Computer-Aided Design 45 (2013) 657–670

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

A robust 2D point-sequence curve offset algorithm with multiple islands for contour-parallel tool path

Lin Zhiwei, Fu Jianzhong *, He Yong, Gan Wenfeng

State Key Laboratory of Fluid Power and Control, Department of Mechanical Engineering, Zhejiang University, Hangzhou, 310027, China

ARTICLE INFO

Article history:
Received 1 August 2011
Accepted 11 September 2012

Keywords:
2D point-sequence curve
Offset algorithm
Bridging
Stuck circle
Tree analysis

ABSTRACT

An offset algorithm is important to the contour-parallel tool path generation process. Usually, it is necessary to offset with islands. In this paper a new offset algorithm for a 2D point-sequence curve (PS-curve) with multiple islands is presented. The algorithm consists of three sub-processes, the islands bridging process, the raw offset curve generation and the global invalid loops removal. The input of the algorithm is a set of PS-curves, in which one of them is the outer profile and the others are islands. The bridging process bridges all the islands to the outer profile with the Delaunay triangulation method, forming a single linked PS-curve. With the fact that local problems are caused by intersections of adjacent bisectors, the concept of stuck circle is proposed. Based on stuck circle, local problems are fixed by updating the original profile with the proposed basic rule and append rule, so that a raw offset curve can be generated. The last process first reports all the self-intersections on the raw offset PS-curve, and then a procedure called tree analysis puts all the self-intersections into a tree. All the points between the nodes in even depth and its immediate children are collected using the collecting rule. The collected points form the valid loops, which is the output of the proposed algorithm. Each sub-process can be complete in near linear time, so the whole algorithm has a near linear time complexity. This can be proved by the examples tested in the paper.

1. Introduction

The roughing process is usually conducted in three-axis machine tools. At the roughing stage excess material should be removed as quickly as possible [1]. The most common method for roughing free-form surfaces is the layer-by-layer machining approach [2,3]. On each layer, the machine area, which is usually represented by a 2D PS-curve, can be obtained by intersecting the layer with the surface [4]. Generally, there are two most popular tool path patterns, direction-parallel and contour-parallel. The direction-parallel tool path suffers more cutting time and worse surface quality than that of the contour-parallel tool path in the case of complex shapes [5]. So for freeform surfaces, the contour-parallel tool path is always preferred. The main task of generating such a tool path is to offset the original profile successively. CAM software is always thirsty for a faster and more stable offset algorithm.

The literature on the 2D PS-curve offset algorithm can roughly be classified into four approaches: Voronoi diagram, level set, pair-wise and bisectors based. To understand the Voronoi diagram and level set method needs a good knowledge of mathematics. Persson [6] introduced a Voronoi diagram to generate a contour-parallel tool path bounded by segments without islands. Held et al. [7] further developed the Voronoi diagram to handle pockets with islands. But constructing a Voronoi diagram is a time consuming process and it also has numerical instability [4,8]. Kimmel and Bruckstein [9] presented a level set method derived from fluid dynamics to handle pockets. However, the main drawback of this method is the initialization of Φ, where Φ is the propagation function [5].

Pair-wise offset can be done by offsetting the start and end points of the segment along the normal direction for an offset distance, while in bisector based offset, the offset point is picked up on the bisector of two adjacent segments. Kim and Jeong [10] proposed a procedure which is able to offset a B-spline curve with islands inside. Actually, their offset algorithm is not really free-form curve based. The B-spline is first converted to Bezier, and the Bezier curve is then subdivided into a PS-curve. The initial offset curve is obtained pair-wisely. But only two cases of local problems are discussed and fixed. Global invalid loops are removed by checking loop area. Positive area means that the loop is valid, otherwise invalid. In their algorithm, islands are bridged manually to the outer profile.

Wong and Wong [11] presented an algorithm which can automatically bridge islands to the outer profile by the nearest distance. But the total times of minimum distance calculation between two PS-curves would be $(n-1)(n-2)/2$, in which n is the total number

* Corresponding author. Tel.: +86 13819193086.
E-mail address: fjz@zju.edu.cn (J. Fu).

0010-4485/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2012.09.002

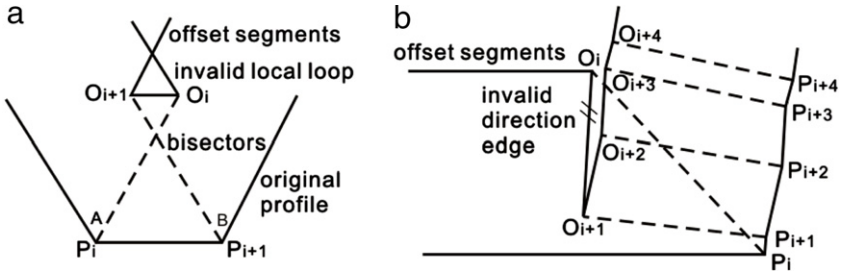

Fig. 1. Failing cases of Lai's and Kim's offset algorithms.

of curves, including the outer profile and islands. They use bisectors to generate the initial offset curve. If the two adjacent bisectors intersect, the segment containing these two bisectors is removed and a new bisector is constructed by the adjacent segments. Global invalid loops are removed by checking the direction of the loop. Unfortunately, the local loop removing algorithm is not given in detail. Frankly, the algorithm developed by us is inspired by the idea of bisector intersection.

Choi and Park [8], Park and Choi [12] and Park and Chung [13] proposed a PWID (pair-wise interference-detection) test based offset algorithm. In their work, they define three interfering segments, which are full-interfering segment, partial interfering segment and reverse-interfering segment; and three interfering relations, which are partial interfering relation, reverse-interfering relation and full-interfering relation. The PWID test includes two basic operations, which are replace-full-segment and replace-reverse-segment. With these definitions and operations, the algorithm can work in linear time. However, in their algorithm, islands are also bridged to the outer profile manually.

Lai et al. [14] proposed a FLTM (forward locus tracing method) based algorithm for invalid loops removal. In their work, all degenerate segments are eliminated by a Voronoi diagram method. Then the FLTM searches for all intervals split by intersections of complicated planar curves directly and transforms 2D transversal intersection problems into 1D interval identification. If the first interval has no interference with other objects, then the odd intervals group can form valid loops. The problem is the Voronoi diagram method they use could go wrong between two adjacent inner angles where $A + B \geq \pi$ when large offset is required, as in Fig. 1(a).

In the algorithm developed by Kim [4] and Kim et al. [15], the raw offset curve is calculated by bisectors. The offset edge which has a reverse direction compared to the corresponding original edge is defined as an invalid offset edge. Local problems are fixed by removing the invalid offset edge. Global loops are removed also by checking the direction of the loop. Islands are merged similarly to the global loops removing process. The algorithm is near $O(n)$ time. However, their algorithm cannot handle cases like in Fig. 1(b).

Lee et al. [16] also use bisectors to generate the initial offset curve. Besides direction validity, they also check position validity. An offset edge is defined position invalid when its start and end points are both in the offset area, otherwise position valid. Local loops are removed by the proposed raw offset line algorithm, in which five different cases are included and then the raw offset curve is established. The global loops are removed by comparing the distance from the offset point to the original profile with the given offset distance. The problem is calculating all the distances from offset points to the original profile involves $O(n^2)$ time if not optimized. The algorithm survives the case in Fig. 1(b), but it fails the cases in Fig. 2(a) and (b). The methods dealing with invalid position edge or invalid direction edge in their paper could have been improper. The fact might be, for an offset edge, it can be used even if it's an invalid position edge (as in Fig. 2(a)), or an invalid direction edge (as in Fig. 2(b)).

Interesting as it is, the work performed by Liu et al. [17] is more for computer graphics science than for a CNC tool path generation process. The salient feature of their work is that it can deal with profile self-intersection, overlapping and small arc problems. The initial offset curve is also obtained pair-wisely, but more cases are covered to fix local problems. Due to its ability, more time is required to get the offset curves. It takes them on average 21 ms to calculate the offset of a polyline including only 8 segments on a 1.7 GHz Pentium IV CPU. With this performance, their algorithm might not be qualified to be applied in the CNC area where thousands of segments might be considered at one offset.

Earlier researches on a PS-curve offset or CNC pocketing algorithm which are not included above can be found in reviews [5,18]. From the above literature, it can be learned there are three main problems in the PS-curve offsetting process: (1) dealing with islands, (2) fixing local problems and (3) removing global invalid loops. If islands exist, basically, there are two ways to deal with them. The first way is to bridge all the islands to the outer profile; and the second way is to offset the outer profile inwards and islands outwards and then trim and merge the offset curves if they intersect. In practice, we have observed from PowerMILL that islands are bridged to the outer profile. Although PowerMILL works very fast and stable, both of its bridging algorithms and offset algorithms remain unknown to the public. In this paper, we have developed a robot offset algorithm which can handle islands. The input of the offset algorithm is a set of PS-curves, in which one of them is the outer profile and others are islands; and the output is offset curves which can be used as a contour-parallel tool path in the roughing process.

The remainder of the paper is organized as follows. Section 2 bridges automatically all the islands to the outer profile with Delaunay triangulation, forming a single PS-curve. Section 3 firstly introduces local profile updating rules, the basic rule and the append rule. With these two rules, the raw offset PS-curve is obtained by updating the original profile. Based on this, Section 4 removes all global invalid loops with a procedure called tree analysis. In Section 5, the time complexity of the proposed algorithm is analyzed. Several examples including a cutting simulation are done to verify the proposed algorithm in Section 6. The last section closes this paper.

2. Bridging islands to the outer profile

The input of the bridging process is a PS-curve set, in which the 0th element is the outer profile (with counterclockwise direction) and the other elements are islands (with clockwise direction), as shown in Fig. 3; and the output is a single PS-curve which connects all the islands and the outer profile. The requirement for the bridging process is that the constructed bridges should not cross with any other bridges or any PS-curves. In this section, we propose a new bridging algorithm based on Delaunay triangulation. To make it easier to understand, we first define the point structure which is used throughout the paper.

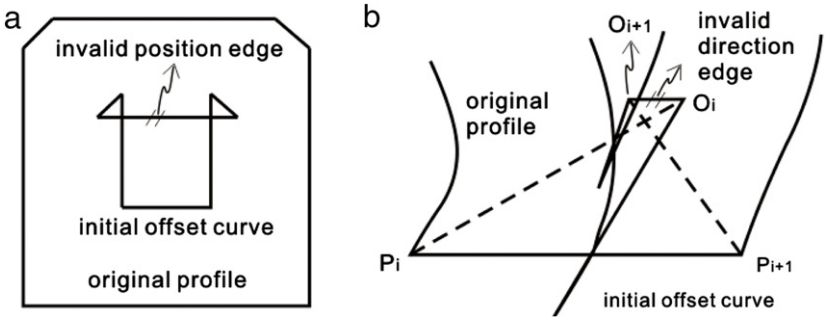

Fig. 2. Failing cases of Lee's offset algorithm.

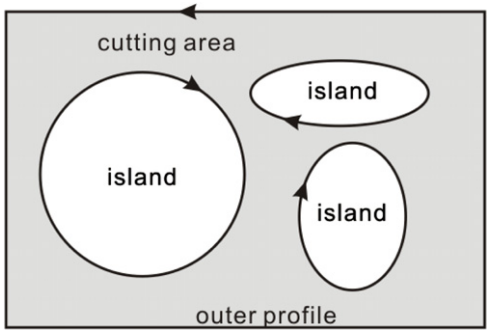

Fig. 3. Outer profile and islands of the detected cutting area.

Definition 1. Struct Point {double x, y, z; int pointIndex; int curveIndex;}.

In this structure, the real numbers x, y and z are the coordinates of the point in Euclidean space; the integer pointIndex is the index of the point in the PS-curve; and the integer curveIndex is the index of the curve (that the point belongs to) in the curve set. For example, for a point P, if $P\cdot\text{pointIndex} = 12$ and $P\cdot\text{curveIndex} = 7$, it means that this point is the 12th point of the 7th curve in the curve set. Each point of each curve from the curve set will be duplicated and imported into a point set $\{P_i\}$ for Delaunay triangulation, with both pointIndex and curveIndex values assigned.

2.1. Bridge construction

For planar point set $\{P_i\}$, a triangulation is said to be "Delaunay" if for each edge the circle circumscribing one triangle does not contain the fourth point. To get the Delaunay triangulation of point set $\{P_i\}$, we employ the incremental algorithm proposed by Berg et al. [19]. The time complexity of their algorithm is $O(n \log n)$, in which n is the size of the point set. As the Delaunay triangulation process reaches its highest performance when the point set is randomly distributed, the point set should be randomized before triangulation. To maintain the point original information for later use, before adding each duplicated point from the input curve set to the point set, it is necessary to assign both the pointIndex and curveIndex values. This is why the pointIndex and curveIndex are necessary in the point structure. Fig. 4(a) shows the result of Delaunay triangulation of the PS-curves in Fig. 3. And Fig. 4(b) shows the triangle set after some unnecessary triangles are removed. These unnecessary triangles include triangles with: (1) all three vertexes on the same curve, and (2) all three vertexes on three different curves.

In the left triangle set, each triangle connects two PS-curves. Taking the triangle ABC shown in Fig. 4(b) for example, vertex A is on island 1, vertexes B and C are on the outer profile. Bridges can be built directly on this triangle to connect the outer profile and island 1, with one end on vertex A and the other end either on vertex B or vertex C, or some point H on edge BC, as shown in Fig. 4(b). To make it simple, we bridge two PS-curves on two vertexes, namely, for triangle ABC, the bridge connecting the outer profile and island 1 is edge AB or edge AC. To make the bridge shorter, the shorter edge between AB and AC is selected as a potential bridge. The bridge structure is defined as:

Definition 2. Struct Bridge {Point end1; Point end2; double length;}.

In this structure, end1 and end2 are the two ends of a bridge; and the real number length is the length of the bridge. For each triangle in the triangle set, the potential bridges are added into a potential bridge set. The number of potential bridges is equal to the number of the triangles in the triangle set.

To build bridges that can connect all the islands to the outer profile, a bridge construction procedure is developed as the flow chart in Fig. 5. In Fig. 5, the macro HUGE is a very big real number used to initialize the minimum length of the bridge. The bridge construction procedure firstly finds in the potential bridge set the shortest potential bridge connecting an island to the outer profile; adds this potential bridge to a bridge set and removes all the potential bridges that connect this island and the outer profile; marks this island as the outer profile by negating the curveIndex of the potential bridge end connecting to this island. This process repeats until there is no potential bridge left in the potential bridge set. In Fig. 5, picking the shortest potential bridge as a bridge is not

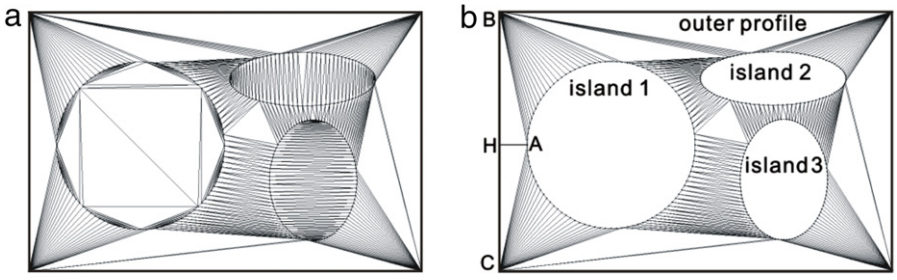

Fig. 4. Delaunay triangulation; (a) Delaunay triangulation of point set from PS-curves; (b) triangle set with unnecessary triangles removed.

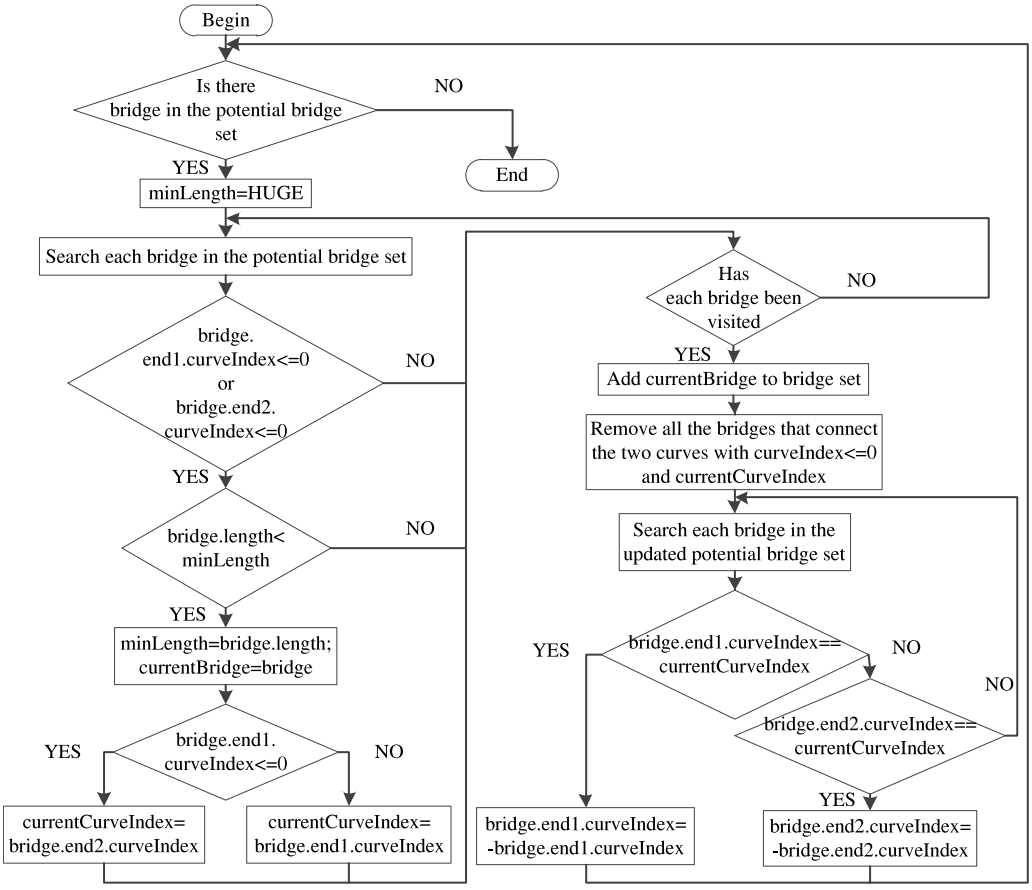

Fig. 5. The flow chart of the bridge construction procedure.

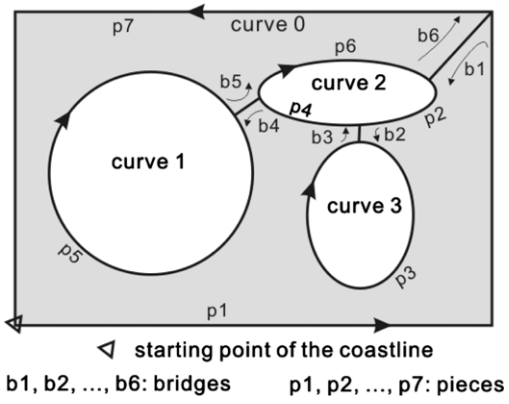

Fig. 6. Bridges and pieces of curves divided by the bridges.

a necessary condition to the bridging problem itself. In fact, any potential bridge can be picked as a bridge.

The output of the bridge construction procedure is a set of bridges, as shown in Fig. 6. The number of bridges in the bridge set is equal to the number of islands inside the outer profile. In this bridge set, bridges with negative curveIndex exist. These negative curveIndexes have to be set positive. After that, all the bridges in the bridge set can be used for the connecting work.

2.2. Curve connection

With the obtained bridges, the following task is to link all the islands and the outer profile as a whole to form a single PS-curve. When an island is bridged to the outer profile, there are actually two bridges with opposite directions, one bridge in and the other bridge out. However, in the bridge set obtained in Section 2.1, the directions of the bridges are not assigned. As a bridge has two ends, i.e. end1 and end2, the bridge direction is assigned as end1 pointing to end2. For each bridge in the obtained bridge set, an opposite bridge is built and added to the bridge set. For example, suppose b is a bridge in the bridge set, and g is the opposite bridge of b, then $g \cdot \text{end1} = b \cdot \text{end2}$ and $g \cdot \text{end2} = b \cdot \text{end1}$. When all the opposite bridges are added into the bridge set, the number of bridges in the bridge set doubles.

In the updated bridge set, each bridge can be expressed with four variables: end1·curveIndex, end1·pointIndex, end2·curveIndex and end2·pointIndex, in which end1 · curveIndex is the curve index on which the start point of the bridge locates, and end2·curveIndex is the curve index on which the end point of the bridge locates; end1·pointIndex and end2·pointindex identify the point indexes in the two curves, respectively.

The outer profile and the islands are cut into pieces by the bridges, as shown in Fig. 6. Starting from the starting point of the outer profile, tracing piece by piece in a right order, as the $p1, p2, \ldots, p7$ in Fig. 6, all the pieces can be linked. As each piece can be identified by three variables, a piece structure is defined as:

Definition 3. Struct Piece {int curveIndex; int startPointIndex; int endPointIndex;}.

In this structure, curveIndex is the index of the curve to which the current piece belongs. StartPointIndex and endPointIndex are the start point index and the end point index of the piece on the curve.

As shown in Fig. 6, taking $p4$ (piece 4) for example, it is enclosed by a pair of bridges, the inward bridge $b3$ and the outward bridge $b4$. Suppose in the previous step, the inward bridge has been given.

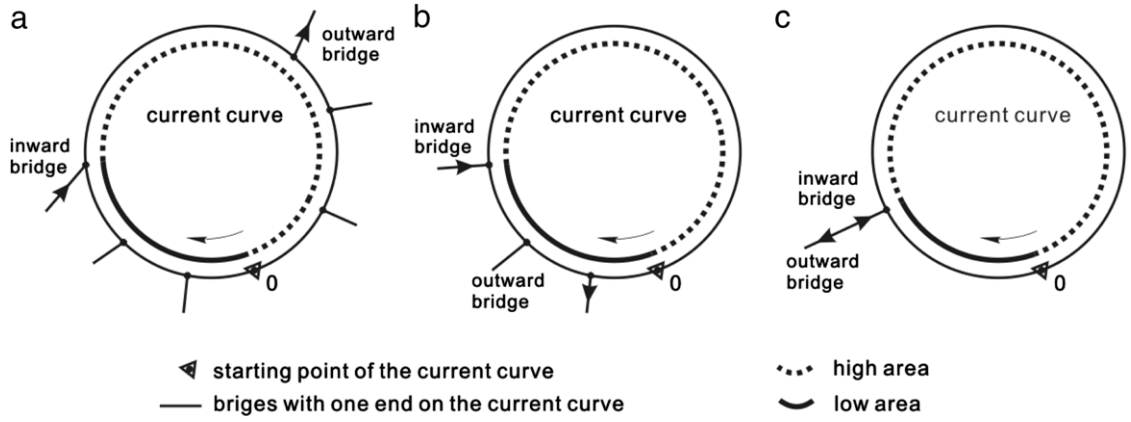

Fig. 7. Three cases of the outward bridge locations; (a) outward bridge in high area; (b) outward bridge in low area; (c) outward bridge coincides with inward bridge.

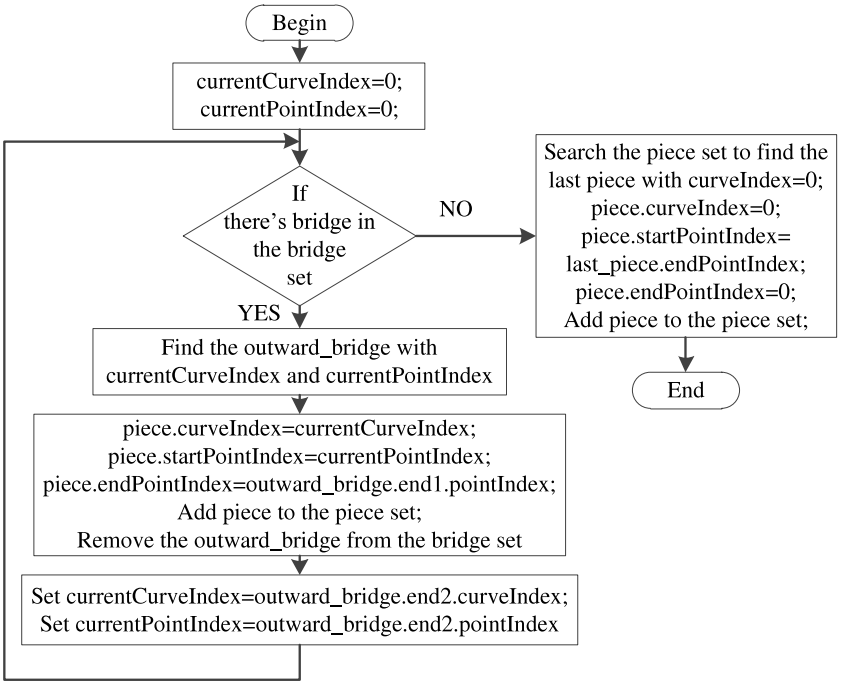

Fig. 8. The flow chart of finding all the pieces of curves.

In the current step, it is necessary to find the outward bridge, so that the enclosed piece can be completely located. As the end point (current point) of the inward bridge is on curve end2 · curveIndex (current curve), so the start point of the outward bridge must also be on the current curve. According to the position of the current point and the starting point of the current curve, the current curve is divided into two areas, the high area and the low area. The high area is the part of the curve starting from the inward bridge to the end point of the current curve, while the low area is the part of the curve starting from the starting point of the current curve to the inward bridge, as shown in Fig. 7. As there might be many bridges on the current curve, three cases need to be discussed to find the only outward bridge for the current inward bridge.

Case 1. If there are bridges in the high area, then the outward bridge is the bridge in the high area which has the minimum end1 · pointIndex, as shown in Fig. 7(a).

Case 2. If there's no bridge in the high area, but there are bridges in the low area, then the outward bridge is the bridge with minimum end1 · pointIndex, as shown in Fig. 7(b).

Case 3. If there's no bridge either in the high area or in the low area, then the outward bridge is the bridge with end1 · pointIndex is equal to the end2 · pointIndex of the inward bridge, as shown in Fig. 7(c).

With the above discussion, the index of the outward bridge in the bridge set for an assigned inward bridge can be found. Then all the pieces of curves can be found according to the flow chart showing in Fig. 8. The output of Fig. 8 is a piece set, with which all the pieces of curves can be collected and connected to form a single PS-curve.

3. Raw offset curve generation

In Section 2, the input PS-curve set, including the outer profile and islands, have been linked into one single PS-curve. This obtained PS-curve will be used as the input of this section. And the output of this section is the raw offset curve without any local problems, like local self-intersections. At first, some basic definitions that will be used in this section are given as follows.

Definition 4. Struct Bisector {Point *P*; Point *O*;}.

In this struct, point *P* stores a point of the PS-curve; point *O* stores the corresponding offset point of point *P* under a given offset

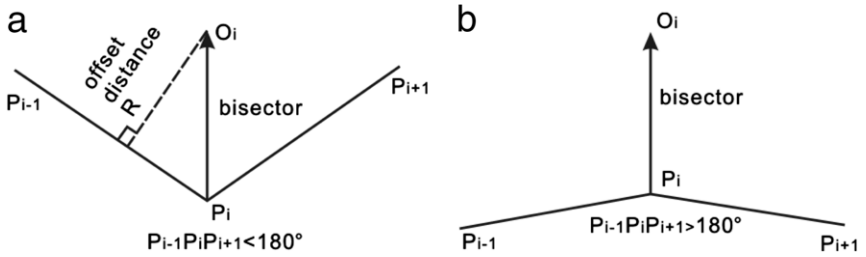

Fig. 9. Two cases of calculating bisectors.

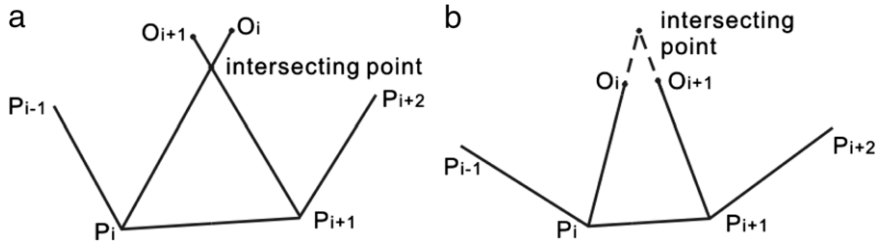

Fig. 10. Two adjacent bisectors intersection; (a) two bisectors intersect; (b) two bisectors do not intersect.

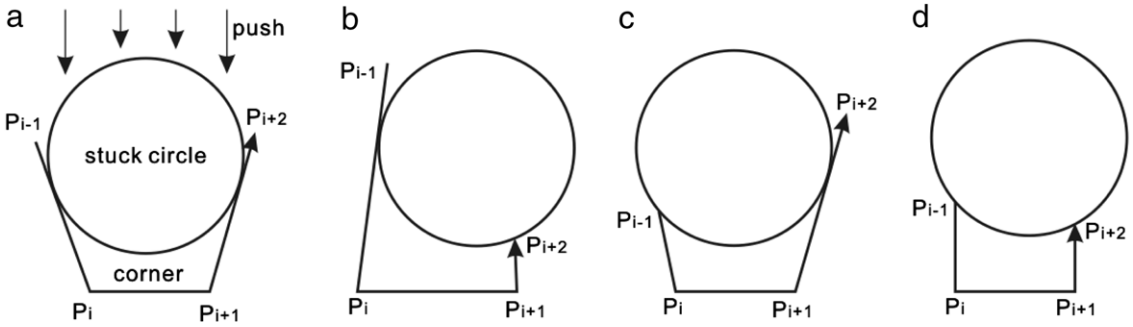

Fig. 11. Stuck circle definition and four different situations.

distance. From this definition, a bisector is a segment with a certain length. As shown in Fig. 9, line P_iO_i is the angular bisector of angle $P_{i-1}P_iP_{i+1}$; and point O_i is the offset point of P_i. O_i can be calculated by Eq. (1):

$$\begin{cases} O_i = P_i + \mathbf{e} \cdot R/\sin(0.5\arccos(\mathbf{e}_1 \cdot \mathbf{e}_2)), & \mathbf{e}_1 \times \mathbf{e}_2 < 0 \\ O_i = P_i - \mathbf{e} \cdot R/\sin(0.5\arccos(\mathbf{e}_1 \cdot \mathbf{e}_2)), & \mathbf{e}_1 \times \mathbf{e}_2 > 0, \end{cases} \tag{1}$$

in which R is the given offset value; $\mathbf{e_1}$, $\mathbf{e_2}$ are the unit vectors of vectors $\mathbf{P_iP_{i-1}}$ and $\mathbf{P_iP_{i+1}}$, respectively; $\mathbf{e}$ is the unit vector of $\mathbf{e_1} + \mathbf{e_2}$; $\mathbf{e_1} \cdot \mathbf{e}_2$ is the dot product while $\mathbf{e_1} \times \mathbf{e_2}$ is the cross product of $\mathbf{e_1}, \mathbf{e_2}$. In fact, the case $\mathbf{e_1} \times \mathbf{e_2} < 0$ happens when angle $P_{i-1}P_iP_{i+1} < 180°$ (as in Fig. 9(a)), while $\mathbf{e_1} \times \mathbf{e_2} > 0$ happens when angle $P_{i-1}P_iP_{i+1} > 180°$ (as in Fig. 9(b)). The bisector defined here is the segment P_iO_i. Two adjacent bisectors are defined intersection only when the intersecting point locates on both the two bisectors (as in Fig. 10(a)); otherwise they do not intersect (as in Fig. 10(b)). All local problems are caused by adjacent bisectors intersecting.

Definition 5. Stuck circle.

For four consecutive points P_{i-1}, P_i, P_{i+1} and P_{i+2} on the PS-curve, if the bisectors P_iO_i and $P_{i+1}O_{i+1}$ intersect, then a stuck circle is defined if it exists. As shown in Fig. 11(a), the radius of a stuck circle is equal to the given offset distance R; and the position of the circle can be obtained by pushing the circle towards the corner until it is stuck by the segments $P_{i-1}P_i$ (besides P_i) and $P_{i+1}P_{i+2}$ (besides P_{i+1}). The most important thing is that the minimum distance from the stuck circle center to segments $P_{i-1}P_i$ and $P_{i+1}P_{i+2}$ must be R, and the minimum distance from the stuck circle center to segment P_iP_{i+1} must be equal to or bigger than R.

Fig. 11 gives four common cases of stuck circle. In Fig. 11(a), the stuck circle is supported by two tangential points on $P_{i-1}P_i$ and $P_{i+1}P_{i+2}$. In Fig. 11(b) and (c), the stuck circle is supported by tangential point of one segment and end point of the other segment. In Fig. 11(d), the stuck circle is supported both by two end points P_{i-1} and P_{i+2}.

There are also cases that when the bisectors P_iO_i and $P_{i+1}O_{i+1}$ intersect, but there's no stuck circle, as shown in Fig. 12. In Fig. 12(a), the circle is supported by points P_{i-1} and P_{i+1}. Being supported by point P_{i+1} does not satisfy the stuck circle definition. In Fig. 12(b), though the circle is supported by two end points P_{i-1} and P_{i+1}, and the minimum distances from circle center to both $P_{i-1}P_i$ and $P_{i+1}P_{i+2}$ are R, but the minimum distance from circle center to P_iP_{i+1} is less than R, which does not satisfy the stuck circle definition, either. In Fig. 12(c), there's simply no stuck circle.

Whichever the case is, when four points P_{i-1}, P_i, P_{i+1} and P_{i+2} are given, basic geometry calculations can be done to identify the stuck circle if it exists.

3.1. Profile updating rules

In this paper, local problems are fixed by updating local original profile. The fact is that local problem happens only when two adjacent bisectors P_iO_i and $P_{i+1}O_{i+1}$ intersect. So if the two bisectors P_iO_i and $P_{i+1}O_{i+1}$ intersect, then construct a stuck circle for four points P_{i-1}, P_i, P_{i+1} and P_{i+2}. If the stuck circle exists, then apply

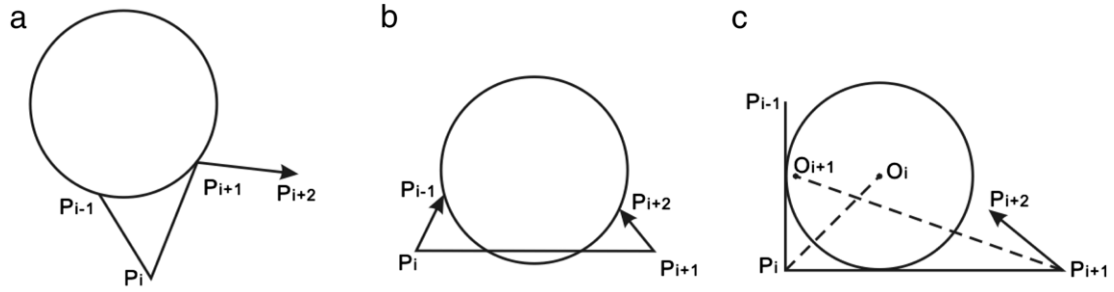

Fig. 12. No stuck circle cases when two adjacent bisectors intersect.

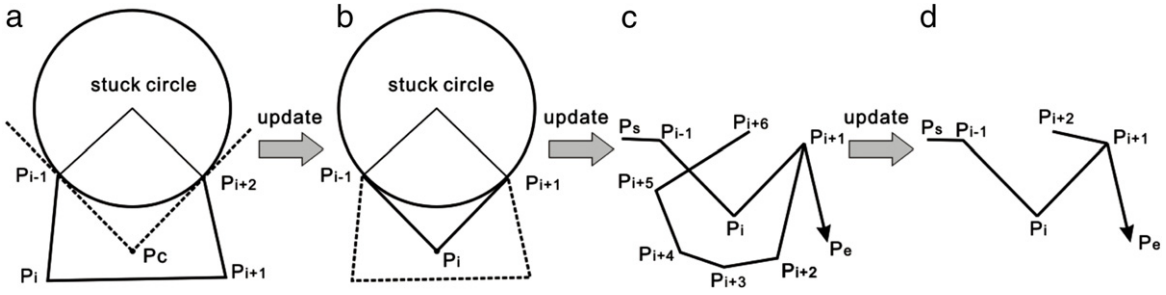

Fig. 13. Basic rule for updating profile.

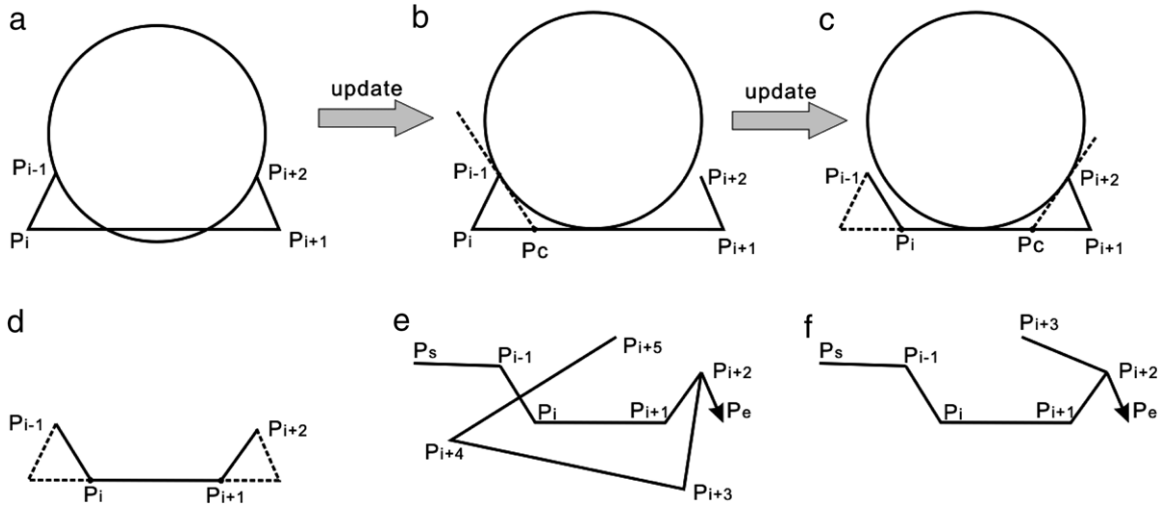

Fig. 14. Append rule for updating profile.

the basic profile updating rule to update the local profile. If there's no stuck circle, then apply the append profile updating rule.

Basic rule. Find the intersecting point P_c of the two tangent lines which are through the two supporting points, as shown in Fig. 13(a). Update the original four points P_{i-1}, P_i, P_{i+1} and P_{i+2} with three points P_{i-1}, P_c and P_{i+2}. Update the sequence of the points in the PS-curve, as in Fig. 13(b). Starting from point P_{i+2}, remove the points that are in the right half-plane divided by polyline $P_sP_{i-1}P_iP_{i+1}P_e$ until a point in the left half-plane is met and updates the point sequence, as in Fig. 13(c) and (d). P_s is a point on the extension of segment $P_{i+1}P_{i-1}$, and P_e is the memorized point of the original P_{i+1} as in Fig. 13(a). Recalculate the bisectors of updated points P_{i-1}, P_i and P_{i+1} with Definition 1.

It has to be explained that although in Fig. 13, the two supporting points are both end points of $P_{i-1}P_i$ and $P_{i+1}P_{i+2}$, the basic rule can be popularized to any case in Fig. 11. The reason for removing the right half-plane points is to make sure that no self-intersection happens on the updated profile.

Append rule. Check angle $P_{i-1}P_iP_{i+1}$, if it is less than π, then update four points P_{i-1}, P_i, P_i, P_{i+1} with the basic rule, as in Fig. 14(b). Check angle $P_iP_{i+1}P_{i+2}$, if it is less than π, then update four points P_i, P_{i+1}, P_{i+1}, and P_{i+2} with the basic rule, as in Fig. 14(c). Starting from point P_{i+3}, remove the points that are in the right half-plane divided by polyline $P_sP_{i-1}P_iP_{i+1}P_{i+2}P_e$ until a point in the left half-plane is met and update the point sequence, as in Fig. 14(e) and (f). P_s is a point on the extension of segment $P_{i+2}P_{i-1}$, and P_e is the memorized point of the original P_{i+1} as in Fig. 14(a). Recalculate the bisectors of updated points P_{i-1}, P_i, P_{i+1} and P_{i+2}.

3.2. Raw offset curve generation

Raw offset PS-curve is defined to be the offset PS-curve without any local problems. In this section, a raw offset PS-curve will be generated by updating the original profile. At the beginning, adjacent collinear segments on the original PS-curve are merged and the bisector for each point is calculated with Definition 4. The flow chart of the whole updating process is shown in Fig. 15.

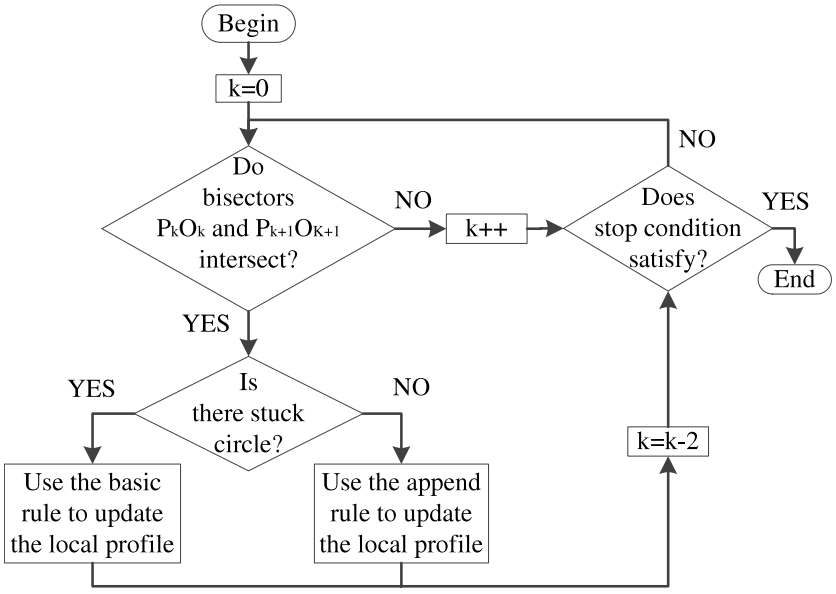

Fig. 15. Flow chart of the raw offset curve generation process.

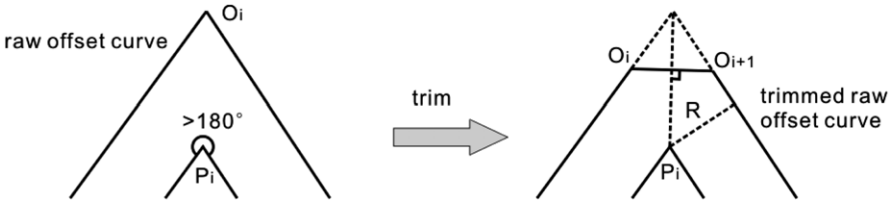

Fig. 16. Trimming of the obtained raw offset curve.

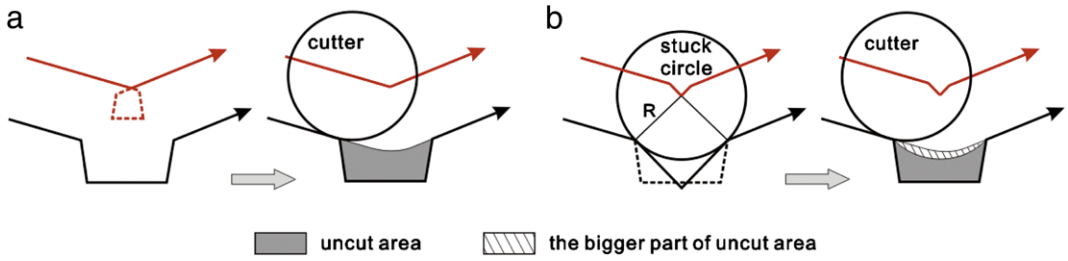

Fig. 17. Benefit of using stuck circle from a machining point of view; (a) uncut area left by the conventional offset methods; (b) uncut area left by stuck circle offset method.

In Fig. 15, if the bisector of current checking point P_k does not intersect the bisector of P_{k+1}, then go on checking the bisectors of P_{k+1} and P_{k+2}; else the basic rule or the append rule is used, and the next checking point becomes P_{k-2}. This is because when the basic rule or the append rule is used, the bisector of point P_{k-1} may have been updated. It is necessary to see whether the bisectors of P_{k-2} and P_{k-1} intersect.

In the updating process, the sequence k might be smaller than zero or bigger than the size of the updated PS-curve, to make it easier, each time when k changes, Eq. (2) is applied to ensure k in the range of $[0, sz)$, where sz is the current total point number of the PS-curve:

$$k = \begin{cases} k, & 0 \le k < sz \\ sz + k, & k < 0 \\ k - sz, & k \ge sz. \end{cases} \tag{2}$$

When all the points in the original PS-curve have been checked at least once, which also means that the checking point has to go through one round around the PS-curve, all the local problems should have been fixed except for that near the starting area, if there's any. To fix all the local problems, an additional round goes through the updated PS-curve, and then the process stops. This is the stop condition for the flow chart in Fig. 15.

After this procedure, raw offset PS-curve can be obtained directly from the offset point of the updated bisectors. To shorten the tool path, and also to avoid a sharp angle in the raw offset PS-curve, the inner angles which are larger than 180° are carefully trimmed with segments, as in Fig. 16. The disadvantage of the trimming process is that it will increase the point number in the raw offset curve. In order to control the increment of point number, only inner angles that are larger than 240° or 270° or even larger are trimmed. From a machining point of view, the use of a stuck circle makes the offset curve more close to the original curve, so more material can be cut than in conventional ways, as shown in Fig. 17.

4. Global invalid loops removal

On the raw offset PS-curve obtained in Section 3, global invalid loops may exist. In this section, all the global invalid loops will be removed by a procedure called tree analysis (TA). To make it

more understandable, a simple demonstration of the TA procedure is given below. Before that, two definitions used in this section are given as follows.

Definition 6. Valid point.

A point O which is inside the PS-curve is defined valid if it is not in the offset region [16] of the PS-curve, otherwise invalid. To check the validity of O needs to calculate the minimum distance from O to the PS-curve. This process results $O(n)$ time, where n is the number of segments in the PS-curve. However, in the proposed offset algorithm, only three points are necessary to check their validity. So the computation is not the concern.

Definition 7. Struct SelfIntersection {Point SI; double s; double b}.

In this struct, point SI is the self-intersecting point of two segments on the PS-curve; real number s and b represent the position of the intersecting point on the two self-intersecting segments, in which s is always smaller than b. A parameter range of 1 is evenly assigned to each segment. For example, for self-intersection (SI, 7.3, 35.6), the intersecting point SI locates on the 7th and 35th segments.

For two self-intersections (SI_1, s_1, b_1) and (SI_2, s_2, b_2), very simple operators $\subseteq$, $\supseteq$ and $\otimes$ are defined:

$(SI_1, s_1, b_1) \subseteq (SI_2, s_2, b_2)$ if $s_1 > s_2$ and $b_1 < b_2$. This is a "belong to" operator.

$(SI_1, s_1, b_1) \supseteq (SI_2, s_2, b_2)$ if $s_1 < s_2$ and $b_1 > b_2$. This is a "contain" operator.

$(SI_1, s_1, b_1) \otimes (SI_2, s_2, b_2)$ if $s_1 < s_2 < b_1$ and $b_2 > b_1$, or $s_2 < b_1 < b_2$ and $s_1 < s_2$. This is a "interfere" operator.

From the literature, it is always indispensable to calculate all the self-intersecting points of the raw offset curve to remove global invalid loops. The problem can be simplified as reporting all the intersecting points for a collection of n segments. An obvious algorithm with $O(n^2)$ time can be easily developed, where n is the number of segments. In Ref. [8], a sweep line algorithm with $O((n+s)\log m)$ time is used to find all the self-intersections, in which n is the number of segments, s is the number of self-intersections and m is the number of monotone chains. In this paper we use the well-known "Bentley–Ottmann Algorithm" [20] to report all self-intersections since it is relatively easy to both understand and implement. This algorithm requires $O((n+s)\log n)$ time, where n is the number of segments and s is the number of self-intersections.

4.1. A demonstration of TA procedure

Fig. 18 shows a 17-segment original profile and its raw offset PS-curve. There's no local problem in the raw offset curve, but there are three global invalid loops. The task of the TA procedure is to remove all these three loops. As in Fig. 18, suppose the first point of the raw offset curve is a valid point (see Definition 6). Based on this, all the self-intersections of the raw offset curve are found and stored in the SelfIntersection struct (see Definition 7). There are in all six self-intersections in the raw offset curve. They are sorted according to the s value in ascending order as (SI_1, 1.65, 16.35), (SI_2, 2.45, 15.55), (SI_3, 3.55, 8.45), (SI_4, 4.35, 7.65), (SI_5, 9.55, 14.45), (SI_6, 10.35, 13.65). If the first point is also considered a self-intersecting point of the first segment (0th segment) and the last segment (17th segment), this intersection can be recorded as (SI_0, 0, 17.99). With this fake one, all the self-intersections in ascending order are (SI_0, 0, 17.99), (SI_1, 1.65, 16.35), (SI_2, 2.45, 15.55), (SI_3, 3.55, 8.45), (SI_4, 4.35, 7.65), (SI_5, 9.55, 14.45), (SI_6, 10.35, 13.65). With the operators $\subseteq$ and $\supseteq$ defined in Definition 5, these self-intersections can be put into a tree, as shown in Fig. 19(a). In the tree, the fake self-intersection (SI_0, 0, 17.99) is the root, other self-intersections are

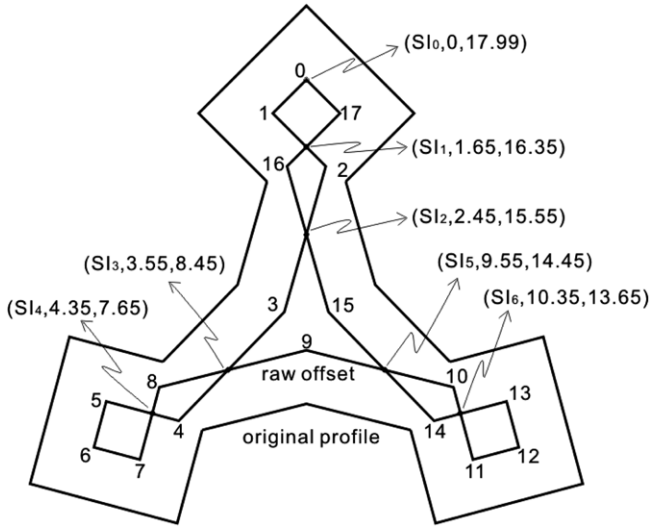

Fig. 18. A simple demonstration of tree analysis procedure.

the nodes. The depth of root and each node have also been marked in the tree.

The TA procedure first checks the depth of each node including the root to see whether it is even or odd. If it is in even depth, then use the collecting rule to collect all the points between the node itself and its immediate children.

The collecting rule is simple. As in Fig. 19(b), the self-intersection (SI_0, s_0, b_0) is in even depth, (SI_1, s_1, b_1) to (SI_n, s_n, b_n) are its immediate children. The following points are to be collected:

$$SI_0, P_{s_0+1}, P_{s_0+2}, \ldots, P_{s_1},$$
$$SI_1, P_{b_1+1}, P_{b_1+2}, \ldots, P_{s_2},$$
$$SI_2, P_{b_2+1}, P_{b_2+2}, \ldots, P_{s_3},$$
$$\vdots$$
$$SI_n, P_{b_n+1}, P_{b_n+2}, \ldots, P_{b_0}.$$

When being subscripts of points, only the integer parts of s_0 to s_n or b_0 to b_n are taken. These collected points can form a valid loop.

So for the above demonstrating case, the root (SI_0, 0, 17.99) is in the 0th depth, which is even, points between the root and its immediate child, which is (SI_1, 1.65, 16.35), is to be collected using the collecting rule. The collected points are: SI_0, P_1, SI_1, P_{17}. These four points form the top valid loop, as in Fig. 18.

Node (SI_2, 2.45, 15.55) is in 2nd depth, which is also even, its immediate children are (SI_3, 3.55, 8.45) and (SI_5, 9.55, 14.45). So the collected points are: $SI_2, P_3, SI_3, P_9, SI_5, P_{15}$. These six points form the middle valid loop, as in Fig. 18.

Node (SI_4, 4.35, 7.65) is in 4th depth, which is even, but it has no child. So the collected points are: SI_4, P_5, P_6, P_7. These four points form the bottom left valid loop, as in Fig. 18. The bottom right loop can be obtained in the same way.

4.2. Removing interferential self-intersections

The above demonstration is simple because for the 6 self-intersections, there are no interferential self-intersections, as in Definition 7. For more complicated cases, interferential self-intersection may exist. In these cases, before the TA procedure, interferential self-intersections must be carefully examined and removed. Otherwise the TA procedure will fail.

Empirically, interferential self-intersections come in threes and two of them must be adjacent if all the self-intersections are sorted in ascending order according to s value. Let (SI_1, s_1, b_1), (SI_2, s_2, b_2),

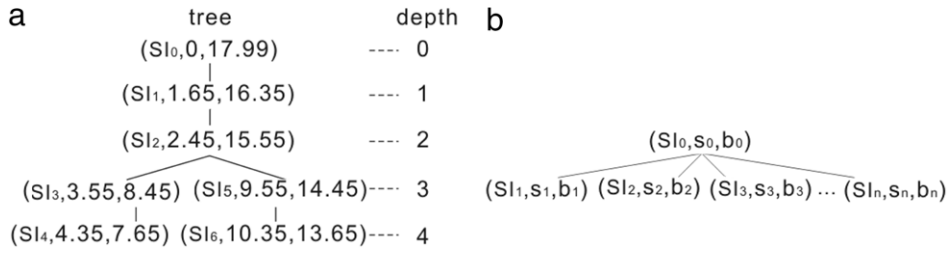

Fig. 19. Tree of self-intersections; (a) self-intersecting tree of the demonstration case; (b) the collecting rule.

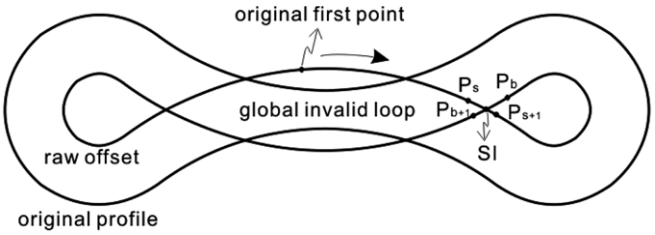

Fig. 20. Find the first valid point.

(SI_3, s_3, b_3) be the three interferential self-intersections. There are two cases that need to be discussed.

For the first case, one of the three self-intersections interferes with both the other two, but the other two do not interfere. In this case, remove the interferential free two self-intersections. For example, (SI_1, s_1, b_1) interferes with both (SI_2, s_2, b_2) and (SI_3, s_3, b_3), but (SI_2, s_2, b_2) does not interfere with (SI_3, s_3, b_3), then remove self-intersections (SI_2, s_2, b_2) and (SI_3, s_3, b_3).

For the second case, any two of the three self-intersections interfere. In this case, remove all these three self-intersections. For example, (SI_1, s_1, b_1) interferes with both (SI_2, s_2, b_2) and (SI_3, s_3, b_3), and (SI_2, s_2, b_2) interferes with (SI_3, s_3, b_3), too, then remove the three self-intersections (SI_1, s_1, b_1), (SI_2, s_2, b_2) and (SI_3, s_3, b_3).

4.3. Locating the first valid point

Note that in the demonstrating case it is assumed that the first point of the raw offset curve is a valid point. As a matter of fact, the TA procedure relies highly on this assumption. If the first point is an invalid point, then the TA procedure fails. So before starting the TA procedure, we must make sure that the first point is valid. If we could have known an offset point P_i to be valid, we can rearrange to point sequences of the raw offset curve to make P_i the first point. If the first point of the raw offset curve originally locates on the invalid loop, and it's in the middle of many other invalid points, as in Fig. 20, then it will take much time to check each point until a valid point is met.

After interferential self-intersections are properly removed, note the fact that the left self-intersecting points exist for the reason to distinguish valid and invalid loops, that is to say, a self-intersecting point must be valid. So there must be some valid points near a self-intersection. Suppose (SI, s, b) is a self-intersection, as in Fig. 20, for four points P_s, P_{s+1}, P_b, and P_{b+1}, it is a fact that one of them must be valid. So it is only necessary to check at most three of them. And rearrange the raw offset PS-curve to make the met valid point the first point, meanwhile, update the s and b value of each self-intersection.

5. Time complexity analysis

The input of the algorithm is a set of PS-curves, in which one of them is the outer profile and the others are islands and the output is the offset curves. The whole algorithm is made up of three parts: (1) bridging islands to the outer profile, (2) raw offset curve generation and (3) global invalid loops removal.

In the first part, obviously, most of the time has been spent on the Delaunay triangulation process, so the time complexity for the bridging process is $O(n \log n)$, in which n is the number of points from all the input PS-curves, including the outer profile and the islands.

In the third part, most of the time has been spent on the self-intersection reporting process, so the time complexity for the global invalid loops removal process is $O(n \log n)$, in which n is the number of points in the raw offset curve generated in Section 3.

As for the second part, for a closed PS-curve with n points, as shown in Fig. 21(a), suppose the local profile from P_l to P_{l+m-1} will be updated under a big offset distance, as in Fig. 21(b). To make it easier, the points of this local profile are renamed with Q_1 to Q_m, as shown in Table 1. When the local profile updating process goes, the local profile is updated.

Initially, $N_{\text{point}} = m$, in which N_{point} is the total point number of the local profile, as shown in Table 1. According to the basic profile updating rules, the adjacent bisectors of Q_i and Q_{i+1} are continuously checked for intersections. If they do not intersect, then $i = i + 1$; otherwise, the two points Q_i and Q_{i+1} are replaced with a new point, which means $N_{\text{point}} = m-1$, meanwhile $i = i-2$.

As in Table 1, in the first step, i starts forwards from 1. Suppose at $i = p_1$, the two bisectors intersect, then i goes back until at $i = q_1$, the two bisectors do not intersect. In this process, $p_1 + (p_1 - q_1)$ times bisectors intersection checking is done. N_{point} becomes $m - (p_1 - q_1)$.

In the second step, i starts forwards from q_1. Suppose at $i = p_2$, the two bisectors intersect, then i goes back until at $i = q_2$, the two bisectors do not intersect. In this process, $(p_2 - q_1) + (p_2 - q_2)$ times bisectors intersection checking is done. N_{point} becomes $m - (p_1 - q_1) - (p_2 - q_2)$.

Then in the kth step, i starts forwards from q_{k-1}. suppose at $i = p_k$, the two bisectors intersect, then i goes back until at $i = q_k$, the two bisectors do not intersect. In this process, $(p_k - q_{k-1}) + (p_k - q_k)$ times bisectors intersection checking is done. N_{point} becomes $m - \Sigma(p_i - q_i)$, in which i is from 1 to k.

Suppose at this time, the local profile updating process is completed, which means $N_{\text{point}} = 0$, so:

$$\sum_{1}^{k} (\boldsymbol{p}_i - \boldsymbol{q}_i) = \boldsymbol{m}. \tag{3}$$

And substituting Eq. (3) into Eq. (4), then:

$$\begin{aligned} N_{\text{time}} &= p_1 + (p_1 - q_1) + (p_2 - q_1) + (p_2 - q_2) \\ &\quad + \cdots + (p_k - q_{k-1}) + (p_k - q_k) \\ &= p_1 + (p_2 - q_1) + (p_3 - q_2) + \cdots + (p_k - q_{k-1}) + m \\ &= (p_1 - q_1) + (p_2 - q_2) + \cdots + (p_{k-1} - q_{k-1}) \\ &\quad + (p_k - q_k) + q_k \\ &= 2m + q_k, \end{aligned} \tag{4}$$

in which N_{time} is the total bisectors intersection checking times.

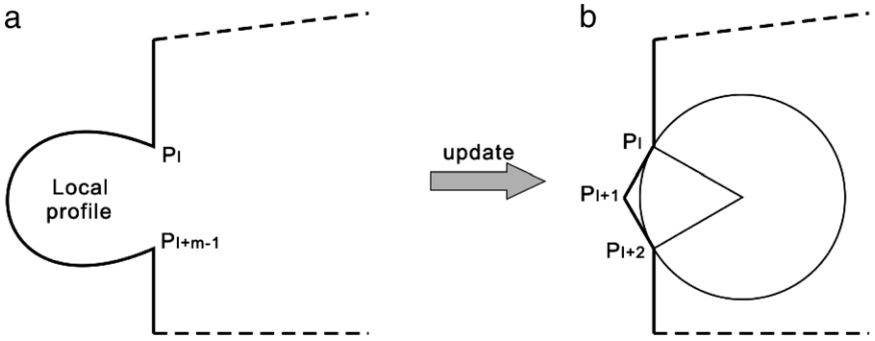

Fig. 21. A closed PS-curve with n points; (a) the local profile with m points; (b) the local profile is updated under a big offset distance with stuck circle.

Table 1
Time complexity analysis of profile updating process.

Step	Profile updating process	Left point number in the local profile (N_{point})	Adjacent intersection checking times (N_{time})
0	Q_1 Q_i Q_{i+1} Q_m	m	0
1	Q_1 Q_{q_1} Q_{p_1} Q_m	$m-(p_1-q_1)$	$p_1+(p_1-q_1)$
2	Q_1 Q_{q_2} Q_{q_1} Q_{p_2} $Q_{m-(p_1-q_1)}$	$m-(p_1-q_1)-(p_2-q_2)$	$p_1+(p_1-q_1)+(p_2-q_1)+(p_2-q_2)$
k	Q_1 Q_{q_k} $Q_{q_{k-1}}$ Q_{p_k} $Q_{m-\sum_i^{k-1}(p_i-q_i)}$	$m-\Sigma(p_i-q_i)$, i from 1 to k	$p_1+(p_1-q_1)+\cdots+(p_k-q_{k-1})+(p_k-q_k)$

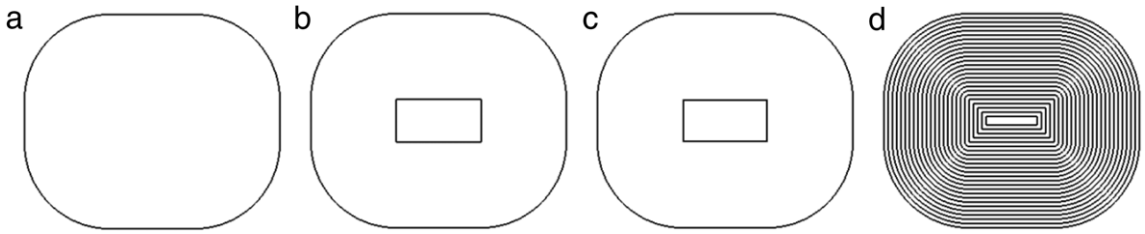

Fig. 22. Near circular portion offsets; (a) rectangle with rounded corners, radii 20 mm; (b) single offset for 20 mm; (c) single offset for 20.1 mm; (d) full offset at 1 mm.

In the last step (kth step), $q_k = 0$, so $N_{\text{time}} = 2m$. So for a local profile with m points, it can be updated in $2m$ times if only the basic profile updating rule is applied.

In the append profile updating rule, one point is split into two points, and then the basic rule is applied to the split points. So the worst case is: for the $2m$ points, each point is split into two points, and then the basic rule is applied to the $2m$ points. In this case, the local profile can be updated in $4m$ times of adjacent bisectors intersection checking.

Above all, for a local profile with m points, it can be updated in at most $4m$ times of adjacent bisectors intersection checking. And for a closed PS-curve with n points, it can be updated in at most $4n$ times of adjacent bisectors intersection checking. So the time complexity for the raw offset curve generation process is $O(n)$.

On the whole, the whole offset algorithm, including its three parts, can be completed in $O(n \log n)$ time, in which n is the point number in all the input PS-curves.

6. Experiments

The proposed offset algorithm has been programmed in the C++ language and run on a personal computer with Pentium (R) Dual-core CPU E6300 @ 2.80 GHz and 2.00 GB RAM. Our algorithm succeeds for many examples that have been tested.

It has been mentioned many times in the literature that traditional offset algorithms have numerical instability near the circular portion. However, our algorithm can deal well with such a problem both in theory (see Section 3) and in practice. Fig. 22(a) shows a rectangle with rounded corners. The radii of the corners are 20 mm. Fig. 22(b) offsets the original profile for 20 mm; it can be seen that the offset curve turns into a rectangle just as it should be. Fig. 22(c) gives a single offset at 20.1 mm; the offset curve is also a rectangle as it should be. Fig. 22(d) shows full offset curves at 1 mm, and the results are good.

Fig. 23 demonstrates the whole process of the proposed offset algorithm on a base face of a 2.5D pocket with 15 islands which was taken from Ref. [21]. Originally, in that paper, it contains 14 trapezoidal, rectangular or circular islands, which are all convex. In this paper, however, we make it more complex by adding a non-convex random shape island. As shown in Fig. 23(a), the point number of the outer profile is about 500; the point number of each circular or elliptical island is 100; and the point number of the random shape is 500. Totally, the point number in Fig. 23(a) is 2220. Fig. 23(b) shows the Delaunay triangulation of all points in Fig. 23(a), with unnecessary triangles removed. The Delaunay triangulation process takes about 25 ms for this example. In Fig. 23(c), all the islands are connected in a serial to the outer profile by the bridges. The bridge construction and curve connection process takes about 4 ms for this case. And Fig. 23(d) is the single offset result of the updated profile and Fig. 23(e) is the full offset result, which can be used as a contour-parallel tool path if these curves are properly linked.

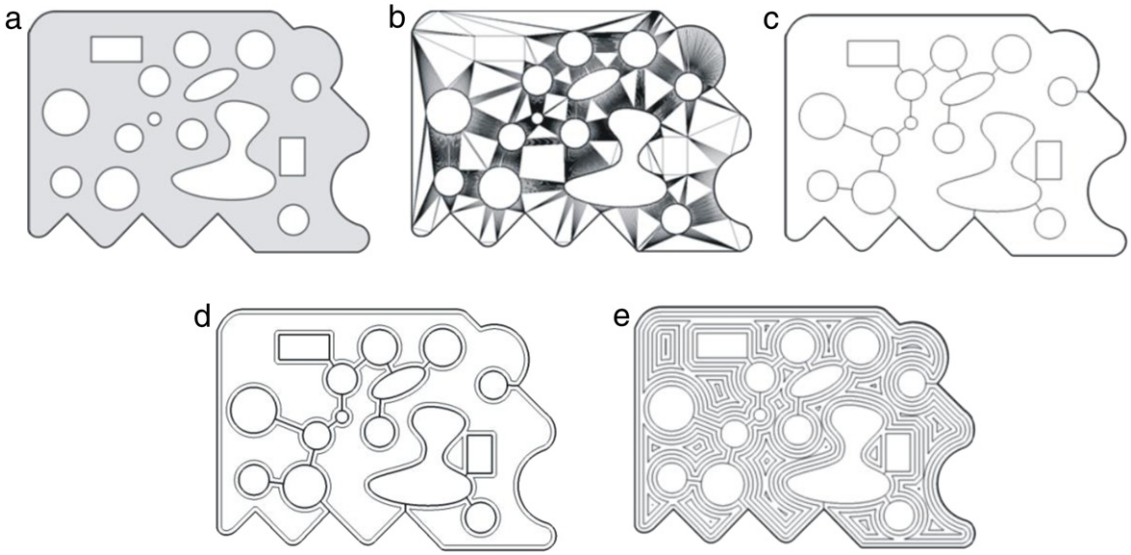

Fig. 23. Demonstration of the proposed algorithm on a pocket profile; (a) the base face of a 2.5D pocket with 15 islands; (b) Delaunay triangulation of the input PS-curves; (c) islands are bridged to the outer profile; (d) single offset of the bridged PS-curve; e. full offset of the bridged PS-curve.

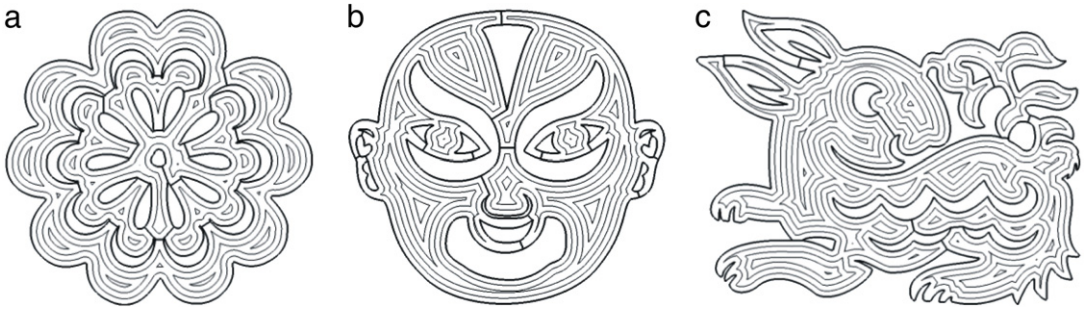

Fig. 24. Three more examples; (a) flower profile with 21 islands; (b) Peking Opera mask profile with 16 islands; (c) rabbit profile with 10 islands.

Table 2
Executing time (ms) of different parts of the proposed offset algorithm.

Profiles	Point number	Bridging	Single offset	Bridging & single offset	Full offset
Pocket	2,220	29.064	6.558	36.268	21.628
Flower	6,019	108.689	23.944	123.042	73.223
Mask	10,000	193.836	54.971	244.106	83.489
Rabbit	13,777	295.743	70.925	366.768	131.135

Together with Fig. 23, three more examples are given in Fig. 24(a)–(c) to verify the proposed algorithm. Fig. 24(a) is a flower profile with 21 islands; Fig. 24(b) is a Peking Opera mask profile with 16 islands and Fig. 24(c) is a rabbit profile with 10 islands. The executing time statistics, including the bridging time, single offset time, bridging plus single offset time, and full offset time, of each profile are shown in Table 2. In Table 2, the point number is the total point number of all input outer profiles and islands. And Fig. 25 gives the plotted result of Table 2. For these examples, the bridging process takes more time than the offset time (single offset) because of the Delaunay triangulation in the bridging process. On the whole, from Fig. 25, it can be seen that the proposed algorithm has a linear time complexity.

After the islands have bridged to the outer profile, in a single offset process, there are two operations: the raw offset curve generation and global invalid loops removal. We may also want to know the executing time of these two parts. Table 3 gives this kind of time statistics with the profile showing in Fig. 26, at different offset distances, and the results are plotted in Fig. 27. From Fig. 27, it can be seen that with the increase of offset distance, the time of raw offset increases while the time of global invalid loop removal decreases. This is understandable: the bigger the offset distance is,

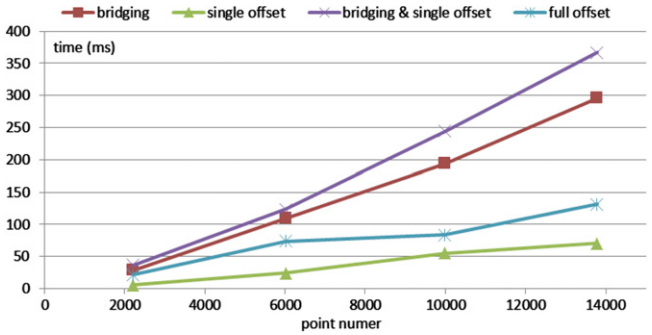

Fig. 25. Plotted result of Table 2.

the more the local problems must be fixed. But the bigger the offset distance is, the fewer the points there are in the raw offset curve, so it must be quicker to report self-intersections and remove global invalid loops. On the whole, from Fig. 20, the added time (a single offset) changes very little.

We have also compared our algorithm with that of some commercial software packages on the same PC, also using the profile

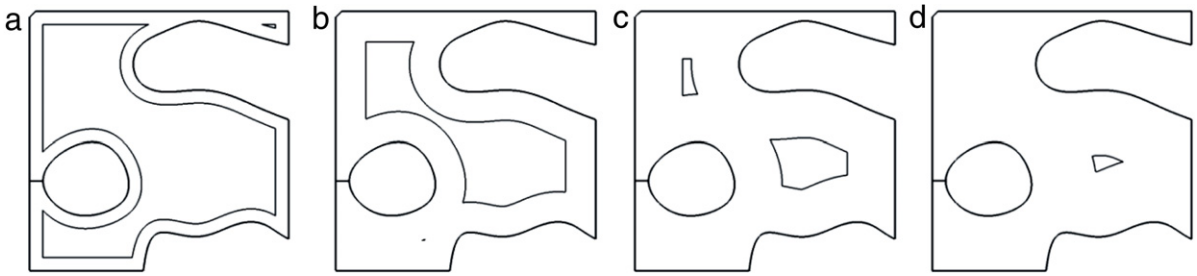

Fig. 26. Different offset distance on a PS-curve with 10,376 points; (a) offset at 3 mm; (b) offset at 7 mm; (c) offset at 11 mm; (d) offset at 15 mm.

Table 3
Executing time (ms) for a single offset with different offset distances, with 10,376 points.

Offset distance	Single offset	Raw offset curve generation	Global invalid loop removal
1	34.093	14.429	15.846
3	39.031	20.048	14.097
5	49.323	24.888	19.739
7	44.477	27.942	12.159
9	47.273	34.364	7.639
11	47.525	36.633	5.923
13	48.648	41.721	2.276
15	49.117	43.227	1.405

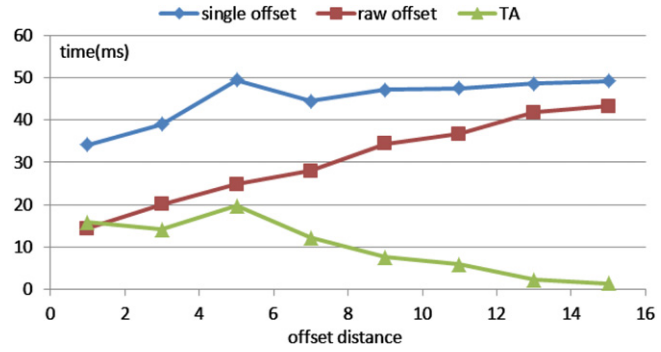

Fig. 27. Plotted results of Table 2.

as in Fig. 26, with 10,376 points. The results are shown in Table 4. Technically, it is not possible to measure the precise executing time of commercial software, a stopwatch is used to measure the time roughly. Some software packages can only do single offset, like AutoCAD 2004, Adobe Illustrator CS5 (AI CS5 for short) and FreeHand MX, so the single offsetting time is measured. UG NX7.0 (CAD module) cannot offset the profile in Fig. 26, with 10,376 points. SolidWorks can fix local problems but cannot remove global invalid loops. AI CS5 offsets fast, but the offset PS-curve has local problems. FreeHand MX offsets slowly. Among the tested software packages, CorelDRAW 12 is the most outstanding. It works very fast. However, for this case, as in Table 4, our program is even faster than CorelDRAW 12.

The last example depicted in Fig. 28 is a layer by layer roughing process simulation in VERICUT. The designed model is a Mickey

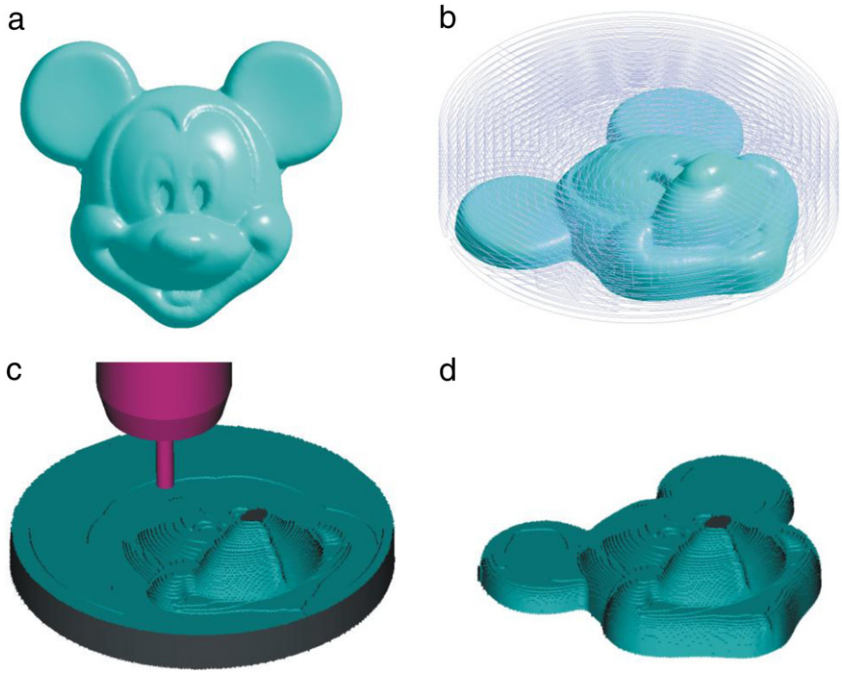

Fig. 28. A cutting simulation on VERICUT; (a) the STL-format Mickey Mouse; (b) contour-parallel tool path generated by our program; (c) cutting simulation in VERICUT; (d) the finished roughing stock.

Table 4
Executing time (s) comparison between commercial software packages and our program, with 10,376 points.

Software	Single offset at 4 mm (s)	Full offset at 0.025 mm (s)
AutoCAD 2004	5.50	
CorelDRAW 12	<1	16.72
AI CS5	<1	
FreeHand MX	37.47	
Our program	0.04	8.95

Mouse in STL format. The mouse has 81,944 triangle facets and is about $80\times70\times25$ mm in size. On each layer, a contour-parallel tool path is generated based on the proposed offset algorithm in this paper and is linked with a linking algorithm proposed in Ref. [13]. To make it safe and simple, when the cutter reaches the end of the tool path on each layer, it retracts to a safe level; and then moves down to the next layer. In this example, a flat end mill is used. The diameter of the cutter is 4 mm. On each layer, the tool path interval is 2 mm, so that no uncut area will be left. And between two adjacent layers, the slice step distance is 0.5 mm. The tool path generated by our program is shown in Fig. 28(b). The tool path is transferred to standard NC blocks, in which most are G01. Then the NC file is imported into VERICUT for simulation. Fig. 28(c) shows a screenshot of the machining process in VERICUT. And Fig. 28(d) shows the finished stock after the roughing process. In Fig. 28(d), obvious step stairs can be observed; to get a smooth Mickey Mouse as in Fig. 28(a), further research has to be done on finishing [22,23] and polishing [24] processes.

7. Conclusions and future work

A new PS-curve offset algorithm is proposed in this paper. The algorithm is able to deal with PS-curves with multiple islands. The input of the algorithm is a set of PS-curves, in which one is the outer profile and the others are islands. Firstly, the islands are automatically bridged to the outer profile with the Delaunay triangulation method, forming a single linked PS-curve. The profile of this curve is then updated with the profile updating rules, the basic rule and the append rule, so that a raw offset curve without any local problems can be generated. After that, the raw offset curve is treated with the TA procedure to remove all invalid global invalid loops. The output of the algorithm is the offset PS-curves that can be used for a contour-parallel tool path. The time complexity of each part is analyzed and on the whole, the proposed algorithm can be completed in $O(n \log n)$ time, in which n is the total point number of input PS-curves.

From a geometrical point of view, the concept of a stuck circle used in this paper might have the potential to be popularized to the 3D case. The best guess might be using a stuck sphere to fix local self-intersections in 3D tessellated surface offset. And in a machining point of view, many problems need to be solved in the future. For example, in the bridge process, bridges are straight lines. This may burden the machine tool where the joint of the bridge and the profile has sharp angles. This problem might be optimized by using curved bridges, because curved bridges can connect two profiles smoothly. And in this paper, when generating the contour-parallel tool path, the machining efficiency is not considered. Theoretically, the highest machining efficiency is reached when the tool path interval equals the cutter diameter. But this will leave uncut areas between two adjacent tool paths. This problem might be solved by firstly locating uncut areas and then finding an optimized path that goes through all these areas with a minimum path length.

Acknowledgment

This work was financially supported by the National Natural Science Foundation of China (51175461).

Appendix. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cad.2012.09.002.

References

[1] Lasemi A, Xue D, Gu P. Recent development in CNC machining of freeform surfaces: a state-of-the-art review. Computer-Aided Design 2010;42:641–54.
[2] D'Souza RM, Sequin C, Wright PK. Automated tool sequence selection for 3-axis machining of free-form pockets. Computer-Aided Design 2004;36:595–605.
[3] Balasubramaniam M, Joshi Y, Engels D, Sarma S, Shaikh Z. Tool selection in three-axis rough machining. International Journal of Production Research 2001;39:4215–38.
[4] Kim H-C. Tool path generation for contour parallel milling with incomplete mesh model. The International Journal of Advanced Manufacturing Technology 2010;48:443–54.
[5] Hatna A, Grieve RJ, Broomhead P. Automatic CNC milling of pockets: geometric and technological issues. Computer Integrated Manufacturing Systems 1998;11:309–30.
[6] Persson H. NC machining of arbitrarily shaped pockets. Computer-Aided Design 1978;10:169–74.
[7] Held M, Lukács G, Andor L. Pocket machining based on contour-parallel tool paths generated by means of proximity maps. Computer-Aided Design 1994;26:189–203.
[8] Choi BK, Park SC. A pair-wise offset algorithm for 2D point-sequence curve. Computer-Aided Design 1999;31:735–45.
[9] Kimmel R, Bruckstein AM. Shape offsets via level sets. Computer-Aided Design 1993;25:154–62.
[10] Kim K, Jeong J. Tool path generation for machining free-form pockets with islands. Computers & Industrial Engineering 1995;28:399–407.
[11] Wong TN, Wong KW. NC toolpath generation for arbitrary pockets with islands. The International Journal of Advanced Manufacturing Technology 1996;12:174–9.
[12] Park SC, Choi BK. Uncut free pocketing tool-paths generation using pair-wise offset algorithm. Computer-Aided Design 2001;33:739–46.
[13] Park SC, Chung YC. Offset tool-path linking for pocket machining. Computer-Aided Design 2002;34:299–308.
[14] Lai Y-L, Wu JS-S, Hung J-P, Chen J-H. A simple method for invalid loops removal of planar offset curves. The International Journal of Advanced Manufacturing Technology 2006;27:1153–62.
[15] Kim H-C, Lee S-G, Yang M-Y. A new offset algorithm for closed 2D lines with islands. International Journal of Advanced Manufacturing Technology 2006;29:1169–77.
[16] Lee C-S, Phan T-T, Kim D-S. 2D curve offset algorithm for pockets with islands using a vertex offset. International Journal of Precision Engineering and Manufacturing 2009;10:127–35.
[17] Liu X-Z, Yong J-H, Zheng G-Q, Sun J-G. An offset algorithm for polyline curves. Computers in Industry 2007;58:240–54.
[18] Dragomatz D, Mann S. A classified bibliography of literature on NC milling path generation. Computer-Aided Design 1997;29:239–47.
[19] Berg Md, Cheong O, Kreveld Mv, Overmars M. Computational geometry: algorithms and applications. 2008.
[20] Bentley JL, Ottmann TA. Algorithms for reporting and counting geometric intersections. Computers, IEEE Transactions on 1979;C-28:643–7.
[21] Hinduja S, Mansor MSA, Owodunni OO. Voronoi-diagram-based linking of contour-parallel tool paths for two-and-a-half-dimensional closed-pocket machining. Proceedings of the Institution of Mechanical Engineers. Part B: Journal of Engineering Manufacture 2010;224:1329–50.
[22] López de Lacalle LN, Lamikiz A, Sánchez JA, Salgado MA. Toolpath selection based on the minimum deflection cutting forces in the programming of complex surfaces milling. International Journal of Machine Tools and Manufacture 2007;47:388–400.
[23] Yuwen S, Dongming G, Zhenyuan J, Haixia W. Iso-parametric tool path generation from triangular meshes for free-form surface machining. The International Journal of Advanced Manufacturing Technology 2006;28:721–6.
[24] López de Lacalle LN, Rodríguez A, Lamikiz A, Celaya A, Alberdi R. Five-axis machining and burnishing of complex parts for the improvement of surface roughness. Materials and Manufacturing Processes 2011;26:997–1003.