Robust reconstruction of 2D curves from scattered noisy point data

Jun Wang, Zeyun Yu, Weizhong Zhang, Mingqiang Wei, Changbai Tan, Ning Dai, Xi Zhang

College of Mechanical Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, China
Department of Computer Science, University of Wisconsin-Milwaukee, Milwaukee, WI, USA
College of Information Engineering, Qingdao University, Qingdao, China
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China
School of Mechatronics Engineering and Automation, Shanghai University, Shanghai, China

ARTICLE INFO

Article history:
Received 15 June 2012
Accepted 13 January 2014

Keywords:
2D curve reconstruction
Noisy point data
Feature-preserving
Normal-based smoothing

ABSTRACT

In this paper, a robust algorithm is proposed for reconstructing 2D curve from unorganized point data with a high level of noise and outliers. By constructing the quadtree of the input point data, we extract the "grid-like" boundaries of the quadtree, and smooth the boundaries using a modified Laplacian method. The skeleton of the smoothed boundaries is computed and thereby the initial curve is generated by circular neighboring projection. Subsequently, a normal-based processing method is applied to the initial curve to smooth jagged features at low curvatures areas, and recover sharp features at high curvature areas. As a result, the curve is reconstructed accurately with small details and sharp features well preserved. A variety of experimental results demonstrate the effectiveness and robustness of our method.

1. Introduction

Curve reconstruction from 2D point data is a fundamental problem in geometric modeling, reverse engineering, computational geometry, computer graphics and image processing, computer vision. For instance, in reverse engineering, one of the effective methods to model point data for fabrication using rapid prototyping techniques is to adaptively slice the data points, along a specific direction, into a series of layers and the points in each layer are treated as planar. By reconstructing the planar curve of each layer, the final model can be created using sweep or loft modeling operations [1,2].

Generally, curve reconstruction can be defined to compute curves to approximate the boundary point data as closely as possible. Over the past two decades, a number of curve reconstruction algorithms have been proposed [3–11]. In spite of considerable advances, there are still some problems with those methods, especially when the input point data contain a high level of noise and outliers. Furthermore, if sharp features (e.g. corners) exist within the curve, the requirement of being resilient to noise is particularly challenging since noise and sharp features are ambiguous, and most current techniques tend to blur out those sharp features or even amplify noisy samples.

In this paper, we present an effective curve reconstruction algorithm, where the input point data consist of a set of unorganized points around curve boundaries, ridden by a high level of outliers and noise. Specifically, by constructing the quadtree of the input point data, we extract the "grid-like" boundaries of the quadtree, followed by applying a modified Laplacian method to smooth the boundaries. The skeleton is computed and thereby the initial curves are constructed by circular neighboring projection. The projection method may produce some jagged edges. Therefore, we exploit a normal-based processing method on the initial curves to smooth out bumpy features at low curvatures areas, and to recover sharp features at high curvature areas. In contrast to previous
works, our method is capable of handling a high level of noise and outliers. Small details of the original curve can be recovered satisfactorily; meanwhile, sharp features (e.g. corners) are also nicely preserved. The pipeline is demonstrated in Fig. 1.

2. Related works

In this section, we briefly review the most related works on curve reconstruction from unorganized point data, and examine whether they have the ability to handle noise and outliers, and preserve sharp features.

Fang et al. [12] presented a method based on spring energy minimization to approximate unorganized point data with a curve. The nonlinear minimization problem of spring energy is solved by successive quadratic programming; however, this solution needs a good initial guess and priors of point data topology. Taubin et al. [13] designed a planar curve reconstruction method from unorganized point data using an implicit simplicial curve, defined by a planar triangular mesh and the values at the vertices of the mesh. These two methods are difficult to handle in cases with noise and outliers.

Pottmann et al. [14] used a pixel-based method to thin input point data to a curve, where the thinning technique is exploited to cope with noise. After defining an appropriate grid on the plane, pixels including one or more points are filled with black creating a binary image. Then the medial axis of the binary image is computed by using the image thinning algorithm. Finally, a smooth curve is achieved via curve approximation. Goshtasby [15] presented a method to compute a radial basis function surface on a point cloud, followed by discretizing the surface into an image. By tracing the spine of the image, the curve is achieved from the point cloud. In these methods, small branches of the median axis are ignored so that the small features of curves are filtered out and the reconstructed curve is consequently inaccurate.

The moving least squares (MLS) technique [16] is a powerful and robust point-set modeling approach. The basic idea is to search the neighbors of each point of the input data, and fit them by a curve with a weighted regression. The point is then replaced by the projection point on the curve. The procedure is repeated until the point data are thin enough to achieve the curve reconstruction. Note that the reconstruction result is dependent on the size of the selected neighbors. Lee [17] proposed a variant of the MLS method to reconstruct curves from unorganized point data, in which the size of neighbors is chosen based on the idea of principal component analysis. With this method, the noise is handled to some extent, but the sharp features are hard to be retained.

Poon [18] proposed an algorithm to reconstruct polygonal closed curves from noisy samples drawn from a set of smooth closed curves, which consists of three steps: point estimation, pruning and output. In the point estimation step, the noise is filtered out and new points are computed. A pruning step is taken to decimate the new points so that the interpoint distances in the pruned subset are large compared with their distances from the curve. Then, the NN-crust algorithm [4] is run in the output step to obtain the final curve. Lin et al. [19] reconstructed curves from non-uniformly sampling data based on an interval B-spline curve. The sequence joining method is exploited to cluster the point cloud into a rectangle sequence, and then two boundary point sequences are computed using the quasicentric point sequence. By fitting two boundary point sequences, an interval B-spline curve is obtained enveloping the strip-based point cloud. As a result, the noise of the points is filtered out. Wu et al. [20] designed an automatic reconstruction method of polygonal curves from unorganized dense planar points. The planar points are sorted, followed by decomposing the sorted points into different levels using B-spline wavelets. Then the polygonal curve is constructed hierarchically from coarser to finer level. From their experimental results, all those methods have good performance on smooth curve reconstruction from point data with a certain level of noise; however, they are incapable of preserving sharp features within curves, especially when the point data are highly noisy.

de Goes et al. [21] proposed a practical algorithm to address the problem of reconstruction and simplification of 2D curves from unorganized point sets based on an optimal transport technique. This method is able to robustly deal with feature preservation, like sharp intersections and corners. It also shows satisfactory robustness to a certain level of noise and outliers. However, it is hard to cope with heavy noise and outliers. In addition, small details from original curves generally are simplified by this method, and consequently the reconstruction results are relatively inaccurate. Our method takes into account all the input points within boundaries,
making the results more faithful to the original curves. Furthermore, the normal-based feature-preserving method is capable of recovering sharp features simultaneously.

3. Quadtree boundary construction

3.1. Boundary extraction

3.1.1. Quadtree subdivision

Given a set of unorganized points \( P \), we first generate a point-based quadtree. The quadtree is an adaptation of a binary tree used to represent 2D point data [22]. There are many criteria to subdivide the quadtree for different applications. Typically, the quadtree is decomposed recursively until there is only one point in each leaf node. Accordingly, the depth of adjacent nodes may be different. For our application, we enforce the condition that all adjacent leaf nodes have the same depth, referred to as uniform quadtree subdivision. The depth is pre-specified in our implementation. Meanwhile, we also have the clue to set the depth. Specifically, we construct the \( k\)-d (i.e. \( k = 1 \)) tree for the input point data and compute the average distance \( d_{avg} \) of all two closest points from the input data; then we set depth = \( \frac{d_{avg}}{\kappa} \), where \( \kappa \) is the diagonal of the bounding box of the input data, and \( \kappa \) is a positive constant (e.g. 0.25).

3.1.2. Quadtree segmentation

If there is more than one curve approximated by the set of points \( P \), the quadtree will consist of more than one connected components accordingly. Then, we adopt the region growing technique to partition the quadtree into the components with connected points. Given \( P \), let \( T = T_{valid} \cup T_{void} \) be the set of the nodes of the quadtree, where \( T_{valid} \) is the set of the nodes each of which contains at least one point (valid nodes), and \( T_{void} \) the set of the nodes without any points (void nodes). Then, segmentation of the quadtree is referred to as the partition of \( T_{valid} \) into \( k \)-disjoint connected components, i.e.:

\[
\bigcup_{i=1}^{k} C_i = T_{valid}, \quad C_i \subseteq T_{valid}.
\]

\[
C_i \cap C_j = \emptyset, \quad i, j = 1, \ldots, k, \quad i \neq j.
\]

Given a valid node, it is regarded as a boundary node if at least one of its 4-neighbors is void. Based on this criterion, we exploit the region growing approach to partition the quadtree. Specifically, starting from a seed (an unsegmented, valid node), the region grows by adding its connected nodes which are valid and unsegmented. The added nodes are set segmented. The seed keeps updating and the growing process is thereby repeated until no more node can be added into the region. We refer to the maximal connected region as a segment. In particular, the connected node of a seed is defined as the one which shares a common edge or corner with the seed. Applying this procedure to the quadtree, we obtain \( k \)-disjoint connected segments. If the input point data contain outliers, it is likely that small segments are generated encapsulating those outliers. The one consisting of a small number of nodes is treated as an outlier–ridden segment. As a result, the outliers can be detected and deleted prior to further processing.

3.1.3. Boundary propagation

With a series of segments, we extract their boundaries via the propagation technique, which proceeds as follows. For each segment, we randomly search one boundary edge, where one of its incident nodes belongs to this segment and the other does not. Let \( e = (\text{pre}_V, \text{cur}_V) \) be the first boundary edge, where \( \text{pre}_V \) and \( \text{cur}_V \) are the starting and ending vertices of \( e \), respectively. We need to find the next boundary edge \( e' = (\text{cur}_V, \text{nxt}_V) \) connected with \( e \). Namely, the goal is to find the ending vertex \( \text{nxt}_V \) of \( e' \), since \( \text{cur}_V \)

must be the starting vertex of \( e' \) (see Fig. 2). For the vertex \( \text{cur}_V \), let \( \text{node}_1, \text{node}_2, \text{node}_3, \text{node}_4 \) be the upper-right, upper-left, lower-left, and lower-right incident nodes of \( \text{cur}_V \), then the next propagation vertex \( \text{nxt}_V \) is determined by:

\[
\text{nxt}_V = (\text{cur}_V, \text{nxt}_V) = \begin{cases} 
\text{cur}_V, & \text{if } \text{S(node}_1,1) = \text{S(node}_2,1) \\
\text{cur}_V, & \text{if } \text{S(node}_2,1) \neq \text{S(node}_3,1) \\
\text{cur}_V, & \text{else if } \text{S(node}_3,1) \neq \text{S(node}_4,1) \\
\text{cur}_V, & \text{else if } \text{S(node}_4,1) \neq \text{S(node}_1,1) \\
\text{cur}_V, & \text{else} 
\end{cases}
\]

(2)

where \( \text{S(node)} \) is the segment number of node. After finding the vertex \( \text{nxt}_V \), the corresponding boundary edge \( e' = (\text{cur}_V, \text{nxt}_V) \) of \( e \) is hence obtained. The propagation procedure is repeated by continuously adding new connected edge until it reaches the first edge, i.e. \( e \), or no more edge can be added. By this means, the boundaries of all segments can be extracted.

3.2. Boundary smoothing

The extracted boundary is “grid-like”, as the segment is represented with quadrilateral nodes. We exploit a weighted Laplacian smoothing method to smooth the boundary. Let \( V = \{ v_0, v_1, \ldots, v_N \} \) be the vertex set of a boundary (\( n \) is the number of vertices on the boundary). For each vertex \( v_i \), the 2-ring neighboring vertices are searched, i.e. \( \text{N}(v_i) = \{ v_{i-2}, v_{i-1}, v_i, v_{i+1}, v_{i+2} \} \). Let \( \text{N}(v_i) \) be the set of vertices contributing to the position of \( v_i \), then \( v_i \) is updated as:

\[
v'_i = v_i + \sum_{v_j \in \text{N}(v_i)} W(\| v_i - v_j \|) \sum_{v_j \in \text{N}(v_i)} W(\| v_i - v_j \|)
\]

(3)

where \( v'_i \) is the new position of \( v_i \), and \( W \) is a standard Gaussian filter in terms of the distance between \( v_i \) and \( v_j \). Applying this filter, all “grid-like” boundaries are nicely smoothed. Fig. 3 illustrates the boundary extraction from 2D point data.

4. Curve reconstruction

In this section, we construct the initial curve based on the Voronoi diagram of the smooth boundaries determined from Section 3.

4.1. Skeleton extraction

We consider each boundary of a segment as a polygon. Given the vertices for a polygon, we compute the Voronoi diagram using Fortune’s fast algorithm [23]. This Voronoi diagram generation algorithm maintains both a sweep line and a beach line, which both move through the plane as the algorithm proceeds. The sweep line is a straight line, which we may assume to be vertical and moving left to right across the plane. At any time during the algorithm, the input points to the left of the sweep line will have been incorporated into the Voronoi diagram, while the points to the right of the sweep line will not have been considered yet. The beach line is not a line, but a complex curve to the left of the sweep line, composed of pieces of parabolas; it divides the portion of the plane within which the Voronoi diagram can be known, regardless of what other points might be to the right of the sweep line, from the rest of the plane. For each point to the left of the sweep line, we can define a parabola of points equidistant from that point and from the sweep line; the beach line is the boundary of the union of these parabolas. As the sweep line progresses, the vertices of the beach line, at which two parabolas cross, trace out the edges of the Voronoi diagram.

By this algorithm, the Voronoi diagram of each boundary of each segment is computed. If the underlying curve in a segment is simply open without any self-intersections, there is only one closed
Fig. 2. Illustration of boundary edge propagation. The next edge of \( e \) is (a) \( e' = (\text{curV}, \text{curV}(\cdot, 0)) \) when \( \text{node}_{(\cdot, 0)} \) and \( \text{node}_{(\cdot, -1)} \) are in different segments; or (b) \( e' = (\text{curV}, \text{curV}(\cdot, +1)) \) when \( \text{node}_{(\cdot, -1)} \) and \( \text{node}_{(0, \cdot)} \) are in different segments; or (c) \( e' = (\text{curV}, \text{curV}(\cdot, +1)) \) when \( \text{node}_{(0, \cdot)} \) and \( \text{node}_{(\cdot, 0)} \) are in different segments.

Fig. 3. Illustration of boundary extraction of 2D point data. (a) 2D point data; (b) Quadtree subdivision; (c) Quadtree grid segmentation, where there are two segments; (d) The grid-like boundaries of point data; (e) The smooth boundaries of point data. The big segment is “closed-loop”, bounded by two “red” boundaries, while the small one is “open”, bounded by a single “blue” boundary. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 4. Skeleton extraction of the “C” point data. (a) The input point data; (b) The smoothed boundary of the quadtree; (c) The outer boundary; (d) The Voronoi diagram of the boundary polygon; (e) The skeleton.

Fig. 5. Skeleton extraction of the “bunny” point data. (a) The input point data; (b) The smoothed boundaries of the quadtree; (c) The inner and outer boundaries; (d) The Voronoi diagram of the two boundary polygons; (e) The extracted skeleton.

boundary (see Fig. 4); otherwise, there are more than one boundary for the segment (see Fig. 5). The latter case occurs frequently, where we observe that there is only one outer boundary, and all others are inner boundaries. The skeleton of the segment only consists of the edges which are inside the outer boundary, and outside all inner boundaries. Therefore, we need to detect all those Voronoi edges whose endpoints locate inside the inner boundaries and delete them. The remaining Voronoi edges form the skeleton of the segment. Figs. 4, 5 show skeleton extraction of the “C” and “bunny” point data.

4.2. Short branch pruning

As can be seen in Fig. 5(e), the vertices of the smoothed boundaries give rise to a certain number of short edges (branches) which are commonly treated as artifacts that do not contribute to the overall salient features of a curve. To remove these short branches, we exploit a tree-based pruning method, which is comparatively more robust than other related pruning approaches [24,25]. The tree structure of the Voronoi edges is first constructed. We search all nodes (i.e. Voronoi vertices) which have more than two incident edges, referred to as dubious nodes. For each dubious node, we consider it as a new root and traverse the tree using the Depth-First Search strategy so that the lengths of all paths are obtained from the root to leaf nodes. If all paths from the dubious node are longer than a pre-defined threshold \( \xi \), nothing is pruned here. If two paths are longer than \( \xi \), then we prune those branches in the paths whose lengths are shorter than \( \xi \). If only one path is longer than \( \xi \), the longest one among the remaining paths is kept so that we prune the branches in all other paths. Fig. 6 illustrates the short branch pruning cases. Fig. 7 presents the pruning result of the skeleton from Fig. 5.

4.3. Skeleton partition

After pruning short branches, the remaining Voronoi edges convey the basic shape information of a curve. To facilitate post-processing, the whole skeleton needs to be partitioned into simple
segments. The graph structure is constructed based on the connectivity of Voronoi edges. Taking the graph as input, we search all terminal nodes of the graph each of which has only one incident edge, and find the longest path between each two terminal nodes. Thus, we are able to get the overall longest path among all terminal nodes and subtract all edges of the path from the graph. The connected edges of the path form a segment of the skeleton. Iteratively, we update the graph and extract the longest path until the graph is empty. As a result, the skeleton is decomposed into different segments. In particular, if the skeleton is closed and hence there is no terminal node in the corresponding graph, we temporarily remove an edge from the graph to generate two terminal codes so as to run the above procedure to carry out skeleton partition. Certainly, the removed edge needs to be added back to the segment ultimately. Fig. 8 illustrates the skeleton partition with the open and closed cases. Note that the whole partition process simultaneously builds the topological connectivity relationship among the Voronoi edges, which lays a foundation for the following processing.

4.4. Circular neighboring projection

From the generation of skeleton, a few noisy points away from the underlying curve can lead to an inaccurate result (see Fig. 9). In Fig. 9(a), b1 and b2 are the smooth boundaries of the quadtree of the input points; skl is the skeleton extracted from b1 and b2, and v is a vertex on skl. We can see that several sparse points below skl play an equally important role in determining the shape of skl as those dense points above skl do. In reality, those sparse points are quite likely to be noise. Therefore, the real distribution of the input point data need to be taken into account as well during reconstruction. Accordingly, we propose a circular neighboring projection algorithm to reconstruct the curve based on the segments of the skeleton, which is illustrated by Fig. 9. We can see that the curve cvr is more faithful to the original input data than skl.

Given a vertex vi of a skeleton segment, let cir(vi) = (vi, ri) be the maximum inscribed circle of vi in terms of the smoothed boundaries of the quadtree, then we may obtain the circular neighbors of vi as:

\[ \text{CirNgbr}(v_i) = \{ v | \|v - v_i\| < r_i, \ v \in \mathcal{P} \} \]

where \( \mathcal{P} \) is the original input point set. We project all points in CirNgbr(vi) onto the bisector line of two incident edges of vi, determined by the unit vector ni and vi. Then, the new position of vi is set as the centroid of those projection points, i.e.:

\[ v'_i = v_i + \frac{1}{|\text{CirNgbr}(v_i)|} \sum_{v \in \text{CirNgbr}(v_i)} \left( (v - v_i) \cdot n_i \right) \]

Fig. 10 gives the circular neighboring projection for two types of points, where one has the branch, the other does not. Apply this projection strategy to all points of the skeleton, the initial curve of the input point data is constructed.

5. Feature-preserving curve smoothing

The initial curve perhaps contains some jagged edges, meanwhile, some sharp features (e.g. corners) may get blurred. To address this problem, we adopt a normal-based processing method to smooth jagged edges and recover sharp features. The basic idea is to modify a curve by adjusting its vertices such that the curve is fit to a field of smoothed normals. Therefore, we need to obtain the smoothed normals first. Basically, if an edge of a curve does not contain vertices located at a sharp corner, then the edge normal tends toward the mean of its neighboring edge normals; otherwise,
Fig. 8. Partition of open and closed skeletons. (a), (b) are open skeletons, while (c) and (d) are closed. For the closed one, one of edge is temporarily removed and the remaining skeleton turns to be open. Thus the longest path between two terminal nodes can be found. Meanwhile, the topological connectivity relationship is constructed.

Fig. 9. Illustration of circular neighboring projection. (a) The input point data and the extracted skeleton skl; (b) A vertex v on skl and its circular neighboring points; (c) Projecting all neighboring points to the bisector line l at v; (d) The centroid u of all projection points on l; (e) u: the updated vertex of v. All vertices on skl are adjusted in this way so that the new curve crv is generated.

Fig. 10. Circular neighboring projection of two types of points with or without associated branches. (a) The original point data with a "rectangle" shape; (b) The boundaries and the median axis, where the "yellow" curve is the main skeleton and the "teal" are branches; (c) The zoom-in view of neighbors of point A with the associated branch: A−p1−p2−p1−p0, where the corresponding maximal inscribed circles of A, p1, p2, p1, p0 are presented and all points inside those circles are given. All those points are projected onto the normal line nA and the average of projection points are obtained as A′ in (d). (e) The zoom-in view of neighbors of point B without a branch, where a single maximal inscribed circle of B and all points inside the circle are presented. By averaging the projection points, the new position B′ of B is given in (f). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

it tends toward the closest normal of neighboring edges. Accordingly, we propose a novel normal smoothing approach based on the bilateral filtering technique.

The bilateral filter was originally conducted in image processing [26]. It is a nonlinear filter derived from Gaussian blur with a feature preservation term that decreases the weights of pixel as a
function of intensity difference. The bilateral filtering for an image $I(u)$, at coordinate $u = (x, y)$, is defined as:

$$
\hat{I}(u) = \frac{\sum_{p \in N(u)} W_c(\|p - u\|) W_f(I(p) - I(u))}{\sum_{p \in N(u)} W_c(\|p - u\|) W_f(I(p) - I(u))}
$$

where $N(u)$, the neighborhood of $u$, is defined with $\{q_k : \|b - q_k\| < \rho = [2\sigma_c]\}$. The spatial smoothing function $W_c$ is a standard Gaussian filter in terms of the distance between $p$ and $u$, and the influence function $W_f$ is a standard Gaussian filter defined on the intensity difference between $p$ and $u$. Accordingly, the intensity value on $u$ is determined mainly by the neighboring pixels that are close in terms of the distance and the intensity. As a result, the large intensity differences, which are considered as image features, are penalized by the influence function $W_f$, thus preserving image features. There are also a number of variants of bilateral filtering [27–29]. Miropsolsky et al. [30] analogized the normal vector to the intensity value in the bilateral filtering formula (6), referred to as geometric bilateral filtering. They applied this geometric bilateral filtering method for data reduction and noise removal on scanned points during mesh reconstruction.

Normal smoothing has recently been adopted for mesh smoothing in geometry processing applications [31–33]. We introduce a new normal smoothing approach based on the bilateral filtering technique. Given an arbitrary edge $e_i$ of the initial curve with an unit normal $n_i$, and a middle point $e_i$ of $e_i$, the smoothed normal $\hat{n}_i$ of $e_i$ is represented as:

$$
\hat{n}_i = \frac{\sum_{j \in N(i)} W_c(\|c_j - c_i\|) W_s(n_j, n_i) n_j}{\sum_{j \in N(i)} W_c(\|c_j - c_i\|) W_s(n_j, n_i)}
$$

where $N(i) = \{ j : |e_i \cap e_j | \neq 0 \}$ is the connected edge set of $e_i$, $n_i$ is the unit normal of the connected edge $e_j$, and $c_i$ is the middle point of $e_i$. $W_c$ is a standard Gaussian filter, and $W_s(n_j, n_i)$ is defined as:

$$
W_s(n_j, n_i) = \begin{cases} 0, & \text{if } (n_i - n_j) \cdot n_i \geq \hat{\mu} \\ \left( (n_i - n_j) \cdot n_i - \hat{\mu} \right)^2, & \text{otherwise} \end{cases}
$$

where $\hat{\mu} = \frac{\sum_{j \in N(i)} \|n_i - n_j\|^2}{\|N(i)\|}$ and $\|N(i)\|$ is the number of elements of $N(i)$. Essentially, the normal vectors are truncated if the differences between them and $n_i$ are greater than the average normal vector difference $\hat{\mu}$. Therefore, the filter ignores the heavy noise and is less sensitive to a high level of noise.

After smoothing the edge normals, the curve is modified by updating the vertices based on these new normals. We first introduce an error function indicates how good the curve fits the field of smoothed normals. For each vertex $v$ of the initial curve $C$, let $e_0 = (v, v_0), e_1 = (v, v_1)$ be the induced edges of $v$, and $n_0, n_1$ be the original normals of $e_0, e_1$. $n'_0, n'_1$ be the smoothed normals of $e_0, e_1$, then the error function of $v$ is defined as:

$$
\text{Error}(v) = \frac{1}{2} \sum_{j \in \{0, 1\}} \left( \|v_j - v\| \right) \left( n'_j \cdot (v_j - v) \right)^2.
$$

Accordingly, the error function of the initial curve $C$ can be expressed by:

$$
\text{Error}(C) = \sum_{v \in C} \text{Error}(v).
$$

The new curve $C'$ is achieved by solving the minimization problem:

$$
C' = \arg\min_{C} \text{Error}(C).
$$

The new curve $C'$ of $C$ is:

$$
C' = \arg\min_{C} \text{Error}(C).
$$

6. Results and discussions

All algorithms described have been implemented and run on a PC with 1.8 GHz CPU and 2 GB RAM. We have tested our algorithm on a variety of 2D scattered point data with either raw or synthetic noise, outlier for analysis of the effectiveness of our method. The synthetic noise is made by a zero-mean Gaussian function with standard deviation proportional to the diagonal length of the bounding box of the input point data. The synthetic outliers are generated randomly in the bounding box of the input point data.

6.1. Parameters

In our algorithm, there are a few parameters: (1) quadtree subdivision depth $\delta$; (2) outlier removal threshold $\tau$; (3) branch length threshold $\xi$; and (4) the iteration number of normal smoothing $n$. Among the parameter, the quadtree subdivision depth is generally on the density of the input point data. If the point data are dense, the depth could be assigned with a relatively big value; otherwise, it should be set with a small value. The selection of outlier removal threshold is based on the level of outlier. If there is a high level of outlier, the threshold could be big; otherwise, it should be set relatively small. The branch length is related to the shape complexity of the input point data. If there are many small branches in the original geometry, the branch length threshold should be smaller; otherwise, it could be a big value. The iteration number of normal smoothing is set according to the level of noise. If there is a high level of noises, it should be given with a relatively big value; otherwise, it should be a small value. Based on a number of experiments, we have the following typical settings in our implementation: $\delta \in \{50–150\}$; $\tau = \omega \ast d$, $\omega \in [0–0.25]$; $\xi = \omega \ast d$, $\omega \in [0–0.1]$ and $n \in [5–15]$.

6.2. Comparisons with previous methods

To demonstrate the effectiveness of our method, we compare it with two related methods, including Lee’s [17] and de Goes et al.’s [21] methods.

Synthetic data: First, we test our approach on the point datasets with synthetic noise, see Figs. 12–17. From the results, Lee’s [17] is fairly sensitive to noise, resulting in incorrect geometries. The basic shapes from de Goes et al.’s [21] method are basically correct, while many details are blurred out. Our method always yields better results, where sharp features and fine details are well preserved.

The comparison results have visually shown the superiority of our approach to other methods in terms of recovering sharp features and preserving fine details. Furthermore, we provide the
Fig. 11. Feature-preserving curve smoothing results of the rectangle point data. (a) The initial curve from circular neighboring projection; (b) The smoothing result from our method; (c) The initial curve and the final curve from our method; (d) The smoothing result from Miropolsky et al.’s geometric bilateral filter method.

Fig. 12. Curve reconstruction on the apple point data. (a) Noise-free point data; (b) Noisy point data (noise: 2% of a bounding box); Reconstruction results from (c) Lee [17], (d) de Goes et al. [21], and (e) ours. de Goes et al.’s and our methods yield better results than Lee’s method. de Goes’s method produces a simplified shape, while our method keeps more details.

Fig. 13. Curve reconstruction on the butterfly point data. (a) Noise-free point data; (b) Noisy point data (noise: 2.5% of a bounding box); Reconstruction results from (c) Lee [17], (d) de Goes et al. [21], and (e) ours.

Fig. 14. Curve reconstruction on the crab point data. (a) Noise-free point data; (b) Noisy point data (noise: 2% of a bounding box); Reconstruction results from (c) Lee [17], (d) de Goes et al. [21], and (e) ours. The shape of Lee’s result is distorted. The “wing” part of de Goes et al.’s result is not correct. Our result is relatively better.

Fig. 15. Curve reconstruction on the dolphin point data. (a) Noise-free point data; (b) Noisy point data (noise: 2% of a bounding box); Reconstruction results from (c) Lee [17], (d) de Goes et al. [21], and (e) ours.

Fig. 16. Curve reconstruction on the face point data. (a) Noise-free point data; (b) Noisy point data (noise: 2% of a bounding box); Reconstruction results from (c) Lee [17], (d) de Goes et al. [21], and (e) ours.
Fig. 17. Curve reconstruction on the monkey point data. (a) Noise-free point data; (b) Noisy point data (noise: 2% of a bounding box); Reconstruction results from (c) Lee [17], (d) de Goes et al. [21], and (e) ours. Our method generates much smoother result, compared with other two methods.

Fig. 18. The histograms show the Hausdorff distances between the original point data and the curves reconstructed from Lee’s [17], de Goes et al.’s [21] and our methods. All original point data are scaled into a unit bounding box (i.e., 1 × 1 × 1). From the histogram, the reconstruction errors from Lee [17] are much bigger than de Goes et al.’s [21] and our results. Notice that de Goes et al.’s [21] errors are still about 10 times bigger than ours.

Fig. 19. Curve reconstruction on the angel point data. (a) The input point data; and the reconstruction results from (b) Lee [17], (c) de Goes et al. [21] and (d) ours. Note that the density of the original point data is non-uniform. The contour is reconstructed successfully by our method, where the sharp corner (e.g. the toe part) and the details (e.g. the highlighted parts) are well recovered.

quantitative comparisons between our method and others. Given the original noise-free point data, we can compute the Hausdorff distance between the original point data and the reconstructed curve, which is used widely for measuring the reconstruction accuracy [34]. Specifically, the reconstructed curve is sampled uniformly into a series of points. Let \( \mathcal{P} \) and \( \mathcal{C} \) be the original point set and the sampling point set of the reconstructed curve, the Hausdorff distance \( d_H(\mathcal{P}, \mathcal{C}) \) is defined by:

\[
d_H(\mathcal{P}, \mathcal{C}) = \max \left\{ \sup_{p \in \mathcal{P}} \inf_{q \in \mathcal{C}} d(p, q), \sup_{q \in \mathcal{C}} \inf_{p \in \mathcal{P}} d(p, q) \right\}
\]

where \( \sup \) represents the supremum and \( \inf \) the infimum, and \( d(p, q) \) is the distance between \( p \) and \( q \). Fig. 18 shows the detailed comparison of the Hausdorff distance results from Figs. 12–17.

Raw data. We also compare our method with others on raw point data. The testing point data result from slicing 3D real point datasets with two close parallel planes, see Figs. 19–21. Lee’s [17] method does not achieve as good results as de Goes et al.’s [21] or our method. de Goes et al.’s [21] approach is able to preserve sharp features to some extent; however, it tends to smooth out fine details of the point data, as illustrated by the highlighted parts in Figs. 19–21.

Non-uniformly sampled raw data. The testing data above are uniformly sampled. To further verify the robustness of our algorithm, some non-uniformly sampled raw data are tested with de Goes et al.’s and our methods. The photogrammetry technique has been widely used to generate 3D point data from a series of photographic images. However, the generated point data could be
Fig. 20. Curve reconstruction on the **buste** point data. (a) Input point data; and the reconstruction results from (b) Lee [17], (c) de Goes et al. [21], and (d) ours. The shape from Lee’s method is completely deformed, and many details disappear in de Goes et al.’s result.

Fig. 21. Curve reconstruction on the **lion** point data. (a) Input point data; Reconstruction results from (b) Lee [17], (c) de Goes et al. [21], and (d) ours.

Fig. 22. Curve reconstruction on the **building** section data. (a) The building image; (b) The point data of the building and the section plane; (c) The 2D section point data; and the curves reconstructed from (d) de Goes et al. [21] and (e) ours; (f) The sub-sampled point data (1:10) of (c); and the curves reconstructed from (g) de Goes et al. [21] and (h) ours. The data are noisy and sampled non-uniformly. All curve segments are well reconstructed with our method, while some segments are missing in de Goes et al.’s [21] results.

fairly noisy and distributed non-uniformly, as shown in Fig. 22(b). Fig. 22(a) shows one part of a building image and Fig. 22(b) gives the corresponding 3D point data of the building generated by the photogrammetry technique. We intersect the point data with a section plane in Fig. 22(b) to generate 2D section point data in Fig. 22(c). Meanwhile, we sub-sample the 2D point data with a ratio of 1:10, resulting in the sparse data in Fig. 22(f). de Goes et al.’s [21] and our methods are run on those two point data. From the results, both methods are able to generate basic curves in the presence of the noisy and non-uniform data. de Goes et al.’s approach is prone to bridging gaps upon missing data, while our method is faithful to the original data. In addition, our method retains details (e.g. short curve segments) better than de Goes et al. [21] does.

Fig. 23 shows the reconstruction results of 3D scanning point data of a **screw nut**. Due to scanning constraints, the scanned point data could be incomplete. As shown in Fig. 23(b), one of the hexagon side faces is scanned insufficiently so that the point data are quite sparse and non-uniform. We project the side face point data onto the base plane to get the 2D point data in Fig. 23(c), followed by running Lee’s [17], de Goes et al.’s [21] and our methods to reconstruct curves. From the results, Lee’s [17] method can hardly handle this kind of point data. In terms of sharp feature
preserving, both de Goes et al.'s [21] and our methods acquire good results (see the sharp corners of the hexagon). For the dense circle data, de Goes et al.'s [21] approach is apt to over-sharpen the curve, as shown in Fig. 23(e). The third row shows the sub-sampling data (1:10) of (c), and the corresponding reconstruction results. We can see that de Goes et al.'s [21] method fails to recover the edge with sparse data, which is partially reconstructed by our method. In addition, since the point data of the circle and the hexagon are fairly close, de Goes et al.'s [21] approach connects them together mistakenly, whereas ours is still able to separate them.

**Point data with heavy noise and outliers.** Our method is also resilient to a high level of noise and outliers, as illustrated in Figs. 24 and 26. Fig. 24 presents the reconstruction results of the point data with different levels of noise and density from de Goes et al.'s [21] and our methods. From the results, de Goes et al.'s [21] approach does not perform well against the heavily noisy data, whereas our method still obtains pleasing results, as illustrated in Fig. 24(a)–(d). The point data are too sparse in Fig. 24(e). Consequently, neither de Goes et al.'s [21] method nor ours gains a good result.

Fig. 25 demonstrates our algorithm is capable of dealing with different levels of noise. From the comparison results, even the noise level increases up to 15%, the result is still acceptable. When the level increases to 25%, the reconstruction fails. Fig. 26 shows the performance of our method running against extremely high levels of outliers, where the number of outliers is even greater than that of the initial point data. From the results, our method is robust to noise and outlier. Even though the number of outlier points is more than that of the input point data, our method still achieves high quality reconstruction, as shown in Fig. 26(c). When the level of noise and outliers is extremely high, we need to increase the depth of quadtree subdivision to try to filter out the noise and outliers, which, however, results in a number of disconnect segments, as illustrated in Fig. 26(d). In this situation, our method can hardly to process. In addition, our method is also versatile to a variety of point data, including Chinese characters, sketch drawings and even noisy images, as shown in Fig. 27. Timings for typical examples are given in Table 1, where parameters are denoted by \(d, r, \xi, n\).

**Limitation.** As shown in Fig. 24(e), our method may fail to achieve desired results if the original point data are too sparse. In this situation, we need to decrease the depth of quadtree so as to avoid generating too many disconnected segments. Nevertheless, low depths usually lead to big reconstruction errors. Fig. 28 presents the reconstruction results of the point data with high sparsity. When the data become too sparse, we notice the results from our method have big errors. Therefore, our method currently has the limitation to handle extremely sparse point data, which could be studied in our future work.

### 7. Conclusions

We have presented a novel and robust reconstruction algorithm for 2D curves with sharp features which takes as input the unorganized scattered point data with noise and outliers. The median axis is extracted from the boundaries of the input data, where the main skeleton, as well as small branches, are obtained respectively. Consequently, the main features, as well as small features, are reconstructed via circular neighboring projection. The normal-based processing method is exploited to smooth jagged features and recover sharp features.

The main value of our approach is the ability to robustly deal with feature preservation, such as sharp corners. In addition, our method is capable of handling the point data with a high level of noise. Even though the noisy point data vary in terms of width, namely with non-uniformly sampling, our algorithm still has good performance.

The quadtree is used in our method, and the subdivision depth needs to be set, which essentially determines the size of the node grid. Since the density of the input point data varies a lot, it is not easy to automatically determine the size of the grid. If the size is too large, the skeleton may not represent the best approximation curve of the point data, while if it is too small, it is possible for the point data to be partitioned into several different components. In our implementation, we scale the point data into an unit bounding rectangle. The uniform scaling, together with the circular neighboring projection, alleviates this problem to some degree.

---

**Fig. 23.** Curve reconstruction on the screw nut section data. (a) The screw nut image; (b) The scanned point data of the screw nut; (c) The 2D section point data; and the curves reconstructed from (d) Lee [17]; (e) de Goes et al. [21] and (f) ours. The second row shows the sub-sampled data (1:10) of (c), and the corresponding curves reconstructed from Lee [17], de Goes et al. [21] and ours. Both de Goes et al.'s [21] and our methods nicely preserve the sharp corners of the hexagon. However, for the smooth circle, de Goes et al.'s [21] sharpens the shape excessively. The data on one edge are fairly sparse (pointed by the arrow in (c)). Ours can still reconstruct the edge from the sparse data successfully, while de Goes et al.'s [21] method fails.

| Table 1 |

<table>
<thead>
<tr>
<th>Models</th>
<th>Number of points</th>
<th>Parameters</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Butterfly</td>
<td>4290</td>
<td>(80, 6, 5, 5)</td>
<td>1.628</td>
</tr>
<tr>
<td>Crab</td>
<td>5 680</td>
<td>(75, 6, 5, 5)</td>
<td>1.759</td>
</tr>
<tr>
<td>Building</td>
<td>5 806</td>
<td>(60, 5, 4, 8)</td>
<td>1.824</td>
</tr>
<tr>
<td>Monkey</td>
<td>6 510</td>
<td>(100, 8, 5, 5)</td>
<td>3.487</td>
</tr>
<tr>
<td>Rectangle</td>
<td>8 306</td>
<td>(100, 7.5, 5, 15)</td>
<td>1.314</td>
</tr>
<tr>
<td>Face</td>
<td>11 430</td>
<td>(120, 8, 5, 12)</td>
<td>4.740</td>
</tr>
<tr>
<td>Table clothing</td>
<td>14 758</td>
<td>(250, 6, 5, 10)</td>
<td>6.979</td>
</tr>
<tr>
<td>Apple</td>
<td>17 000</td>
<td>(100, 8, 5, 5)</td>
<td>2.991</td>
</tr>
<tr>
<td>Angel</td>
<td>19 505</td>
<td>(150, 10, 5, 6)</td>
<td>6.204</td>
</tr>
<tr>
<td>Buste</td>
<td>20 383</td>
<td>(120, 8, 5, 6)</td>
<td>4.192</td>
</tr>
<tr>
<td>Dragon</td>
<td>25 554</td>
<td>(150, 10, 5, 10)</td>
<td>7.436</td>
</tr>
<tr>
<td>Lion</td>
<td>28 707</td>
<td>(100, 8, 5, 5)</td>
<td>4.487</td>
</tr>
<tr>
<td>Screw nut</td>
<td>64 091</td>
<td>(250, 10, 5, 12)</td>
<td>10.958</td>
</tr>
</tbody>
</table>

---

\[\text{J.Wang et al.} / \text{Computer-Aided Design 50 (2014) 27–40}\]
Fig. 24. Reconstruction results of the table clothing point data with noise. (a) The original noisy data (14758 points); and the sub-sampled point data with the ratio of (b) 1:2 (7379 points); (c) 1:5 (2952 points); (d) 1:10 (1474 points). The second, third rows show the curves reconstructed from de Goes et al.'s [21] method and ours, respectively.

Fig. 25. Robust reconstruction from the point data with different levels of noise. N% is the percentage of noise. The first row gives point data with different levels of noise; the second row shows the corresponding results with our method, where the “green” curves are the reconstruction results from noisy data, and the “red” curves are the reconstruction results from the data without noise. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 26. Robust reconstruction from the point data with heavy noise and outliers. (a) The input noise-free point data (6210 points); (b) The point data with noise (added noise: 2% of bounding box); (c) The point data with outliers (added outliers: 8000 points); (d) The point data with outliers (added outliers: 40,000 points). The second row shows the corresponding reconstructed curves.
Fig. 27. More reconstruction results from Chinese characters, sketch drawings and noisy images. We apply intensity-based thresholding on the inputs to get the corresponding point data, which inevitably contain a lot of noise due to unclean backgrounds.

Fig. 28. Reconstruction results of the ellipse point data with different densities. (a) The original point data (114 points); and the curves from (b) de Goes et al.’s [21] method and (c) ours. Since the right area of the ellipse is sparest, de Goes et al.’s [21] approach generates an open curve consequently. Our result is more faithful to the original shape. The second and third rows show the sub-sampled point data (38 points and 19 points, respectively), and the corresponding curves from de Goes et al.’s [21] method and ours. The quadtree depths for three data are 20, 12, 8, respectively. As can be seen in our results, the reconstruction error becomes bigger and bigger as the depth decreases.

However, it is still worthwhile to find a way to automatically determine the depth of quadtree subdivision in the future work.

Acknowledgments

We thank Ravish Mehra for providing data, In-Kwon Lee for offering the source code, and Fernando de Goes for giving the results of all testing data in the paper. Some models are downloaded from the AIM@SHAPE shape repository and the Stanford 3D scanning repository.

References