Int J Adv Manuf Technol (2006) 29: 1169–1177
DOI 10.1007/s00170-005-0013-1

ORIGINAL ARTICLE

Hyun-Chul Kim · Sung-Gun Lee · Min-Yang Yang

A new offset algorithm for closed 2D lines with Islands

Received: 10 November 2004 / Accepted: 17 February 2005 / Published online: 15 September 2005

Abstract In this paper, a new offset algorithm for closed 2D lines with islands is introduced and the result is illustrated. The main point of the proposed algorithm is that every point is set to be an offset using bisectors, and then invalid offset lines, which are not to be participated in offsets, are detected in advance and handled with an invalid offset edge handling algorithm in order to generate raw offset lines without local invalid loops. As a result, the proposed offset method is proved to be robust and simple, moreover, has a near $O(n)$ time complexity, where n denotes the number of input lines. The proposed algorithm has been implemented and tested with 2D lines of various shapes.

Keywords Offset · Bisector · Invalid offset edge · Invalid offset edge handling algorithm

1 Introduction

The majority of CNC milling tasks can be performed using 2.5D milling. This is partially due to the fact that a surprisingly large number of mechanical parts are 2.5D and even more complicated objects are usually produced from billet by a 2.5D roughing and 3D–5D finishing [1]. Hence, the computation of a tool path for 2.5D milling tasks, or pocket machining, is one of the most important issues in CAM.

A popular tool path style in pocket machining is called contour-parallel path, which is generated by successive offsets of the input boundary. In addition, commercialized CAM software systems are using polyhedral machining because polyhedral data based NC tool path generation is superior to surface based tool path generation [2]. Thus offsetting closed lines is essential to generate polyhedral data based contour-parallel tool path, and the problem has widely been studied, mostly as a pocket machining problem [3–17]. The literatures on pocket machining algorithms can be roughly divided into two different approaches when finding the successive offsets of the boundary: pair wise intersection and voronoi diagram. The first approach for finding successive offsets uses pair wise intersections of individual offset segments to trim them into an offset boundary. As described by Bruckner [4], this task is executed by the following three steps, as shown in Fig. 1: (1) for every contour element, an elementary offset element is constructed, (2) gaps between the offset elements are closed by joining arcs – this results in some closed curves, and (3) self-intersections of the curves are eliminated and portions of the curves are cleared away yielding the final offset curves. The basic geometric entities related to the pair wise intersection method are shown in Fig. 1b. As shown in Fig. 1b, there are two types of invalid loops: local invalid loop bounded by a single self intersection point and global invalid loop bounded by a pair of self intersection points. The main problem with this approach is that determination of all self-intersections and eliminattion of local invalid loops after an offset are time consuming processes leading to a $O(n^2)$ calculation complexity [13] and prone to numerical errors, as pointed out by Held et al. [11]. The second approach, described by Persson [3], uses offset curve zones based on the voronoi diagram. Using this approach, when the offset curve zones are known, offsets can be accomplished by simply connecting the offset segments in each zone. Although the offset approach using a voronoi diagram is more robust and more widely used than pair wise offset approach, constructing a voronoi diagram is not a trivial and time consuming process. It also has numerical instability [18].

A few algorithms have been proposed to cope with the above mentioned difficulties, but traditional offset approaches still have drawbacks because the removal of local invalid loops has a numerical instability inherent in a near circular portion when many short line segments, which are obtained by slicing a polyhedral model with a horizontal plane, are offset in a corner [19].

H.-C. Kim (✉) · S.-G. Lee · M.-Y. Yang
Department of Mechanical Engineering,
Korea Advanced Institute of Science and Technology,
305-701 Daejeon, South Korea
e-mail: michael77@kaist.ac.kr
Tel.: +82-42-8693264
Fax: +82-42-8693210

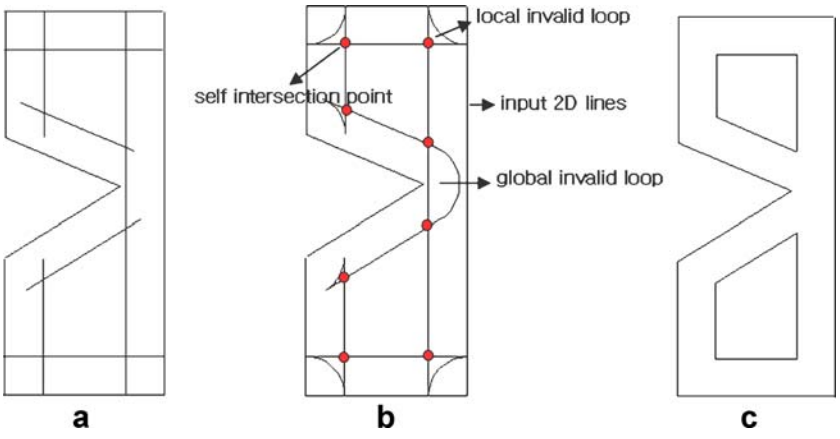


Fig. 1 Pair wise intersection algorithm. (**a**) elementary offset. (**b**) closed curves. (**c**) final offset curve

In this paper, a new offset algorithm using bisectors and an invalid offset edge handling algorithm (to be elaborated later) are presented. This offset algorithm detects the lines not participated in an offset beforehand using bisectors. These lines are handled with a special algorithm, an invalid offset edge handing algorithm, explained later in order to obtain raw offset lines without local invalid loops. The offset procedure consists of the following steps:

1. Calculating offset points using bisectors and generating offset lines.
2. Checking the validity of generated offset lines and handling invalid offset lines.
3. Eliminating global invalid loops.
4. Merging generated boundaries and islands.

The remainder of this paper will be presented in the following order. In Sects. 2 and 3, an algorithm, which is the most important part in this paper, to generate raw offset lines without local invalid loops is presented. Global loops processing and merging algorithms are described in Sects. 4, 5, and in Sect. 6, various examples are shown. The final section contains discussions and concluding remarks.

2 Computation of offset points using bisectors

The boundary profile is in the counterclockwise direction when viewed from above and clockwise direction for island profile(s), as in Fig. 2. Each line has a parameter range defined over $t\varepsilon[0,1]$.

Before continuing, we introduce here some definitions that appear throughout this paper. Noting that closed lines make a polygon, the term 'vertex' is used to denote a point in the line and 'edge' is used for a line segment joining two consecutive vertices.

Definition 1 A vertex ν is called *convex* if the internal angle between the edges incident upon ν is $>\pi$, *parallel* if the internal angle is equal to π, and *concave* otherwise.

Definition 2 An offset edge also has a direction. If the direction of an offset edge is reversed after an offset, the offset edge is called invalid and valid otherwise.

Definition 3 The valid offset edge, first detected by an opposite direction from any invalid offset edge, is called a backward edge and the one by a same direction is called a forward edge.

Computing an offset point of a vertex is according to vertex types above defined, and in every case, offset points are placed on bisectors, or dashed lines in Fig. 3. In Fig. 3, $P_i(i=0,1,...)$ are vertexes, and $Q_i(i=0,1,...)$ are the offset points of P_i. In the case of Fig. 3a, where P_i is *concave*, the offset point Q_i is generated by

$$Q_i = P_i + \frac{e_{i+1} - e_i}{|e_{i+1} - e_i|} \cdot \frac{R}{\sin\left(\theta_i/2\right)} \tag{1}$$

where $e_i = \frac{P_i - P_{i-1}}{|P_i - P_{i-1}|}$, R is the offset distance, and θ_i is the angle between $P_{i-1}P_i$ and $P_{i+1}P_i$.In Fig. 3b, P_i is *convex*, and the offset point Q_i is generated by

$$Q_i = P_i + \frac{e_i - e_{i+1}}{|e_i - e_{i+1}|} \cdot \frac{R}{\cos\left(\frac{\pi-\theta_i}{2}\right)} \tag{2}$$

In Fig. 3c, the vertex P_i is *parallel*, and the offset point Q_i are generated as

$$Q_i = P_i + n \cdot R \tag{3}$$

where

$$n = \frac{Q_i - P_i}{|Q_i - P_i|}$$

3 Validity check and invalid offset edge handling algorithm

As defined in definition 2 above, Fig. 4 shows an invalid offset edge, the direction of which is reversed after an offset. That is, if $l-(l'+l'')\leq 0$ is satisfied in Fig. 4b, the edge is an invalid offset edge. Raw offset lines can be obtained

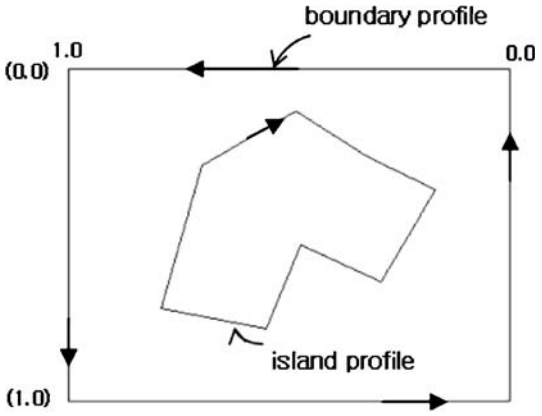


Fig. 2 Input profile

by connecting offset points in order. However, among raw offset lines, invalid offset edges should not belong to raw offset lines. Figure 5 shows raw offset lines without invalid offset edges. In that case, offset lines can not make a closed loop because a backward edge and a forward edge, detected by any invalid offset edge, are not connected. Raw offset lines are twisted becuase of a change of offset line direction as shown in Fig. 4. In case of this twist, simple connection of a forward edge and a backward edge will be processed by a loop- processing algorithm which will be explained later. However, there exists special cases in which no twist occures by invalid offset edges. For these cases, an invalid offset edge handling algorithm is required. An invalid offset edge handling algorithm is applied differently in two different cases.

Case (I) The number of invalid offset edges between a backward edge and a forward edge detected by an invalid offset edge is one.

Case (II) The number of invalid offset edges between a backward edge and a forward edge detected by an invalid offset edge is more than two.

Taking computation of offset points into account, processes described above may be expressed as follows:

```
Computation of offset points and validity check:
    //input: points[] representing closed 2D lines, offset
    distance//
    //output: offset points[], offset lines[] made by offset
    points[]//
    //offset lines.size()=offset points.size()-1//
    for(int i=0; i<points.size(); ++i)
    {
        offset points[i]=an offset point corresponding to
        points[i];
        offset lines[i-1]=a line from offset points[i-1] to
        offset points[i];
        check the validity of offset lines[i-1];
    }

Invalid offset edge handling algorithm:
    //input: offset lines[]//
    //output: raw offset lines without local invalid loops//
    for(int i=0; i<offset lines.size(); ++i)
    {
        if(offset lines[i] is an invalid offset edge)
        {
            if(an invalid offset edge is consecutive)
                Case II algorithm;
            else
                Case I algorithm;
            if(update by Case I or Case II algorithm == false)
                connect a backward edge and a forward edge
                corresponding to offset lines[i];
        }
    }
```

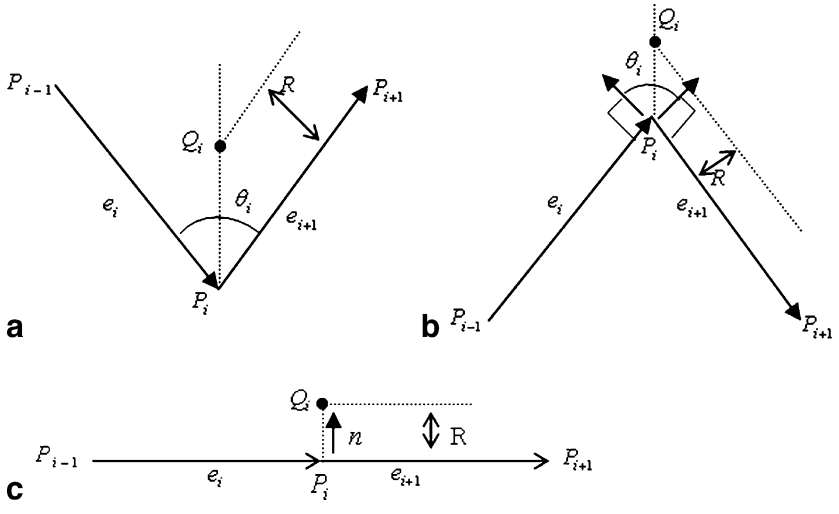


Fig. 3 Offset points according to vertex types. (**a**) concave. (**b**) convex. (**c**) parallel

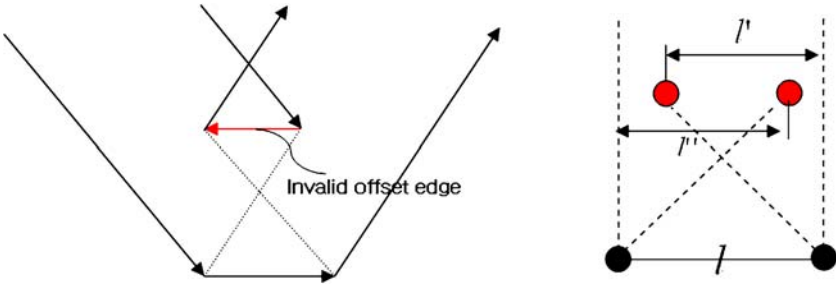


Fig. 4 Invalid offset edge

3.1 Invalid offset edge handling algorithm in Case I

In the case that the number of invalid offset edges between a backward edge and a forward edge detected by an invalid offset edge is one, that is, invalid offset edges are not consecutive, the following algorithm can be applied.

The procedure of this algorithm is as follows.

1. Extending a backward edge and a forward edge and searching an intersection of two extended lines.
2. Updating an end point of a backward edge and a start point of a forward edge with an intersection point.

In Fig. 6a, an offset line Q_2Q_3 is an invalid offset edge because an offset line Q_2Q_3 is reversed after a boundary edge P_2P_3 is offset. Therefore an invalid offset edge Q_2Q_3 must be removed. Because the number of invalid offset edges between a backward edge and a forward edge detected by an invalid offset edge Q_2Q_3 is one, an invalid offset edge handling algorithm equivalent to Case I is applied. The process is shown in Fig. 6 and the algorithm is as follows:

Invalid offset edge handling algorithm in Case II:

```
//input: offset lines[]//
if(an invalid offset edge is not consecutive)
{
    find a backward edge and a forward edge;
    calculate an intersection between them;
    update the result;
}
```

3.2 Invalid offset edge handling algorithm in Case II

In the case that invalid offset lines are consecutive, the lines are handled with following procedure.

1. Boundary edges equivalent to invalid offset edges are chosen, and chosen edges are checked whether intersected or not by valid offset edges and indexes of checked edges are memorized.
2. Boundary edges equivalent to memorized indexes are offsets by the conventional pair wise offset method.
3. Offset lines by a pair wise offset method, a backward edge, and a forward edge are modified with intersection information of these lines.

In Fig. 7a, offset lines Q_1Q_2, Q_2Q_3 are invalid offset edges. Because they are consecutive, that is, the invalid offset edges between a backward edge and a forward edge is two, an offset edge handling algorithm equivalent to Case II is applied. A boundary edge P_1P_2 among boundary edges P_1P_2, P_2P_3 equivalent to invalid offset edges Q_1Q_2,

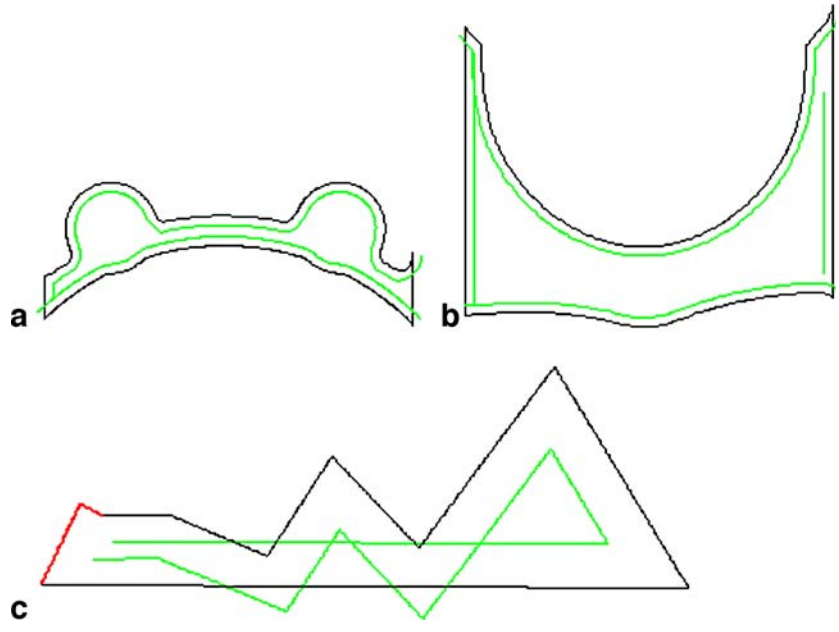


Fig. 5 Raw offset lines without invalid offset edges

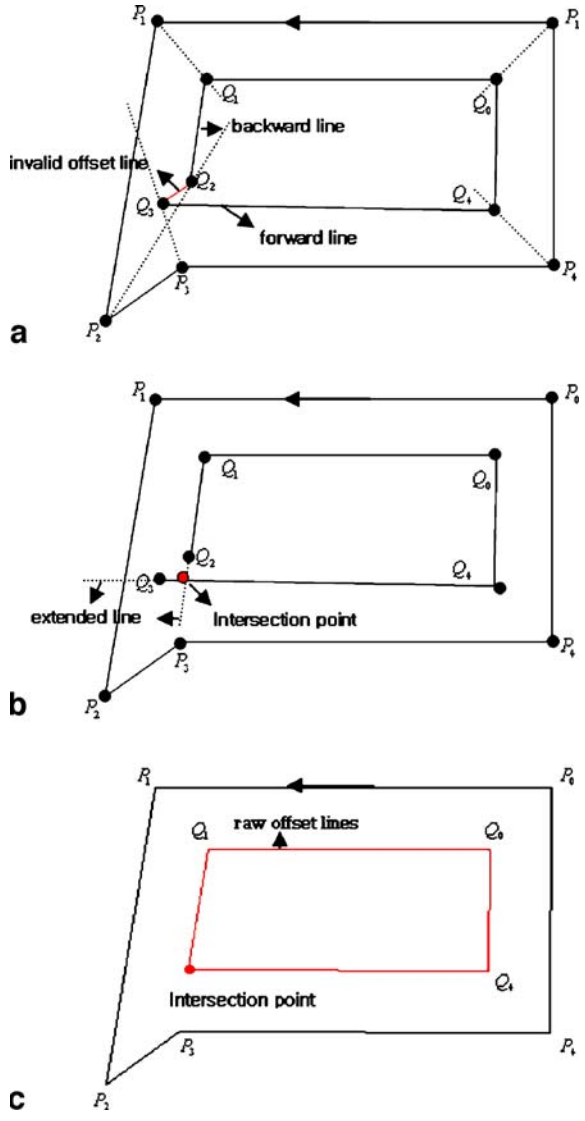


Fig. 6 Invalid offset edge handing algorithm of Case I

Q_2Q_3, respectively, has an intersection by a valid offset edge Q_3Q_4. Therefore, as shown in Fig. 7b, an additional offset line is obtained as a result of offset of the line P_1P_2 by the pair wise offset method. Then an additional offset line, a backward edge, and a forward edge are updated by intersection points of these lines. The process of Case II is illustrated in Fig. 7 and the algorithm is as follows:

Invalid offset edge handling algorithm in Case II:

```
//input: offset lines[]//
if(an invalid offset edge is consecutive)
{
    find a backward edge and forward edge;
    search boundary edges equivalent to invalid offset
    edges and get additional edges;
    calculate intersections between them;
    update the result;
}
```

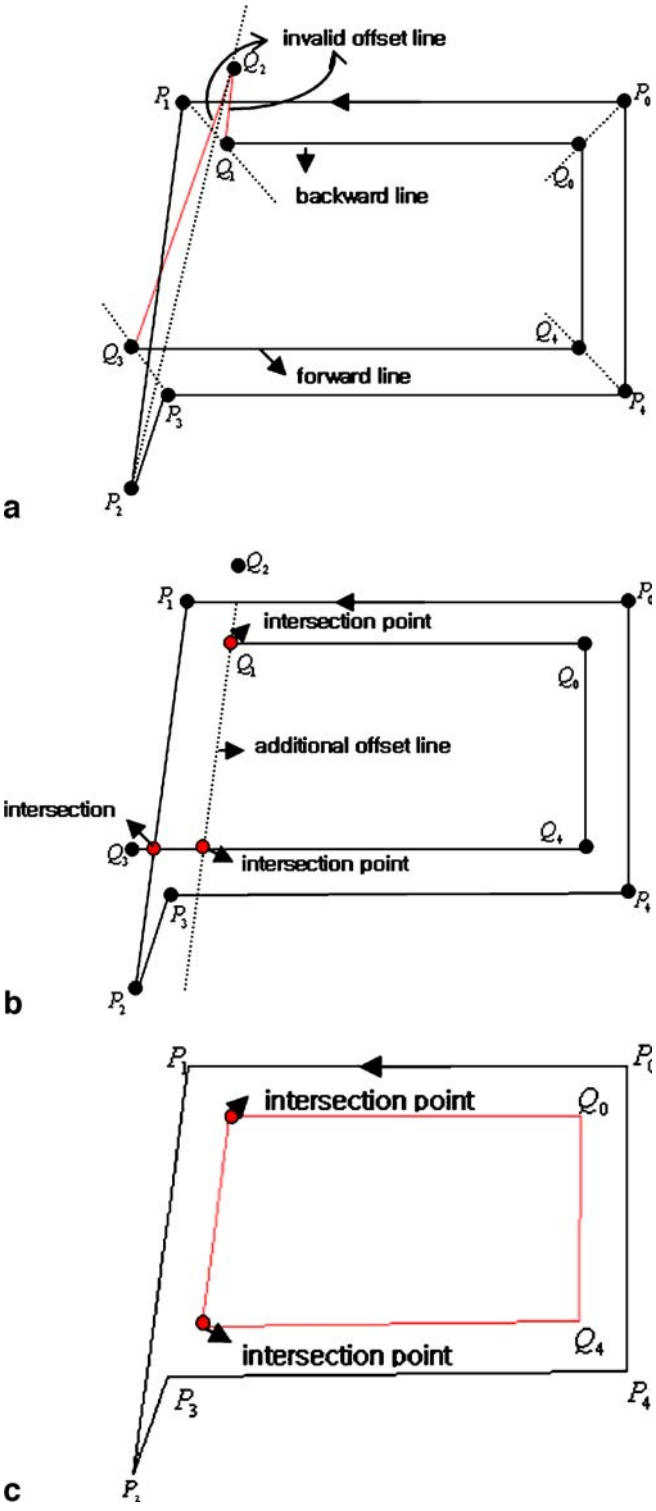


Fig. 7 Invalid offset edge handing algorithm of Case II

As stated above, offset points computed by bisectors are connected, and invalid offset edges not participated in offsets are checked in advance. As an invalid offset edge handling algorithm is applied in these invalid offset edges, raw offset lines without local invalid loops can be obtained as shown in Fig. 8.

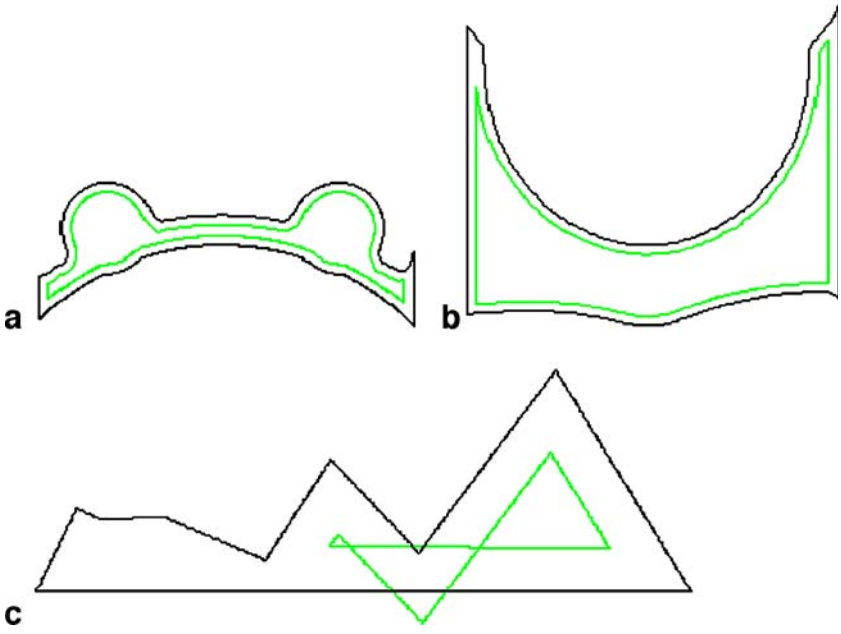


Fig. 8 Raw offset lines after an invalid offset edge handling algorithm is applied

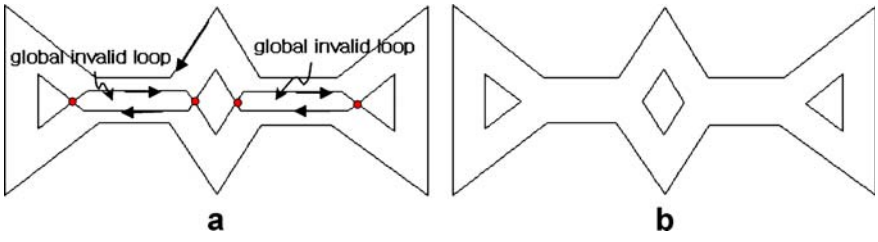


Fig. 9 Global invalid loops removal. (**a**) global invalid loops. (**b**) global invalid loops removal

4 Processing loops of offset lines

When given closed 2D lines are offsets with above stated algorithms, some global invalid loops may result, as illustrated in Fig. 9a. In this case, global invalid loops have to be removed while others are kept. A loop-processing algorithm is based upon the fact that the boundary profile was in the counterclockwise direction and the island profile in the clockwise direction before the offset procedure started. In the loop-processing algorithm, valid loops are obtained by the following tracing rule and 4 steps:

– Tracing rule: offset lines are traced along their directions from a stating point until any intersection point is encountered. At this time, the line index and line parameter are checked whether to have been visited or not. If the line index and line parameter have not been visited, they are stored in a stack as the next starting information, and the travel direction is transferred to the intersected line index and the point parameter.

1. Arbitrary chosen starting line index and starting point parameter are stored and checked for visiting.
2. Raw offset lines are traced along their directions by the tracing rule, which is explained above. While tracing, tracing line index and point parameter are checked whether starting line index and starting point parameter are equal to them or not. When tracing line index and point parameter are equal to starting line index and starting point parameter, all tracing lines are returned.
3. According to above procedure a closed loop forms and the direction of it is then checked. If the direction of an

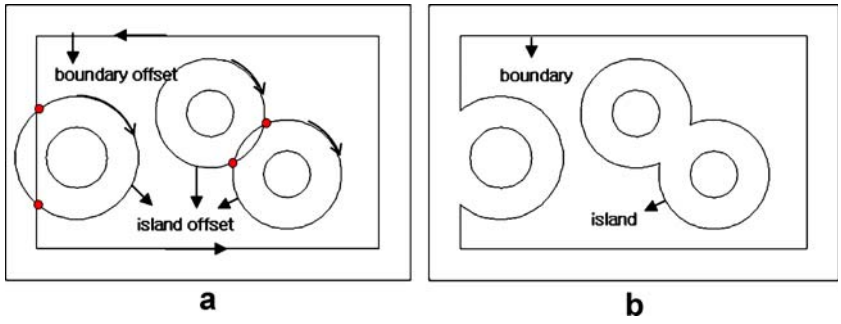


Fig. 10 Merging of a boundary and islands. (**a**) raw offset lines of a boundary and islands. (**b**) merging

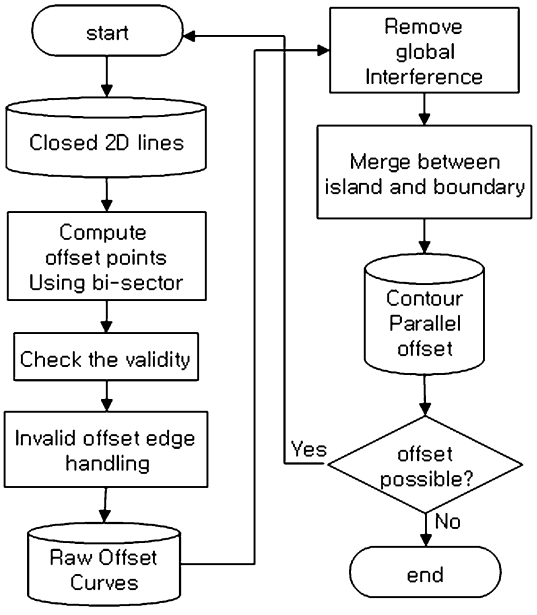


Fig. 11 The process of new offset approach

obtained loop is the same as a given profile's, the loop is valid, otherwise invalid. Then, the invalid loop is removed.

4. If the stack is not empty, top information of the stack is popped as new starting information. Steps1−4 procedure are repeated until stack becomes empty.

Direction checking for a loop can be easily done using the simple-closed-path theorem. It states that simple polygons always appear to have total turning equal +360° or −360°, depending on the direction in which we traverse the path. The result that invalid offset loops are removed using a loop-processing algorithm is shown in Fig. 9b.

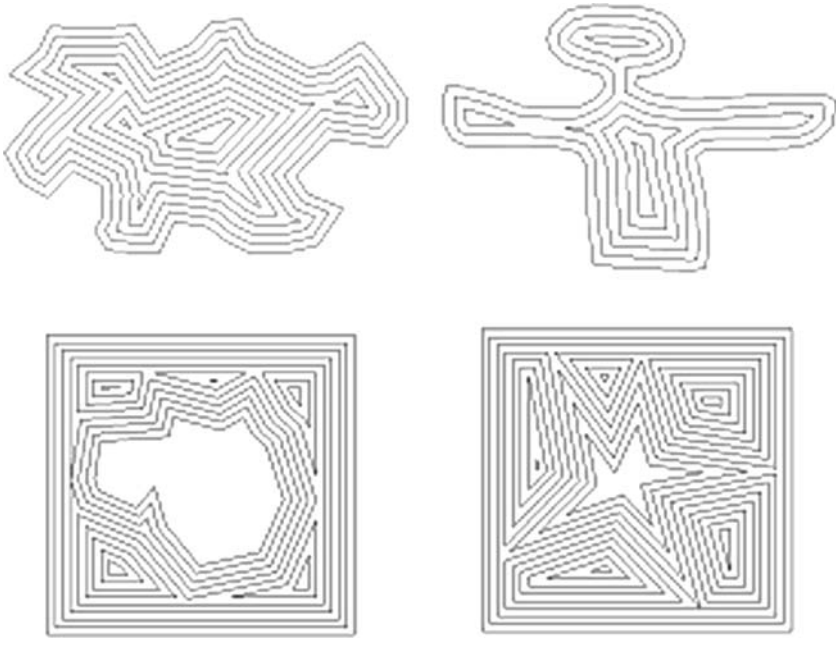
5 Merging boundaries and islands

Finally, the operation of merging boundaries and islands, which do not have local invalid loops and global invalid loops, is required. This method is very similar to a loop-processing algorithm. Offset lines are traced along their directions starting from any arbitrary point until any intersection point is encountered. At an intersection point, the travel direction is changed by the tracing rule explained in Sect. 4. This procedure is repeated until every point has been visited. After this grouping procedure has been performed, every loop is treated as a separate profile. The method to determine whether each loop is a boundary or not, is as follows. If there is any boundary line among traced lines, the loop is boundary, otherwise island. An island must be included in a boundary. The decision whehter to be included in any boundary is by using a bounding box. This merging process is illustrated in Fig. 10, and each boundary including islands is used for a next offset as an input profile.

6 Offset algorithm process and its time-complexity

An overall flowchart of a new offset algorithm is presented in Fig. 11. Initially, closed 2D lines are given as a closed sequence of 2D points, where each point is regarded as a vertex. According to each vertex type, offset points are calculated by using bisectors and the validity of offset lines obtained by offset points is checked. After that, an invalid

Fig. 12 Offset examples



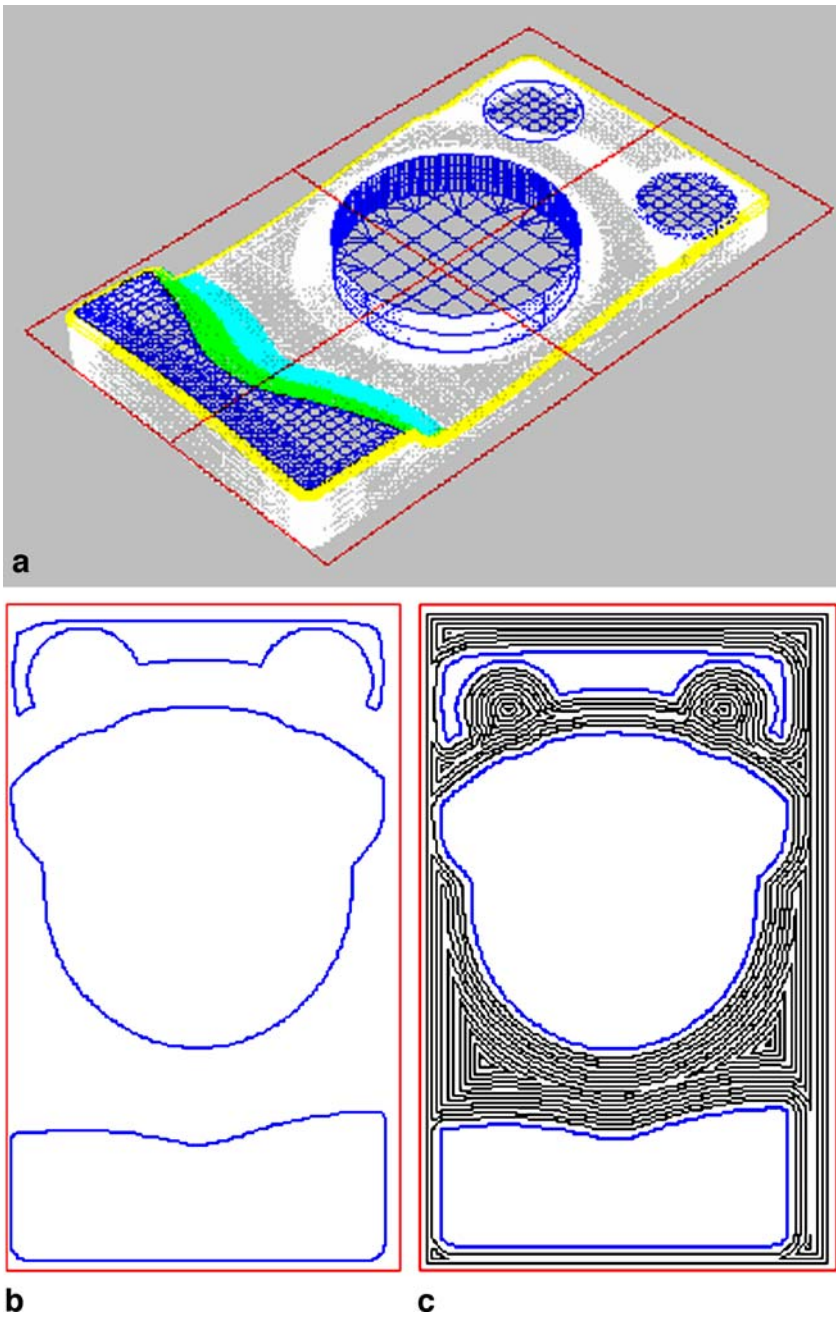


Fig. 13 Offset of a speaker. (**a**) polyhedral model. (**b**) lines generated by slicing. (**c**) offset

offset edge handling algorithm is applied to checked invalid offset lines. As a result, raw offset lines without local invalid loops are obtained. As mentioned processes are the most important part in this paper, detail algorithms were explained in Sect. 3. The time complexity of generating raw offset lines is near $O(n)$, where n is the number of lines, because all lines are visited only one time when offset points are computed by using bisectors and detected invalid offset edges are handled with constant time complexity. One of the highlighted features is that problems regarding numerical instability and time complexity of previous researches are solved. That is, time complexity for removing local invalid loops is reduced, and numerical stability is guaranteed.

The loop-processing algorithm and merging algorithm described in Sects. 4 and 5 also have $O(n)$ calculation complexity since all lines are visited only once if all self-intersections are found in advance in the raw offset lines. Finding self-intersections in the raw offset lines utilizes the gird method having time complexity of $O(n \log n)$ [20].

7 Examples

The proposed offset algorithm of 2D closed lines was implemented with C++ language on a PC. Figure 12 shows offset results of various shapes with or without islands.

Figure 13 shows an offset result of 2D lines obtained in a polyhedral model of a speaker. Shown in Fig. 13a is a polyhedral CAD model of a speaker, and 2D lines in Fig. 13b were obtained by intersecting the speaker with a horizontal plane. The number of 2D lines is 2069. The offset result generated by the proposed offset algorithm with an offset distance of 3 mm is shown in Fig. 13c.

8 Conclusion

This paper presents a new offset algorithm for closed 2D lines with islands. With bisectors, invalid offset lines are found in advance, and these lines are handled with an invalid offset edge handling algorithm proposed in this paper. As a result, raw offset lines without local invalid loops are obtained. The time complexity of generating raw offset lines is near $O(n)$, where n is the number of lines. This gives a solution for instability and time consuming problems that could be occurred in the conventional pair wise intersection approach in case of detecting and eliminating local invalid loops.

The proposed algorithm in this paper is numerically stable and simpler than the offset approach using a voronoi diagram which also has numerical instability, and there is no necessity for doing supplementary operation like constructing a voronoi diagram. In addition, the proposed algorithm is very fast and proved by offset examples of various shapes.

References

1. Held M (1991) On the computational geometry of pocket machining. Springer, Berlin Heidelberg New York
2. Jun CS, Kim DS, Park SH (2002) A new curve-based approach to polyhedral machining. Comput Aided Des 34(5):379−389
3. Persson H (1978) NC machining of arbitrary shaped pockets. Comput Aided Des 10(3):169−174
4. Bruckner LK (1982) Geometric algorithms for 2.5D roughing process of sculptured surfaces. In: Proc. Joint Anglo-Hungarian Seminar on Computer-Aided Geometric Design, Budapest, Hungary, October 1982
5. Preiss K (1989) Automated mill pocketing computations. Proc International Symposium on Advanced Geometric Modeling for Engineering Applications, Berlin, Germany
6. Prabhu P, Gramopadhye A, Wang H (1990) A general mathematical model for optimizing NC tool path for face milling of flat convex polygon surfaces. Int J Prod Res 28(1):101−130
7. Guyder MK (1990) Automating the optimization of 21/2 axis milling. Comput Ind 15(3):163−168
8. Suh Y, Lee K (1990) NC milling tool path generation for arbitrary pockets defined by sculptured surfaces. Comput Aided Des 22(5):273–284
9. Kramer T (1992) Pocketing milling with tool engagement detection. J Manuf Syst 11(2):114−123
10. Hansen A, Arbab F (1992) An algorithm for generating NC tool path for arbitrary shaped pockets with islands. ACM Trans Graph 11(2):152−182
11. Held M, Lukacs G, Andor L () Pocket machining based on contour-parallel tool paths generated by means of proximity maps. Comput Aided Des 26(3): 189−203
12. Kim K, Jeong J (1995) Tool path generation for machining free-form pockets with islands. Comput Ind Eng 28(2):399−407
13. Lambregts C, Delbressine F, de Vries W, van der Wolf A (1996) An efficient automatic tool path generator for 21/2D free-form pockets. Comput Ind 29(3):151−157
14. Choi BK, Kim BH (1997) Die-cavity pocketing via cutting simulation. Comput Aided Des 29(12):837−846
15. Chuang S, Lin W (1997) Tool path generation for pockets with free-form curves using Bezier convex hulls. Int J Adv Manuf Technol 13:109−115
16. Jeong J, Kim K (1999) Generating tool paths for free-form pocket machining using z-buffer-based voronoi diagrams. Int J Adv Manuf Technol 15(3):182−187
17. Jeong J, Kim K (1999) Generation of tool paths for machining free-from pockets with islands using distance maps. Int J Adv Manuf Technol 15(5):311−316
18. Kokichi S (1998) Degeneracy and instability in geometric computation. Proc IFIP WG5.2 GEO-6 Conference in Torkyo University, pp 5−15, 7−9 December 1998
19. Choi BK, Park SC (1999) A pair-wise offset algorithm for 2D point-sequence curve. Comput Aided Des 31(12):735−745
20. Lee DY, Kim SJ, Lee SG, Yang MY (2003) Incomplete mesh based tool path generation. Proc SMPE Spring Conference 2003, pp 844–847