an elliptical approximation can be computed based on Euclidean distance or algebraic distance [27]. Because different point-wise error norms can lead to different segmentation results, approximation implementations that use same form of approximating curves but different point-wise error norms should be considered as different implementations.

2.2 Evaluation

We have mentioned in the introduction section that a curve segmentation problem can be viewed as a compression problem with the goals of decreasing the description length and increasing approximation accuracy. Generally an objective global cost function C can be defined as $C(L, E)$: a function of L the description length and E the approximation error with the constraint satisfying the condition that C is monotonically increasing on L and E.

One simple way to compute description length L is to use the number of segments in a segmentation. A more general way is to assign a weight to each segment and sum them. The weight can be determined by the complexness of the approximating curve, the more complex the approximating curve the higher the weight. For example, we can use the number of parameters required for an approximating curve segment as its weight. The literature that study this factor on curve segmentation is few. One example is in [27], where N_{segs}/N_{points} and (N_{paras}/N_{points}) (N_{segs}: number of segments, N_{points}: number of points, N_{paras}: number of parameters) are used to compute description length (normalized).

The computation of approximation error E of a segmentation has been well studied in the literature. Commonly used error Norms include E_2 (square integral error norm), E_σ (mean square error), E_∞ (maximum error) and E_r (relative error). However, few literature have stated explicitly that an error norm actually contains two parts, the segment error norm E^{seg}, which computes the approximation error of a segment using the point-wise approximation error, and the aggregate error norm E^{agg}, which computes the error of a segmentation based on all segment errors. Using these notations, the commonly used error norms mentioned above can be defined as following:

- E_2:

$$\begin{aligned} E &= E^{agg}(E_1^{seg}, E_2^{seg}, \cdots, E_n^{seg}) \\ &= \sum_i E_i^{seg} \\ E_i^{seg} &= \sum_{p \in S_i} E_p^2 \end{aligned}$$

- E_σ:

$$\begin{aligned} E &= E^{agg}(E_1^{seg}, E_2^{seg}, \cdots, E_n^{seg}) \\ &= \max_i E_i^{seg} \\ E_i^{seg} &= \frac{1}{|S_i|} \sum_{p \in S_i} E_p^2 \end{aligned}$$

- E_∞:

$$E = E^{agg}(E_1^{seg}, E_2^{seg}, \cdots, E_n^{seg})$$

$$\begin{aligned} &= \max_i E_i^{seg} \\ E_i^{seg} &= \max_{p \in S_i} E_p \end{aligned}$$

- E_r:

$$\begin{aligned} E &= E^{agg}(E_1^{seg}, E_2^{seg}, \cdots, E_n^{seg}) \\ &= \max_i E_i^{seg} \\ E_i^{seg} &= \frac{1}{|S_i|} \max_{p \in S_i} E_p \end{aligned}$$

We note that E^{agg}s in these error norms take the form of either the *sum* function or the *maximum* function. We call an error norm *Summable* if it uses summation to compute the aggregate error norm or *Uniform* if it uses maximization. We will see later that some algorithms actually depend on whether the error norm chosen is summable or uniform.

One property that both summable error norms and uniform error norms have is that they are *aggregatable*, i.e, a sequence of errors can be assembled in any ways to compute the overall approximation error:

$$\begin{aligned} E^{agg}(E_1, E_2, E_3) &= E^{agg}(E^{agg}(E_1, E_2), E_3) \\ &= E^{agg}(E_1, E^{agg}(E_2, E_3)) \\ &= E^{agg}(E^{agg}(E_1, E_3), E_2) \end{aligned}$$

This property ensures that divide and conquer can be used in segmentation algorithms to optimize the approximation error and it is necessary for many current algorithms such as dynamic programming and split and merge.

Given L and E, a global cost function $C(L, E)$ can be defined by specific applications. However, generally the cost function C does not have the *aggregatable* property and therefore, the corresponding optimization algorithms are harder to obtain. This is the reason why most existing curve segmentation algorithms do not directly optimize a global cost function to segment curves. One example of using the global cost function as a performance criterion is in [27], where the cost function takes the form of $C(L, E) = L * E$ (the FOM-1 and FOM-2 merit functions).

2.3 Segmentation refinement

The segmentation refinement component deals with the problem of breaking an existing segmentation into more shorter segments so that the curve can be more accurately approximated. This problem can be further divided into two sub-problems: 1) Which segment should be chosen to break? 2) How to break a chosen segment?

For the segmentation algorithm which seeks to bring the approximation error below a given threshold, such as split and merge and scan along algorithms, if the underlying error norm is uniform, then any segments that has an approximation error exceeding the threshold should be broken. If the underlying error norm is summable, the usual practice is to break the segment of the maximum approximation error. This may not break the problematic segments every time, but it will eventually break all the problematic segments with the cost of possible over-segmentation on some segments. This shows different error norms can constrain the possible segmentation refinement techniques that can be used with them. Typical split and merge algorithms [21, 29] do have two different versions for summable and uniform error norms. The scan along algorithms can support only uniform error norms.

As to the question of where to break a given segment, different algorithms uses different criteria. Some commonly used techniques are listed below:

- Maximum deviation: break the segment at the point where the maximum point-wise approximation error occurs. This is typically used by split and merge algorithm and also by the non-parametric binary tree segmentation algorithm [27].
- Scan along error: sequentially test the break points starting from one end of the segments until the approximation error exceeds the given threshold and break there. This is used by various scan along algorithms and also by the dynamic focusing algorithms [12].
- Local significance: break at all the points that have a local significance measure above a threshold. Most local saliency detection algorithms use this criteria with different definitions of saliency.

2.4 Segmentation optimization

The segmentation optimization component deals with two problems: 1) Reduce over-segmentation and, 2) Reduce the approximation error by adjusting the locations of segmentation points.

In split and merge, the strategy for the first problem is to merge two adjacent segments whose merge gives the least approximation error rise under the condition that the overall approximation error is still below a given threshold. The binary tree segmentation [14, 27] algorithm can be viewed as a restricted version of split and merge algorithm that allows only the merge of siblings with the same parent.

For the second sub-problem, Pavlidis proposed an end-point adjustment algorithm [19] that repeatedly moves a segmentation point left or right in small steps if the move can decrease the overall approximation error until no move exists for further approximation error reduction. This is a greedy procedure that guarantees approximation error reduction. It can be applied to segmentations obtained from any algorithms.

2.5 Existing algorithms: their contributions and over-restrictions

With the above four main components in mind, we now review the existing curve segmentation algorithms to see where are their main contributions and where their implementations over-restrict their generalities.

The first class of algorithms is dynamic programming [4, 10]. Dynamic programming algorithms have only the first two main components: approximation and evaluation. They only require the global evaluation function to be aggregatable to work. They do not have any over-restrictions. However, dynamical program algorithms require the number of segments to be fixed and they explore all possible different segmentations exhaustively which leads to its impracticable computational cost.

Split and merge [19, 21, 29, 5] approach covers a large class of segmentation algorithms. The main contributions of split and merge approach include: 1) It has all the four main components, which makes its algorithms generally generate better segmentation results according to the global evaluation function compared with other algorithms except dynamic programming algorithms; 2) It distinguishes summable error norms from uniform error norms and can use summable error norms where most other algorithms can not except dynamic programming; 3) The end-point adjust procedure first proposed by split and merge algorithms is generally applicable to all other algorithms. Most implementations of split and merge algorithms suffer from over-restrictions. In the approximation component, current algorithms commit themself unnecessarily to straight lines or constant curvature curves. In the break point location part in the curve refinement component, almost all algorithms commit themself unnecessarily to the maximum deviation technique.

Binary tree algorithms [14, 27] are restricted versions of split and merge algorithms. Though in [27], Rosin et al proposed an extension to allow the merging of siblings of different parents, the extension makes the algorithm more complex and cumbersome. Generally binary tree algorithms have worse performance compared with non-restricted split and merge algorithms. Their implementations also suffer from the same over-restrictions that the split and merge algorithms suffer.

Scan along algorithms address only the breakpoint location part in the segmentation refinement component. This is where their main contributions are. Its algorithms usually do not have the segmentation optimization components which leads to their poorer performance compared with split and merge algorithms. Implementations also suffer from over-restrictions in the approximation component and support only uniform error norms. The dynamic focusing approach [12, 31] can be viewed as scan along algorithms but operates in the reverse direction.

Various local saliency detection algorithms address only the breakpoint location part in the segmentation refinement component, which is their main contribution. They usually have only the curve refinement component and can not be treated as full-fledged edge approximation segmentation algorithms. They also can not provide a global evaluation function to evaluate their results. These algorithms can be used as specific implementations of the break location part in an full edge approximation algorithm.

Algorithms using different forms of approximating curves make their contributions to the approximation component by providing mathematical formula for computing the parameters of the approximating curve and point-wise approximation error. The approximation component is more closely related to computational geometry rather than curve segmentation. However, the techniques developed from it should be shared by all curve segmentation algorithms.

Another class of algorithms [28, 3, 13, 5] are classified as the chain code algorithms by the literature. In chain code, each point on a curve is represented by its length S from the starting point and its tangent direction ψ (chain code) or its tangent differentiation (differential chain code). Early work uses a small fixed number of chain codes (typically 8) which results in very noisy $\psi - S$ curves. Later work [5] uses more directions and noise filtering techniques to smooth the $\psi - S$ curves. The chain code method can be viewed as a transformation technique. After a curve is transformed, the above-mentioned segmentation techniques can be used to segment the curve in its new representation. We classify the chain code method and other transformation methods as pre-processing techniques. We do not include this pro-processing stage as a main component in curve segmentation. The main contributions of chain code method algorithms are their explorations of the properties of this transformation, such as that a polygonal approximation to a curve's $\psi - S$ representation is actually a constant curvature curve approximations to the curve in its original space.

To summarize, split and merge algorithms provide a general framework for curve segmentation algorithms that contains all four main components: approximation, evaluation, segmentation refinement and segmentation optimization. Various other existing algorithms make their contributions to one or some of the components by offering their specific implementations. Now we are interested in abstracting these components as independent modules, with different techniques as different implementations for one of the modules, so that existing and new segmentation algorithms can be derived by combining different implementations of all modules. We will discuss this in detail in the next subsection.

2.6 Independent modules and their combinations to derive concrete segmentation algorithms

We formulate the above-mentioned main components as independent modules, or in the terms of our implementation language C++, abstract interface classes. Each interface class can have different implementations using different techniques. A concrete algorithm can be instantiated by the multiple inheritance of the implementations of all the interface classes. We have written a C++ library according to the above principles that supports a number of different implementations of each module. The library can be extended by adding more implementations.

The current library has four abstract interface classes (actually six, with the other two for curve data management and segmentation management which are just implementation details):

1. Approximation: defines methods to compute the approximation of a segment and the point-wise approximation error. Current implementations include linear approximation using normal distance and linear approximation using functional distance.

2. Error norm : define methods to compute the approximation error of an segment given point-wise approximation error and aggregate approximation error given a sequence of segment approximation error, i.e, methods to compute E^{seg} and E^{agg}. Current implementations include E_2, E_σ, E_∞.

3. Break point location: define a method to find break points for a given segments. Currently it has only one implementation that uses the maximum deviation method as described in section 2.3.

4. Algorithm: define a method to segment the curve. Current implementations include the two different versions of split and merge algorithms based on summable error norm or uniform error norm, and dynamic programming algorithm.

Note this is not directly translated from the four main components, because some parts of the segmentation refinement component and segmentation optimization component are not easy to be abstracted to have common interface that can be shared by different algorithms, and these parts are left to be implemented by the implementation subclasses of the algorithm class. Also, the endpoint adjustment algorithm is implemented as a public method and can be accessed by any subclasses of the algorithm class.

Under this architecture, different approximation techniques, error norms and break point location techniques can be plugged into the library any time as specific implementations of the corresponding abstract classes to extend the capabilities of the libraries. Different segmentation algorithms can also be implemented and added into the library without unnecessarily committing themself to some fixed approximation, error norm and break point location implementations. The current library supports only a small number of implementations to each abstract class but it can already derive dozens of different segmentation algorithms. The library code will be published in public domain and it is our hope that it can be extended and shared by other researchers to facilitate the developing and experimenting of different segmentation algorithms for different applications.

3 The Iterative Thresholding Technique

The iterative thresholding technique is inspired by the split and merge algorithm [21], though it is a general technique not limited to curve segmentation. The split and merge algorithm requires a error threshold to be given as a parameter. It then generates a segmentation result that has an approximation error below the threshold. In the merge stage, the algorithm uses greedy methods to reduce the number of segments under the threshold constraint while at the same time optimize the approximation error if the number of segments is not increased. This prevents the algorithm from over-segmenting a curve with respect to the given error threshold while at the same time seeks to obtain an approximation error as low as possible.

The split and merge algorithm requires an error threshold. How could we find an appropriate error threshold for an arbitrary curve? As has been discussed in the introduction section, this is generally application dependent. However, we still hope to find some criteria that can be established by the intrinsic curve properties alone.

Given an error threshold eT, the split and merge algorithm generates a segmentation that has an approximation error E satisfying $E < eT$. We have observed that for different eTs, the ratio eT/E varies. Most of the time it is close to 1. However, sometimes it can be much larger than 1. This means that we can reduce the threshold significantly and still get the same result, i.e., the threshold is stable at that point.

To find an appropriate threshold dynamically, we need a sequence of thresholds that generate different segmentation results and a heuristic to choose one from them. One natural way to find the next threshold that can give a better result (in the sense of smaller approximation error) is making it slightly smaller than the current approximation error, i.e., if current threshold eT_i gives an approximation error E_i, the next threshold eT_{i+1} can be set to $r * E_i$ where r is slightly less than 1(which is set to 0.99 in all the experiments of this paper). We call this method *iterative thresholding*. Using *iterative thresholding*, we can generate a sequence of thresholds that give segmentations of decreasing approximation error. To start it, we can just use a threshold that is arbitrarily large.

We propose two simple and general heuristics to evaluate thresholds generated by the iterative thresholding method. The first is called *Absolute Threshold Stability (ATS)* and the second is called *Relative Threshold Stability (RTS)*. Given a sequence of error threshold $\{eT_i\}$ and their corresponding approximation error $\{E_i\}$ generated by iterative thresholding, the ATS_i and RTS_i with respect to threshold eT_i are defined as:

$$ATS_i = \frac{e_{i-1}}{e_i}$$
$$RTS_i = \frac{ATS_i}{ATS_{i+1}}$$

The larger the heuristic value the better the corresponding threshold.

Since $E_{i-1} \approx eT_i$, ATS defines the stability of a threshold, i.e., to what fraction can a threshold decrease with the segmentation result unchanged. The RTS, on the other hand, compares the stability of two adjacent thresholds. It favors thresholds that are stable and have not so stable successors, i.e., thresholds at the transition of stability, where qualitative changes are likely to occur.

Though the iterative thresholding technique is illustrated by the split and merge algorithm in the above discussion, it is a general technique that can apply to a large class of algorithms. We now formulate the conditions under which it works.

Let $\mathbf{R}$ denotes the field real numbers, $\mathbf{I}$: the set of inputs, $\mathbf{O}$: the set of outputs, $\mathcal{G} : \mathbf{R} * \mathbf{I} \rightarrow \mathbf{O}$ an algorithm that takes an input and a threshold parameter and generates an output, $\mathcal{E} : \mathbf{O} \rightarrow \mathbf{R}$ an evaluation function, then the iterative thresholding technique applies if the two conditions below are satisfied:

1. $\forall I, t \quad (I \in \mathbf{I}) \wedge (t \in \mathbf{R}) \Longrightarrow \mathcal{E}(\mathcal{G}(t, I)) < t$
2. $\forall I, t, s \quad (t \in \mathbf{R}) \wedge (I \in \mathbf{I}) \wedge (\mathcal{G}(t, I) = O) \wedge (\mathcal{E}(O) = e) \wedge (e < s \leq t) \Longrightarrow \mathcal{G}(s, I) = O$

The first condition ensures the algorithm G always generates results that satisfy the threshold constraint. The second condition tells that for a threshold t and the evaluation value e of its output, all thresholds in the interval (e, t] are equivalent. This property ensures the iterative thresholding technique generates all representative thresholds and the ratio t/e is a good indicator of the stability of the threshold t.

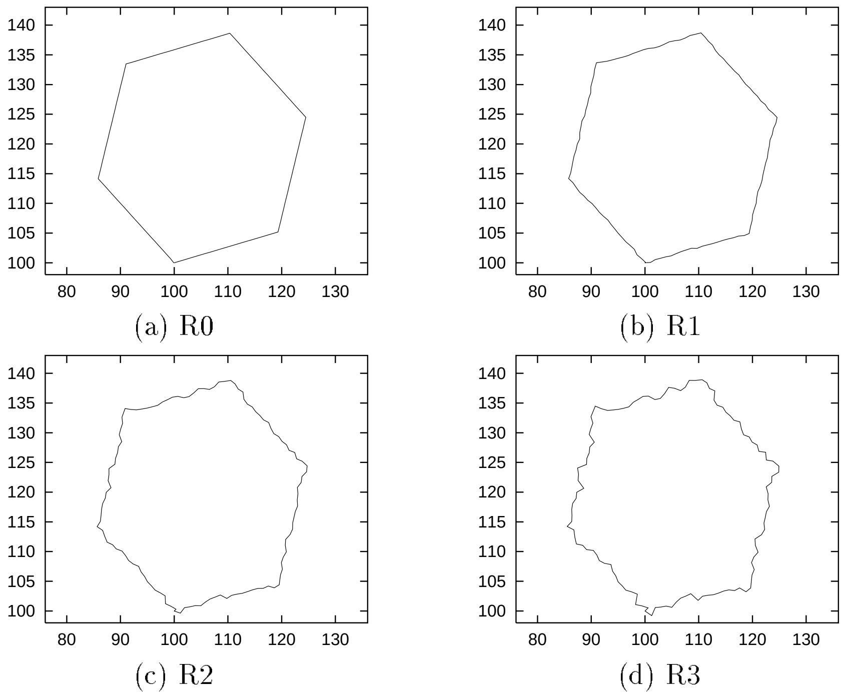
Note that the second condition need not be satisfied exactly for the iterative thresholding technique to be useful. The split and merge algorithm does not satisfy this condition exactly either because of its usage of some greedy methods in its optimization procedures.

The iterative thresholding technique generates a sequence of representative thresholds and their stability heuristics which can be used to explore the inherent properties of the input data under studying. In the next section, we will show that it can be used to distinguish noise from data and discover structures of different scales in curve segmentation.

The ATS and RTS are two simple and general heuristics. We can obtain more complex and application-oriented heuristics by adding other related parameters into the formula. For example, in split and merge, a heuristic may explicitly consider the effect of the number of segments. We have not done any research in this direction and we will explore this direction in our future work. Our experiments show that the two above general heuristics generate very good results for our experimental data.

4 Experimental Results

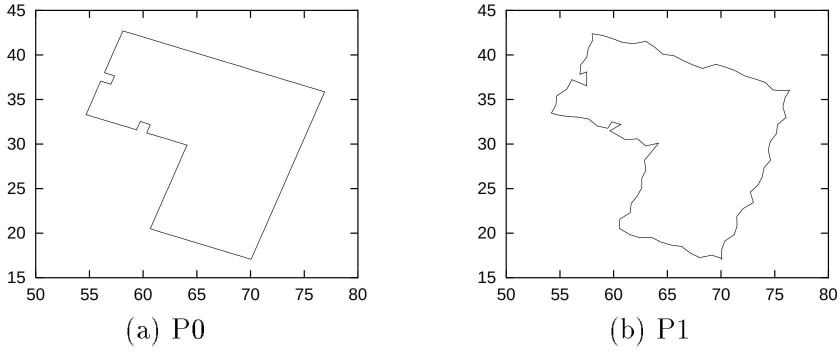
Two sets of data are used in this paper to demonstrate the performance of the algorithms derived from our combination approach and the effectiveness of the iterative



(a) R0 (b) R1

(c) R2 (d) R3

Figure 1: A regular hexagon and three of defective versions generated by the methods described in th text



(a) P0 (b) P1

Figure 2: A polygon with two levels of details and its defective version

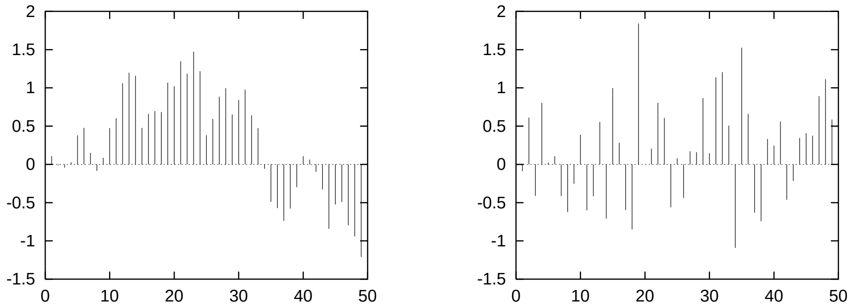


Figure 3: Noise sequences generated with different parameters. Left: (0.3, 0.9), Right: (0.69, 0.1)

thresholding technique. The first set of data includes a regular hexagon and three of its defective versions with different levels of noise as shown in Fig. 1. The second dataset includes a polygon with features of two different scales and its defective version as shown in Fig. 2.

The defective versions are generated as following: Let (x_i, y_i) be the i_{th} point on the original curve, θ_i be the normal direction of the i_{th} point, and $\{f_i\}$ a sequence of noise generated for the perturbation of the curve, then the i_{th} point (x'_i, y'_i) of the defective version is defined as:

$$
\begin{aligned}
x'_i &= x_i + f_i cos\theta_i &\quad (1)\\
y'_i &= y_i + f_i sin\theta_i &\quad (2)
\end{aligned}
$$

i.e., each point is perturbed along its normal direction by a randomly generated noise f_i.

The sequence of noise $\{f_i\}$ is generated using the below formula:

$$
\begin{aligned}
f_0 &= \sigma N(0,1) &\quad (3)\\
f_i &= \sigma N(0,1) + \rho f_{i-1} &\quad (4)
\end{aligned}
$$

where $N(0,1)$ is standard normal noise and ρ a decaying coefficient which is less than 1. The two parameters (σ, ρ) control the amplitude and the correlation of the noise sequence. In fact, we can derive:

$$
\begin{aligned}
f_i &= \sigma N(0,1) + \rho f_{i-1}\\
&= \sigma N(0,1) + \rho\sigma N(0,1) + \rho^2 f_{i-2}\\
&= \sigma(N(0,1) + \rho N(0,1) + \cdots + \rho^i N(0,1))
\end{aligned}
$$

So each f_i is normally distributed with mean 0 and standard deviation:

$$
\begin{aligned}
\sigma_{f_i} &= \sigma\sqrt{\tfrac{1-\rho^{2i+2}}{1-\rho^2}}\\
&\approx \tfrac{\sigma}{\sqrt{1-\rho^2}}
\end{aligned}
$$

The correlation between f_i and f_{i-1} is:

$$
\rho_{f_i, f_{i-1}} = \frac{E[(f_i - \mu_{f_i})(f_{i-1} - \mu_{f_{i-1}})]}{\sigma_{f_i}\sigma_{f_{i-1}}}
$$

	approximation	error norm	break point location	algorithm
SM1	linear approximation with normal distance	E_∞	maximum deviation	split and merge
SM2	same as SM1	E_2	same as SM1	split and merge
DP1	same as SM1	E_∞	NA	dynamical programming

Table 1: The implementation details of the algorithms used in this paper.

$$
\begin{aligned}
&= \frac{E(f_i f_{i-1})}{\sigma_{f_i}\sigma_{f_{i-1}}} \\
&= \frac{E(\sigma N(0,1) f_{i-1} + \rho f_{i-1}^2)}{\sigma_{f_i}\sigma_{f_{i-1}}} \\
&= \frac{\rho \sigma_{f_{i-1}}^2}{\sigma_{f_i}\sigma_{f_{i-1}}} \\
&\approx \rho
\end{aligned}
$$

We choose this noise generating method because it is purely random and the sequence it generates is correlated. Fig. 3 shows two noise sequences generated with parameters of $(0.3, 0.9)$ and $(0.69, 0.1)$. They have roughly the same standard deviation around 0.69 and therefore, roughly the same amplitude, but the second one oscillates much more than first one because the adjacent elements are less correlated.

The noise parameters for the three defective versions in the first dataset are: (0.1, 0.5), (0.3, 0.5) and (0.5, 0.5). The four hexagons are named R0-R3 according to their increasing level of noise. The noise parameter for the defective version polygon in the second dataset is (0.3, 0.5) and the two polygons are named P0 and P1 respectively.

We used three algorithms: $SM1$, $SM2$ and $DP1$, which are implemented by combining different implementations of different modules, in our experiments. The details of their implementations are described in Table 1.

Fig. 4 is a validation check. It has been shown in the literature [29] that split and merge algorithm generates near optimal results if appropriate error thresholds are given. Fig. 4 also demonstrates this. In all four hexagons, the error thresholds used are the same: 5.5 for $SM1$ and 80 for $SM2$. The segmentation results are exactly the same except for the most noisy hexagon where the differences are one point shifts.

Fig. 5 shows how iterative thresholding can be used to choose appropriate thresholds. The segmentation algorithm used is $SM1$. Fig. 5(a) and (b) plot the ATS and RTS heuristics of thresholds generated by the iterative thresholding technique against the number of segments of the corresponding thresholds. Both heuristics have very sharp peaks when the number of the segments is six. We also see the amplitude of the peak decreases as the noise increases from R0 to R3. Fig. 5(c) plots ATS and RTS for R3. We can see that even for the most noisy data R3 in this data set, the peaks at 6 segments are still very significant in the ATS and RTS plot and can lead to the selection of the corresponding thresholds unambiguously. Table 2 gives a complete trace of iterative thresholding on R0 using the $SM1$ algorithm.

Iterative thresholding can also be used for shapes that have features of different scales. For these shapes, the ATS and RTS data will have multiple peaks. $P0$ is a

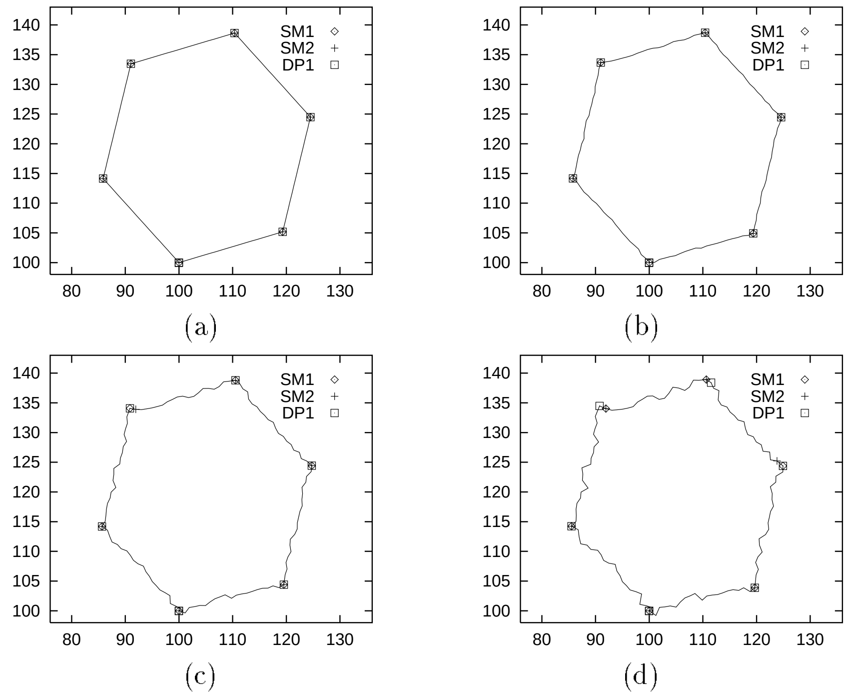


Figure 4: The segmentation results of different algorithms. The error threshold for SM1 is 5.5 and 80 for SM2.

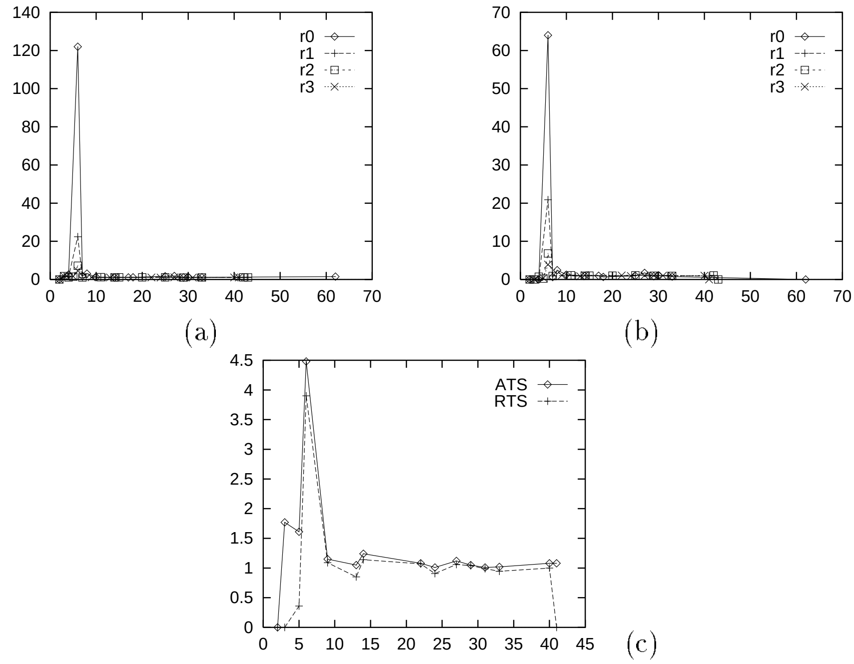


Figure 5: The iterative threshold technique. The peaks in the heuristic data correspond to good threshold candidates

i	N_{segs}	eT_i	E_i	ATS_i	RTS_i
1	2	1e+07	17.3	0	0
2	4	17.2	6.55	2.65	0
3	6	6.48	0.0537	122	64
4	7	0.0532	0.0282	1.9	0.624
5	8	0.0279	0.00925	3.05	2.43
6	10	0.00915	0.00735	1.26	1.24
7	14	0.00728	0.00724	1.01	0.941
8	17	0.00717	0.00672	1.08	0.959
9	18	0.00665	0.00598	1.12	0.666
10	25	0.00592	0.00354	1.69	0.931
11	27	0.0035	0.00195	1.81	1.73
12	29	0.00193	0.00186	1.05	0.84
13	30	0.00184	0.00149	1.25	1.09
14	33	0.00147	0.0013	1.14	0.763
15	62	0.00129	0.000869	1.5	0

Table 2: A trace of iterative thresholding on R0 using the $SM1$ algorithm.

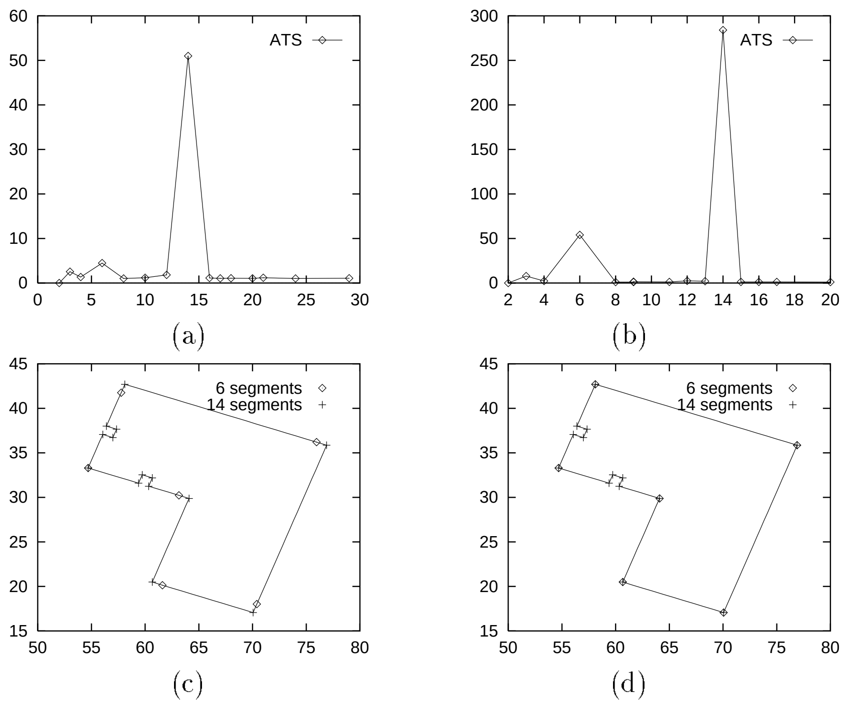


Figure 6: Iterative thresholding on a polygon that have two levels of structures. (a)-(b): ATS plots for $SM1$ and $SM2$. (c)-(d): segmentation results using the two peak thresholds and $SM1$ and $SM2$ respectively.

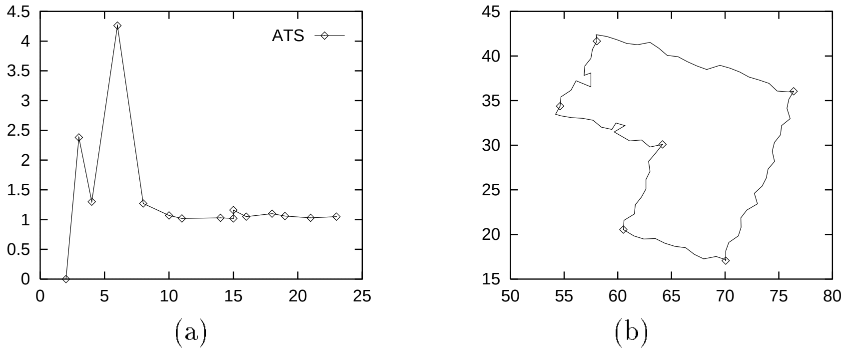


Figure 7: Iterative thresholding results for $P1$ using algorithm $SM1$. (a): ATS plots. (b): segmentation results.

polygon with sides of two different scales. Fig. 6 shows its ATS heuristics and segmentation results using segmentation algorithms $SM1$ and $SM2$. There are two large peaks in the ATS plots for both $SM1$ and $SM2$ that correspond to segmentations of 6 segments and 14 segments. These two peaks indicate that the shape under processing has features of two different scales.

Fig. 6(c) and (d) shows the segmentation results of the corresponding thresholds. $SM1$ and $SM2$ give same segmentation results for the second peak. However, it is interesting to see their segmentation results for the first peak (6 segments) differ and $SM2$ gives a better result. This is because $SM1$ uses a uniform error norm that counts only the biggest segment error so the algorithm need not optimize as much as possible on segments that do not have the largest segment error. This experiment demonstrates that for shapes with features of different scales, algorithms using summable error norms work better than algorithms using uniform error norms when segmenting these curves into large scale features.

Fig. 7 shows the use of iterative thresholding technique on P1. The underlying segmentation algorithm is $SM1$. We can see when the noise becomes large enough to inundate the features of small scales, the corresponding peaks disappears in the ATS plot. This indicates that ATS and RTS are general heuristics, they can not be used to distinguish fine details from noise. To do that, prior knowledge about the shape need to be incorporated, either in a more application specific heuristics or in a more specific segmentation algorithms or both.

In curve segmentation, the peaks in the ATS and RTS plots represent the place where the underlying approximation model of the segmentation algorithm can suddenly fit the curve very well to lead to a large approximation error decline. The thresholds at these points are certainly worth more consideration than other thresholds that have flat performance in reducing approximation error.

There are also some small peaks in the ATS and RTS plots when the number of segments is small, such as the peaks in Fig. 6 (a), (b) and Fig.7(a) where the number of segments is three. These peaks occur because when the number of segments is small, a small increase of the number of segments may lead to substantial approximation error reduction. One way to deal with this effect is to ignore peaks of small number of segments, but this requires prior knowledge about the possible number of segments a curve may have. Another way is to use heuristics that explicitly consider the effect of the small number of segments rather than ATS and RTS. In this paper we just ignore any peaks when the corresponding number of segments is less than four.

5 Conclusions

In this paper, we have identified the four main components in curve segmentation algorithms using edge approximation approach. We have abstracted the main components as independent software modules that lead to an architecture which supports different curve segmentation techniques and the generation of different specific curve segmentation algorithms by combining different techniques in different modules. We have implemented a curve segmentation library package using this architecture. The existence of this kind of library packages can help both academic research and engineering projects. For a specific system, a specific and appropriate segmentation algorithm can be quickly developed and tested using the library. For a general purpose system, a number of different algorithms can be instantiated from the library and used on different sub-tasks and different curve classes. From the point of view of research, this architecture allows researchers to focus on developing new segmentation techniques, without worrying about other implementation details and unnecessarily restricting their techniques to a small number of fixed algorithms.

We have also developed the iterative thresholding technique which can be used by a large class of algorithms to find appropriate thresholds without any specific knowledge of the input data. The peaks of the ATS and RTS heuristics indicate the existence of some inherent structures of the data and the corresponding thresholds can be used to explore these structures. We have applied the iterative thresholding technique in curve segmentation using the split and merge algorithm and the results we obtained are very encouraging.

Although the mathematical equations, algorithms and experiments are all given to deal with 2D curves, the techniques discussed in this paper are by no mean limited to 2D curves and can be extended to 3D curves effortlessly.

Many different segmentation techniques and algorithms have been developed by researchers. One problem is how to use different techniques more flexibly. Our independent module approach gives a very good solution to this problem. Another much harder problem is how to evaluate different algorithms and parameterize them under different situations. The number of algorithms that researchers have already developed might be sufficient for many hard applications. But our knowledge about when and why they work is insufficient. We believe this is the reason why the curve segmentation problem in a general purpose system is still far from been solved. Our work also contributes to this area. The decomposition of an algorithm into modules can simplify the analysis of the performance of algorithms and help to evaluate the appropriateness of an algorithm to some specific data. The iterative thresholding technique can be used to parameterize an algorithm according to the inherent properties of the input data.

References

[1] H. Asada and M. Brady. The curvature primal sketch. *IEEE Trans. on Pattern*

Analysis and Machine Intelligence, PAMI-8:2–14, January 1986.

[2] F. Attneave. Some informational aspects of visual perception. *Psychol. Rev.*, 61:183–193, 1954.

[3] O. Baruch and M.H. Loew. Segmentation of two-dimensional boundaries using the chain code. *Pattern Recognition*, 21:581–589, 1988.

[4] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Comm. ACM*, 4:284, 1961.

[5] J.M. Chen, J.A. Ventura, and C.H. Wu. Segmentation of planar curves intor circular arcs and line segments. *Image and Vision Computing*, 14:71–83, 1996.

[6] I.D. Faux. and M.J. Prett. In Chichester, editor, *Computational Geometry for Design and Manufacture*. England: Ellis Horwood, 1979.

[7] M.A. Fischler and R.C. Bolles. Perceptual organization and curve partitioning. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8:100–105, January 1986.

[8] M.A. Fischler and H.C. Wolf. Locatng perceptually salient points on planar curves. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:113–129, February 1994.

[9] H. Freeman. On the classification of line drawing data. In W.W. Dunn, editor, *Models for the Perception of Speech and Form*. Cambridge, 1967.

[10] B. Gluss. Further remarks on line segment curve-fitting using dynamic programming. *Comm. ACM*, 5:441–443, 1962.

[11] D.D. Hoffman and W.A.Richards. Representing smooth plane curves for recognition: Implications for figure-ground reversal. In *Proc. Nat. Conf. Artif. Intell.*, 1982.

[12] C. Ichoku, B. Deffontaines, and J. Chorowicz. Segmentation of digital plane curves: A dynamic focusing approach. *Pattern Recognition Letters*, 17:741–750, 1996.

[13] W. Kim and R.H. Park. Contour coding based on the decomposition of line segments. *Pattern Recognition Letter*, 11:191–196, 1990.

[14] D.G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31:355–395, 1987.

[15] J.W. McKee and J.K. Aggarwal. Computer recognition of partial views of curved objects. *IEEE Trans. Comput.*, C-26:790–800, 1977.

[16] G. Medioni and Y. Yasumoto. Corner detection and curve representation using cubic b-splines. *Computer Vision, Graphics, and Image Processing*, pages 298–303, 1987.

[17] F. Mokhtarian and A. Mackworth. Scale-based description and recognition of planar curve and two-dimentional shapes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-8:34–43, January 1986.

[18] A. Oliviero and G. Scapetta. A new approach to contour coding. *Comput. Vision Graphics, Image Processing*, 15:87–92, 1981.

[19] T. Pavlidis. Waveform segmentation through functional approximation. *IEEE Trans. on Computers*, C-22:689–697, 1973.

[20] T. Pavlidis. Curve fitting with conic splines. *ACM Trans. Graphics*, 2:1–31, 1983.

[21] T. Pavlidis and S.L. Horowitz. Segmentation of plane curve. *IEEE Trans. on Computers*, C-23:860–870, August 1974.

[22] W.A. Perkins. A model-based vision system for industrial parts. *IEEE Trans. Comput.*, C-27:126–143, 1978.

[23] A. Rattarangsi and R.T. Chin. Scale-based detection of corners of planar curves. In *Proc. 10th IEEE Int. Conf. on Pattern Recognition.*, pages 923–930, 1990.

[24] A. Rosenfeld and E. Johnston. Angle detection on digital curves. *IEEE Trans. Comput.*, C-22:875–878, 1973.

[25] P.L. Rosin and G.A.W. West. Segmenting curves into elliptic arcs and straight lines. In *Third Int Conf. Computer Vision.*, pages 75–78, 1990.

[26] P.L. Rosin and G.A.W. West. Curve segmentation and representation by superellipses. In *Australia and New Xealand Conf. Intelligent Information Systems.*, pages 530–534, 1993.

[27] P.L. Rosin and G.A.W. West. Nonparametric segmentation of curves into various representations. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17:1140–1153, December 1995.

[28] A. Sirjani and G.R. Gross. An algorithm for polygonal approximation of a digital curve. *Pattern Recognition Letter*, 7:299–303, 1988.

[29] J.A. Ventura and J.M. Chen. Segmentation of two-dimensional curve contours. *Pattern Recognition*, 25:1129–1140, 1992.

[30] K. Wall and P.E. danielson. A fast sequential method for polygonal approximation of digitized curves. *Comput. Vision Graphics, Image Processing*, 28:220–227, 1984.

[31] Wenhua Wan and Jose A. Ventura. Segmentation of planar curves into stright-line segments and elliptical arcs. *Graphical Models and Image Processing*, 59:484–494, November 1997.

[32] A. Witkin. Scale-space filtering. In *Proc. 7th Int. Joint. Conf. Artificial Intell.*, pages 1019–1021, 1983.

[33] Daniel M. Wuescher and Kim L. Boyer. Robust contour decomposition using a constant curvature criterion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13:41–51, January 1991.