Process information transferring for cutting paths in laser manufacturing

Qirui Hu\textsuperscript{1,2} · Zhiwei Lin\textsuperscript{1,2} · Jihui Liu\textsuperscript{1,2} · Jianzhong Fu\textsuperscript{1,2}

Received: 17 May 2022 / Accepted: 14 July 2022 / Published online: 23 July 2022
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2022

Abstract
In laser manufacturing, the cutting parameters, such as laser power, cutting speed, assist gas pressure, and stand off distance, have a great influence on the quality of the machined parts. To ensure the quality of the machined part, the process information (cutting parameters) needs to be manually assigned and adjusted according to the different geometry characters of the part edges, like arc and line segment. The part contours are then used for generating the laser cutting paths, leaving the process information behind. Quickly and accurately transferring the process information to the corresponding cutting paths is not an easy task. In this paper, an algorithm is proposed for this task. The proposed algorithm passes information by the points in the cutting paths. First, all part edges containing process information are offset to generate closed contours. Then, whether the points in the cutting paths are included in the closed contours are judged and marked. Finally, the information as NC codes through the adjacent points in the cutting paths is outputted. To simplify the computation work, a fast approach is also proposed to determine whether the point cloud is contained in a closed contour. The feasibility and computational efficiency of the proposed algorithms are verified by examples, and the actual cutting comparison verifies the superiority of the proposed algorithm by cutting details of machined parts.

Keywords Laser cutting · Cutting path · Cutting parameters · Point-inclusion problem

1 Introduction
Laser cutting is one of the main modern technologies in sheet metal product manufacturing industry. The use of laser cutting brings many benefits, such as minimizing the cutting width and reducing thermal deformation \cite{1}. A typical production process consists of designing parts, nesting the parts on metal sheets, and cutting the parts from the sheets \cite{2}. In the cutting process, the cutting parameters, e.g., laser power, cutting speed, assist gas pressure, and stand off distance, will affect the quality of the processed parts. Sundar et al. \cite{3} studied the effect of cutting parameters on the cutting quality of steel plates in laser cutting and found that cutting speed and gas pressure play important role in achieving the desired quality in cutting of thick carbon steel. Li et al. \cite{4} investigated the effect of cutting parameters on laser cutting quality of different workpiece materials and found that for the same thickness of different material plates, the optimal cutting parameters are also different.

During the whole process, the optimal cutting parameters may dynamically change to achieve the desired quality of parts, as Fig. 1 shows. Figure 1a shows a part containing a circular hole and Fig. 1b shows a part composed of arcs and line segments. In order to ensure the quality of the parts, cutting at the arc part should not be approximated by multiple straight segments, but in the form of arcs (as Fig. 1c shows) by changing the cutting parameters. Wang and Gao \cite{5} found through modeling analysis that under the same cutting parameters, there are differences between the surface roughness of inner arc, outer arc, and line segment. Moreover, the smaller the radius of the arc, the greater the difference in surface roughness. As shown in Fig. 1d, arcs (shown in red) and straight segments (shown in blue) should be cut with different cutting parameters, which is called Layered Cutting (LC) in this paper. The implementation method of LC will be introduced in Sect. 2. LC distinguished the cutting parameters according to the edges of the part, such
as straight segments and arcs in Fig. 1. This step usually requires the cutting parameters to be set manually on the corresponding part edge. LC can also be applied to another cutting technology, direct part marking (DPM), which has been proved feasible in the industrial environment, and is highly automated and environmentally friendly [6]. As illustrated in Fig. 2, the ring-shaped part has a marking contour “Z” (shown in blue) in addition to two circular shape contours (shown in red). Naturally, cutting marked contours requires lower laser power and larger stand off distance compared to cutting shaped contours.

Although LC can help cut higher quality parts, one challenge is to synchronize the process information from the part edges to the corresponding cutting paths. Considering the cutting width of the laser, the cutting path is not the original contour of a part, but an offset contour. For example, the part in Fig. 3 contains an inner contour and an outer contour, and the corresponding cutting paths are generated by offsetting these two contours (outer contour offsets to the outside, inner contour offsets to the inside with the laser radius being the offset distance). Moreover, offsetting may result in a different number of vertices or a different number of contours for the part contours and the offset contours. In the example of Fig. 4, the outer contour has 4 vertices, and the offset contour (shown in red) has 8 vertices. At the same time, one inner contour generates two offset contours. The characteristics of the offset prevent parts and offset contours from passing process information in a one-to-one correspondence. Not only that, the offset contours cannot be used directly as cutting paths, but has to be adjusted by cutting processes, like piercing, common cut, “jumper” [7], and bridging [8], as shown in Fig. 5. It can be seen from Fig. 5a that piercing may split the original offset contour and generate a new vertex. Common cut will remove overlapping offset contours like Part2 in Fig. 5b. Figure 5c shows an offset contour which is divided into two cutting paths by a “jumper.” And bridging will connect multiple offset contours together, as shown in Fig. 5d. In summary, original part contours need to be considered for the cutting width to generate offset contours, and the offset contour generates final cutting paths through cutting processes. In this way, it is not a simple task to transfer the process information from the part contours to the
corresponding cutting paths, especially when the number of parts is large.

In recent years, many studies on laser cutting have also considered the cutting width and cutting processes. Anaud and Babu [9] investigated the problem of nesting rectangular parts considering the cutting width (in his research called Kerf-width). Petunin and Stylios [8] studied the generation of optimal cutting paths (tool paths in his study) under multiple cutting constraints including cutting width and cutting processes like piercing and “jumper.” Chentsov et al. [10] used dynamic programming to solve the problem of cutting order for nested parts with constraints. And as with Petunin, the processing considered the cutting width. Huang et al. [11] studied the layout and path planning of rectangular parts with the influence of piercing. Although there have been many studies on laser cutting in recent years that have considered the inconsistency of part contours and cutting paths due to cutting width and cutting processes, to the best of our knowledge, there are no studies on process information transferring between part edges and cutting paths. Theoretically, it is possible to calculate the distances from the points on the part contours to the points on the cutting paths sequentially by the brute force (BF) algorithm, and find the points which carry information by comparing them with the distances of the cutting radius. However, the BF algorithm suffers from low computational efficiency in practice. As a typical situation for laser cutting, many parts are nested on one sheet, each of which contains at least one contour, and each contour is composed of several vertices, as Fig. 6 shows. Moreover, considering the possibility of cutting processes, cutting paths may create new vertices from original offset contour. Consequently, process information transfer cannot be performed only on individual parts or offset contours, but on all parts after nesting and generating the cutting paths by cutting processes. In that case, the algorithm based on the distance between points generates a huge amount of computation, resulting in a significant increase in computation time.

To solve this problem, a process information transferring algorithm based on contour offset and point cloud inclusion detection is proposed in this work. Specifically, after generating the cutting paths, all the part edges that carry process information are offset individually to generate closed contours. All generated contours are saved in different sets $S_i$ according to their different process information, in which “$i$” is the number of process information types. Next, all the vertices of the cutting paths are combined as a point cloud, and the vertices in the point cloud are judged and marked whether they need to accept the corresponding process information by determining whether they are inside the contours of $S_i$ (point-inclusion problem). Finally, the process information in the original part contours can be passed to
the corresponding cutting paths based on the marks in the
adjacent vertices of cutting paths. In this paper, we consider
the effect of offset distance on the contours containing pro-
cess information to avoid transfer errors. Besides, in order
to improve the computational efficiency, we consider the
corresponding algorithm for point cloud inclusion detection
to reduce calculation time. We have also tried various cases
to verify the feasibility and computational efficiency of the
proposed algorithm, and actual cutting experiment on 304
steel sheet verifies the superiority of the proposed algorithm.

The remainder of this paper is organized as follows. In
Sect. 2, the implementation method of LC and the offset
algorithm for closed and non-closed contours are introduced.
Clipper Lib, as an open-source geometry library, is also
introduced as a tool for calculating offset contours of closed
and non-closed contours in this paper. Section 3 presents
the process information transfer problem and corresponding
algorithm, along with the approach to simplify the computa-
tion of the point-inclusion problem. Section 4 gives some
examples to verify the feasibility and computational speed
of the algorithm proposed in this paper, and also conducts actual cutting experiments to verify the quality of the parts. The last section concludes this paper.

2 Layered cutting and contour offset

As introduced in Sect. 1, LC means that the cutting parameters are changed in the machining according to the different part contour characteristics like arc or laser mark. Cutting parameters are usually manually set on the edge of the part. In Fig. 7a, the arc edge of the part is set to be cut using different cutting parameters (shown in red). Each part needs to be set only once, and then, it can be nested in the desired quantity and angle, as shown in Fig. 7b. Figure 7b shows 10 parts of Fig. 7a nested on one sheet. It is worth mentioning that the cutting parameters include laser power, cutting speed, assist gas pressure, and stand off distance. If the specific values of these parameters are set on the edges of the part, it will bring more computational effort to subsequently transfer the information to the corresponding cutting paths. Each combination of different parameter types is usually considered as a cutting layer, so that the edges of the part only need to save the number of the cutting layer. In the example of Fig. 8, three cutting layers are set up. Cutting layer 1 is shown in black and represents a combination of 900 W laser power, 400 mm/min cutting speed, 5.5 bar gas pressure, and a stand off distance of 2 mm. We can just save the number of cutting layer to the needed edges of the part, and then pass the number to the corresponding cutting paths, and output the NC code. The machine cuts parts according to the parameters set by the number of cutting layer, to achieve dynamic changes in cutting parameters of the processing, i.e., LC. In this article, process information is the combination of cutting parameters and is simplified to the ordinal number of the cutting layer in LC. The algorithm proposed in this paper is to pass the process information from the part edges to the corresponding cutting paths, which will be introduced in Sect. 3.

Because of the cutting width and the cutting process, the part contour cannot be used directly for cutting, but has to be offset to generate the offset contour, which then generates the cutting paths according to the cutting processes (piercing, common cut, “jumper,” and bridging). The contour offset problem is a traditional and general plane geometry problem in computer graphics, and has been applied widely in rapid prototyping, tool paths for CNC machine, etc. The contour offset problem can be defined as follows: For an input contour, generate a new contour (or contours) on its side such that the distance from each point on the new contour (or contours) to the original contour is the given offset distance. Intuitively, it is to roll a circle with a radius of offset distance along the contour, and the trajectory of the circle center is the offset contour, as Fig. 9 shows. So far, contour offset algorithms are roughly divided into four categories: Voronoi diagram [12–14], level set [15, 16], edge offset [17, 18], and vertex offset [19]. Although contour offset is a simple problem and there are many algorithms to implement it, it is not easy to calculate the contour offset quickly and accurately. Fortunately, in the field of computational geometry, there are a number of readily available free open-source geometry libraries that can help us implement offsetting functions for planar contours, like CGAL (www.cgal.org) and Clipper (www.angusj.com/delphi/clipper.php). In addition to the offset of closed contours, non-closed contour offset, i.e., the offset of part edges, is also needed in the algorithm proposed in this paper, as shown in Fig. 10a. The example in Fig. 10a shows the offset of part edge. It can be noticed that although the edge of the part is non-closed contours, the generated offset contour is closed, which is called “influence area” in this paper and will be introduced in Sect. 3.2. Figure 10b shows how closed contours generate “influence area.” It should be noted that when the offset

| Laser power (W) | Cutting speed (mm/min) | Gas pressure (bar) | distance (mm) |
|-----------------|------------------------|--------------------|---------------|
| 900             | 400                    | 5.5                | 2             |
| 850             | 250                    | 5                  | 2             |
| 500             | 400                    | 3.5                | 4             |
contour is generated for closed contour, it follows the rule that the outer contour is offset outward, and the inner contour is offset inward. But when the “influence area” is generated for closed contour, it is offset in two directions, and the “influence area” is composed of offset contours in two directions, as Fig. 10b shows.

All offset calculations in this paper are finished by Clipper Lib, including the part contour and the part edges (closed contour and non-closed contour). Clipper Lib is an open-source computational geometry library dedicated to Boolean operations on closed or non-closed contours in the plane, enabling intersection, union, difference, and difference and offset operations on plane contours. Clipper Lib defines the contour direction for the input and output as counterclockwise for the outer contour and clockwise for the inner contour. The input contour is a set of vertices, i.e., if there are contours defined by topological information such as arcs (defined by the circle center coordinates, radius, start angle, and end angle) in the part contour, these contours need to be discretized into multiple vertices, as shown in the example in Fig. 1c.

3 Process information transferring algorithm

3.1 Problem definition and data structure

In order to ensure the quality of the processed parts, different types of part contours should be cut with different cutting parameters in laser cutting. We refer to each cutting parameter combination as a cutting layer as introduced in Sect. 2. The cutting layer can be further divided into special cutting layer and default cutting layer. The special cutting layer can be used to cut arcs or laser marks, etc., while the default cutting layer is used to cut line segments. In general, all part contours are processed with the cutting parameters of the default cutting layer. If there are contours in the part that need to be processed with different cutting parameters, these special contours are put into the corresponding special cutting layers, which are called contours containing process information with the process information being the number of the special cutting layers to which they belong. The part contours are offset to generate the offset contours, and the offset contours need to set the cutting processes to generate the final cutting paths. In order to machine the part with different cutting parameters, it is necessary to pass all process information from the part contours to the corresponding cutting paths.

For example, the part in Fig. 11a consists of an arc and line segments, which belong to different cutting layers (distinguished by blue and black). The arc part only has topological information (circle center coordinates, radius, starting angle, ending angle) and cannot be offset directly, which needs to be replaced by a set of line segments as Fig. 11b shows. In Fig. 11b, the original arc is replaced by 8 line segments and 7 new vertices are generated. Next, the modified part contour is offset to generate the offset contour, as shown in Fig. 11c. The target is to pass process information...
of the original part to the corresponding cutting paths like Fig. 11d shown. It can be seen that the newly added 7 vertices in Fig. 11b increase to 14 in the offset path. It is conceivable that if an arc is replaced by $n$ line segments, which creates $n - 1$ new vertices, then at least $2n - 2$ new vertices will be added to the newly generated offset cutting paths. By comparing Fig. 11c, d, it is obvious that if the process information can be transferred to the cutting paths, not only will the quality of the cut part contour be improved, but the overall cutting time and the number of NC codes will also be reduced.

The data structure of all parts used for nesting is shown in Fig. 12. All parts are in a part set ($S_p$) and each part is also a contour set. Each contour is a vertex set, and two adjacent vertices form an edge. To make it easier to understand, the vertex structure which is used throughout the paper is defined first.

**Definition 1.** Struct Vertex {double $x$, $y$; bool layer1, layer2, layer3 = False;}

In this structure, the real numbers $x$ and $y$ are the coordinates of the vertex in Euclidean space; the Boolean variables layer1, layer2, and layer3 are used to determine if the vertex is in specific cutting layers and both default to the initial value of “False” (the initial case vertex in the default cutting layer). The number of Boolean variables in the data structure of the vertex is kept consistent with the actual number of specific cutting layers, and only three are listed here. Generally speaking, all parts on a plate can be machined in one cutting layer, the default cutting layer. In contrast, parts such as arcs or laser marks that require different cutting parameters are in different special cutting layers and the process information is which special cutting layer the part contour is located in.

This paper does not discuss part nesting algorithm, cutting process settings, and cutting path planning algorithm.
The algorithm in this paper is to pass all process information to all cutting paths before the NC code is finally generated. The example in Fig. 11 discusses just one part or one contour, but in actual laser cutting, there are usually hundreds of parts on a sheet, as shown in Fig. 13. As shown in Fig. 13, all the process information is saved in the nested part and needs to be passed to the corresponding cutting paths. For discrete substitution of original contours such as arcs, the process information is also assigned to the corresponding discrete points. In this example, the BF algorithm generates a very large amount of computation, so we need to use other methods to simplify the computation.

3.2 Process information transferring algorithm

As introduced in Sect. 3.1, all process information is stored in nested parts, as Fig. 14 shows. In the example shown in Fig. 14, there are two nested parts to be cut, each containing one or two special cutting layers (represented in red and blue), and the part edges of these special cutting layers are stored in the corresponding sets (S_{L1} and S_{L2}). Next, all the part edges in S_{L1}, S_{L2}, and S_{L3} need to be offset to generate closed contours, i.e., “influence areas”, and all the “influence areas” generated from S_{L1}, S_{L2}, and S_{L3} are saved into S_{PL1}, S_{PL2}, and S_{PL3}. Figure 15 illustrates the effect of line segment offset distance on the results. The rectangular part in Fig. 15a contains a line segment of a special cutting layer (shown in red) L_1, which needs to pass process information to P_1, P_2, P_3, and P_4. If the offset distance of L_1 is the same as the cutting radius R_C, as shown in Fig. 15b, the points P_1, P_2, P_3, and P_4 are just on the offset contour. Considering the truncation error, the “influence areas” may not include all points P_1, P_2, P_3, and P_4. The most suitable offset distance should be slightly larger than the cutting radius, as shown in Fig. 15c. If the offset distance is too large, then the “innocent” points will be affected, as in Fig. 15d, two “innocent” points, P_5 and P_6, will also be affected. In the actual algorithm, we use 1.0001 R_C as the offset distance of the “influence areas.”

With all the “influence areas” available, it is also necessary to prepare all the vertices on the cutting paths that are expected to receive the process information. All vertices on the cutting paths need to be determined in turn whether they are included in the “influence areas” of S_{PL1}, S_{PL2}, and S_{PL3}. And the Boolean variables layer1, layer2, and layer3 in the data structure of vertex can mark whether these vertices...
belong to special cutting layers. If a vertex is within any of the contours in SPL1, layer1 marks true. Accordingly, the layer2 and layer3 properties are also true if this vertex is also within the contours of SPL2 and SPL3. Each vertex will exist in at most two cutting layers, as shown in Fig. 16. In Fig. 16, two line segments, L1 and L2, are in different special cutting layers (red and blue), resulting in the vertices P1 and P2 being in two special cutting layers. Each vertex on the cutting paths has to perform point-inclusion calculation with the contours in SPL1, SPL2, and SPL3, which will generate a huge amount of computation.

The point cloud formed by these vertices is denoted as SV. Then, the points in SV and the contours in SPL1, SPL2, and SPL3 need to be detected one by one to pass process information to the points in SV. To reduce the amount of computation, we propose an approach to quickly determine which points in a point cloud are included in a closed contour, which will be presented in Sect. 3.3. Once the points in SV are recorded with process information, they need to be converted to specific cutting layers in the NC code. Taking Fig. 16 as an example, the rectangular part contains two special cutting layers of edges L1 and L2. The layer1, layer2, and layer3 attributes of each point on the cutting path and the generated NC code are shown in Fig. 17. In the NC code of Fig. 17, different cutting layers are distinguished by “!GOP0!,” “!GOP1!,” and “!GOP2!” “!GOP0!” is the default cutting layer. “!GOP1!” and “!GOP2!” represent special cutting layer1 and layer2 respectively. Based on the Boolean values of the layer1, layer2, and layer3 attributes of the two adjacent vertices on one cutting path, the process information can be converted and output by the NC code. When both vertices have the same attribute value of true, the line segment is cut using the corresponding special cutting layer, such as P2P3 (layer1), P3P4 (layer1), P5P6 (layer2), and P6P1 (layer2). When there is no match between the two vertices, the default cutting layer is used, such as P4P5, P5P6, and P6P7. P6P1 is a special line segment, and it can be seen that the layer1 and layer2 properties are true for both P1 and P2, meaning that this line segment is on two special cutting layers at the same time. This is because the adjacent edges on the original part are in different special cutting layers, and P1P2 is not an edge of the original part, but just a new “connecting edge” after offset. In practice, such “connecting edge” is very short and have no effect on the quality of the cut part. Therefore, either layer1 or layer2 cutting layer can be used.

### 3.3 Point-inclusion calculation

Section 3.2 describes the process information transferring algorithm, in which inclusion detection is required for contours in SPL1, SPL2, and SPL3 and point clouds SV. This section describes the approach we adopt to reduce the computation.

Taking point cloud and a contour calculation as an example, the vertices in the point cloud are first sorted in ascending order of their X-coordinate values (if the X-coordinate values of two points are the same, they are arranged in ascending order of Y-coordinate values), and the arranged point cloud is recorded as SV. Sorting point clouds can help reduce a lot of time since point clouds need to be calculated with many contours. In addition to
the vertices, the contour also needs to be sorted by edges. Consider a closed contour with eight vertices (P1–P8) and eight edges (EP1P2–EP8P1), as shown in Fig. 18. Pmax (Xmax, Ymax) and Pmin (Xmin, Ymin) are the maximum and minimum points on the rectangular enclosure of this contour, respectively. All eight edges are sorted in ascending order by the minimum value of X-coordinate for each edge (the smaller X-coordinate values of the two vertices). If both edges have the same X-coordinate values (four points of the two edges have same X-coordinates values), then they are sorted in ascending order by Y-coordinate value. The sorted edges are saved in a set of edges (in this example, the order is EP2P3, EP1P2, EP3P4, EP4P5, EP8P1, EP7P8, EP5P6, EP6P7). Once the ordering of the point cloud and contour edges is complete, the point-inclusion calculation can be performed.

First, based on Xmin and Xmax of Pmin and Pmax, the vertices in SOV can be screened out by the following inequality:

\[ X_{\text{min}} \leq X_i \leq X_{\text{max}} \]  

in which X_i is the X-coordinate value for a point in SOV. All points that do not satisfy the inequality 1 cannot be inside the contour. Since these contours are offset and the offset distance is 1.0001 \( R_C \), all points will not be on the boundary of the contour, so there is no need to take the equal sign on both sides of inequality 1. Because the points in SOV have been sorted, when the first point with X_i greater than Xmax is detected in SOV, the detection can be terminated to reduce the amount of calculation. Similarly, if the number of vertices in the point cloud is large, the first vertex with X_i greater than Xmin can be found by dichotomy. Points satisfying inequality 1 in SOV are denoted as SOV\', and if the number of points in SOV\' is still large, they can be sorted by Y-coordinate value in a similar way and screened out again according to the following inequality:

\[ Y_{\text{min}} \leq Y_i \leq Y_{\text{max}} \]  

in which Y_i is the Y-coordinate value for a point in SOV\'. So far, we have excluded all points in SOV that are not within the rectangle enclosure of the contour, and the remaining points are recorded as SS.

Next, take a point Pi (X_i, Y_i) from SS each time to judge whether it is inside the contour. According to Pi, all edges of the contour can also be screened, which is similar to the screening of the point in SOV. Figure 19 shows an example of the inclusion calculation method of point cloud and contour. In Fig. 19, P1 (X1, Y1) and P2 (X2, Y2) are the two points of SS, and EP2P3–EP8P1 are the eight edges of the contour. Firstly, according to X1, the edges whose X_{E\text{min}} is greater than X1 and X_{E\text{max}} is smaller than X1 can be screened out. Because the edges are also sorted, when the first unsatisfactory edge (X_{E\text{min}} is greater than X1) is detected, the subsequent segments can be excluded too. The remaining edges are denoted as SP_E. For P1, the
remaining edges $S_{P1E}$ are $E_{P1P2}$, $E_{P2P3}$, and $E_{P3P4}$. Similarly, $Y_1$ can be used to further screen out edges that do not intersect with $P_1$, and the remaining edges are denoted as $S_{P1E}'$ ($E_{P2P3}$ and $E_{P3P4}$ for $P_1$). Then, calculate whether the edges in $S_{P1E}'$ intersects with the ray of $P_1$, and judge whether $P_1$ is included in the contour according to the total number of intersections (odd number is in the contour, even number is not). Then, when it comes to $P_2$, because the point cloud and the contour are sorted, it is not necessary to traverse all contour edges again to find the $S_{P2E}$ but can dynamically maintain $S_{P1E}$ (delete edges that do not meet the constraints, add new edges that meet the requirements) to reduce the amount of computation.

For each contour set like $S_{PL1}$, if a vertex $P_0$ in $S_{PE}$ is included in a contour in $S_{PL1}$, the corresponding Boolean variable of this vertex will be marked, as introduced in Sect. 3.1. And $P_2$ does not need to be judged again whether it is included in other contours in $S_{PL1}$. For multiple contours in $S_{PL1}$, the number of points in the $S_{PE}$ that need to be judged can be reduced by this method.

4 Examples

In order to verify the feasibility and superiority of the method proposed in Sect. 3, several experiments are demonstrated in this section.

To verify the practicability of the proposed problem and algorithm, we select a part as shown in Fig. 20. Figure 20a shows the part we chose, consisting of an arc and three line segments. Figure 20b shows the cutting paths of two selected parts on the plate. The difference between the two parts is that the part on the right includes the process information using the method proposed in this paper (shown in red on the cutting path). Figure 20c is the NC code corresponding to the cutting path in Fig. 20b. It can be found from Fig. 20c that the process information is passed to the corresponding cutting path; different cutting layers are also used for cutting in the NC code. However, due to the limitation of the offset method, the original arc is replaced by multiple line segments, resulting in the original arc not being cut according to the curve mode, but replaced by multiple straight lines. In order to cutting arcs by curve mode, the topology information of arcs needs to be restored from the cutting path. Figure 21 shows two methods to solve this problem. The method shown in Fig. 21a is that after passing the process information, the center of the arc can be found through the intersection of the mid-perpendicular lines, and the radius

---

![Fig. 19](https://example.com/fig19.png)

**Fig. 19** Example of the inclusion calculation method of point cloud and contour

![Fig. 20](https://example.com/fig20.png)

**Fig. 20** Example of part cutting with process information

---
can be calculated through the vertex on cutting path. If the topology information in the original part is saved, the radius can be calculated from the existing center of the arc, as shown in Fig. 21b. Figure 21c shows the NC code generated from the arc topology information. Comparing with Fig. 20c, it can be found that cutting arcs with curve mode will not only save processing time and improve the quality of part, but also reduce the size of processing file by greatly reducing the lines of NC code.

After verifying the proposed problem and the practicability of the algorithm, we designed a more complex part for testing the computational speed of proposed algorithm. As can be seen in Fig. 22, the designed part has both inner and outer contours, involving a total of five cutting layers (four special cutting layers and a default cutting layer). The running speed and stability of the proposed algorithm will be tested by adjusting the number of the designed part. The proposed algorithm has been programmed in the C++ language and run on a personal computer with a dual-core CPU @2.70GHz and 8 GB RAM. For different numbers of designed parts, the calculation time from process information transfer to the generation of NC code is shown in Table 1. The running time for each part number in Table 1 is the average of 50 calculations. For 100 parts, this would involve calculating the point inclusion of 4800 points and 400 contours and converting the process information of 4800 points into the corresponding NC code in a time of 0.219 s. And for the calculation of 48,000 points and 4000 contours, it only takes 75.265 s. From this table, it can be seen that the proposed algorithm is able to quickly and accurately transfer the process information of all parts to the corresponding cutting paths and generate the corresponding NC codes. Besides this example, the proposed algorithm succeeds in many examples that have been tested.

![Fig. 21](image1.png)  Two methods to restore the topology information of arcs

![Fig. 22](image2.png)  Example of a designed part with process information

![Table 1](image3.png)  Running time for different number of parts

| Part amount | Vertices’ amount | “Influence areas” amount | Average running time (s) |
|-------------|-----------------|------------------------|-------------------------|
| 100         | 4800            | 400                    | 0.219                   |
| 200         | 9600            | 800                    | 0.844                   |
| 300         | 14,400          | 1200                   | 3.064                   |
| 400         | 19,200          | 1600                   | 7.972                   |
| 500         | 24,000          | 2000                   | 14.578                  |
| 1000        | 48,000          | 4000                   | 75.265                  |
In order to verify the quality of the processed parts, we used a 3000-W laser cutting machine to actually cut 3-mm-thick parts. The material of the part is 304 steel, and the cutting radius is 2 mm. Figure 23a shows two parts from the example in Fig. 23, where the left part was cut using the conventional method and the right part was cut after passing the process information and adjusting the NC code accordingly. From Fig. 23a, it can be seen that for parts with arcs or other types that need to be processed by changing the cutting parameters, the proposed method can improve the quality of the machined parts. The actual cut part for the example in Fig. 22 is shown in Fig. 23b. As can be seen from Fig. 23b, the algorithm proposed in this paper can be used not only for cutting of arcs, but also for laser marks.

5 Conclusions

In laser cutting, the choices of laser power, cutting speed, assist gas pressure, and stand off distance determine the quality of the cut part. Literature indicates that different cutting parameters should be used for different materials with different thicknesses of plates to obtain the best quality of the cut parts. In addition, it has also been confirmed that the same cutting parameters can produce quality differences between different cutting paths on the same sheet. This problem can be solved by dividing the parts into several cutting layers, using different cutting parameters for each layer, like line segments and arcs, which is called Layered Cutting in this paper. Due to the cutting width and cutting processes, the actual cutting paths are not the part contours. In order to change the cutting parameters to cut different contours of the part, it is necessary to pass the process information in the part contours to the corresponding cutting paths, which produces the process information transfer problem.

In order to solve the problem of process information transfer, an offset-based process information transferring algorithm is proposed in this paper. Specifically, all the vertices in the cutting paths form a point cloud. Next, all part edges containing process information are offset into "influence areas" and saved separately by special cutting layers. All points in the point cloud transfer process information by determining whether they are contained within the "influence areas." Eventually, the process information of the adjacent points in the cutting paths can be used to generate the NC code containing the special cutting layers. To reduce the computational effort, a fast approach is also proposed to determine whether the point cloud is within a specific contour. The feasibility and efficiency of the algorithm are verified by the given examples. And the superiority of the algorithm is proved by actual cutting experiments.

Author contribution Not applicable.
Funding This work was financially supported by the Key R & D plan of Zhejiang Province (No. 2020C01069).

Availability of data and materials Not applicable.

Code availability Not applicable.

Declarations

Consent to participate Consent to participate.

Consent for publication Consent for publication.

Competing interests The authors declare no competing interests.

References

1. Makarovskikh T, Panyukov A (2021) Software for the problem of constructing cutting tool paths in CAD/CAM systems for technological preparation of cutting processes. Automat Rem Contr+ 82(3)
2. Makarovskikh T, Panyukov A (2018) Development of routing methods for cutting out details. OPTA-SCL
3. Sundar M, Nath K, Bandyopadhyay K, Chaudhuri S, Dey P, Misra D (2009) Effect of process parameters on the cutting quality in laser cutting of mild steel[J]. Int J Adv Manuf Technol 40(9–10):865–874
4. Li M, Li S, Yang X, Zhang Y, Liang Z (2019) Effect of lay-up configuration and processing parameters on surface quality during fiber laser cutting of CFRP laminates[J]. Int J Adv Manuf Technol 100(1):623–635
5. Wang D, Gao X (2015) Study on kerf roughness differences between line and arc analysis and elimination of laser precision cutting. Chinese Journal of Lasers 42(3)
6. Zhang L, Chen Y, Li G (2020) A study of laser direct-part marking of 2D code on the aviation aluminum alloy part. Conference on Advanced Laser Processing and Manufacturing IV 12–16
7. Reginald D, Pieter V, Dirk C (2016) A review of cutting path algorithms for laser cutters. Int J Adv Manuf Technol 87(5–8)
8. Petunin A, Stylios C (2016) Optimization models of tool path problem for CNC sheet metal cutting machines[J]. IFAC-PapersOnLine 49(12):23–28
9. Anand K, Babu A (2015) Heuristic and genetic approach for nesting of two-dimensional rectangular shaped parts with common cutting edge concept for laser cutting and profile blanking processes. Comput Ind Eng 80:111–124
10. Chentsov G, Chentsov A, Petunin A and Sesekin N (2018) Model of megalopolises in the tool path optimisation for CNC plate cutting machines. Int J Prod Res
11. Huang X, Xi F, Li J, Zhong Z (2009) Optimal layout and path planning for flame cutting of sheet metals[J]. Int J Comput Integr Manuf 22(1):30–41
12. Persson H (1978) NC machining of arbitrarily shaped pockets[J]. Comput Aided Design 10(3):169–174
13. Held M, Lukács G, Andor L (1994) Pocket machining based on contour-parallel tool paths generated by means of proximity maps[J]. Comput Aided Design 26(3):189–203
14. Held M (2001) VRONI: An engineering approach to the reliable and efficient computation of Voronoi diagrams of points and line segments. Comput Geom 18(2):95–123
15. Kimmel R, Bruckstein A (1993) Shape offsets via level sets[J]. Comput Aided Design 25(3):154–162
16. Shen H, Fu J, Chen Z, Gan W (2011) Iso-scallop controlled tool path generation for free-form surfaces based on level set method[J]. J Comput Theor Nanosci 4(8):3140–3145
17. Kim K, Jeong J (1995) Tool path generation for machining freeform pockets with islands[J]. Comput Ind Eng 28(2):399–407
18. Liu X, Yong J, Zheng G, Sun J (2007) An offset algorithm for polylines curves[J]. Computer Ind 58(3):240–254
19. Lee C, Phan T, Kim D (2009) 2D curve offset algorithm for pockets with islands using a vertex offset[J]. Int J Precis Eng Manuf 10(2):127–135

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.