Spiral tool paths for high-speed machining of 2D pockets with or without islands

Abrahamsen, Mikkel

Published in:
Journal of Computational Design and Engineering

DOI:
10.1016/j.jcde.2018.01.003

Publication date:
2019

Document version
Publisher's PDF, also known as Version of record

Document license:
CC BY-NC-ND

Citation for published version (APA):
Spiral tool paths for high-speed machining of 2D pockets with or without islands

Mikkel Abrahamsen

Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100 København Ø, Denmark

1. Introduction

A fundamental problem often arising in the CAM industry is to find a suitable tool path for milling a pocket that is defined by a shape in the plane. A CNC milling machine is programmed to follow the tool path and thus cutting a cavity with the shape of the given pocket in a solid piece of material. The cutter of the machine can be regarded as a circular disc with radius \( r \), and the task is to find a tool path in the plane such that the swept volume of the disc, when the disc center is moved along the path, covers the entire pocket. We assume for simplicity that the given pocket is consisting of all the points where the center of the cutter is allowed to be so that the tool path is allowed to be anywhere in the pocket and nowhere outside.

Some work has been made on spiral tool paths that morph a point within the pocket to the boundary of the pocket (Banerjee, Feng, & Bordatchev, 2012; Bieterman & Sandstrom, 2003; Chuang & Yang, 2007; Held & Spielberger, 2009, 2014; Held & de Lorenzo, 2018; Huang, Lynn, & Kurfess, 2017; Huertas-Talón, García-Hernández, Berges-Muro, & Gella-Marín, 2014; Patel & Lalwani, 2017; Romero-Carrillo, Torres-Jimenez, Dorado, & Díaz-Garrido, 2015; Xu, Sun, & Zhang, 2013; Zhou, Zhao, & Li, 2015). The method described by Held and Spielberger (2009) yields a tool path that (i) starts at a user-specified point within the pocket, (ii) ends when the boundary is reached, (iii) makes the cutter remove all material in the pocket, (iv) has no self-intersections, (v) is \( G^1 \) continuous,\(^{2} \) (vi) makes the width of the material cut away at most \( \delta \) at any time, where \( \delta \) is a user-defined constant called the stepover.

Held and de Lorenzo (2018) simplified the method by Held and Spielberger. Note that we must have \( \delta < r \), since otherwise some material might not be cut away. See Fig. 2(c) for an example of such a spiral tool path. It is the result of an algorithm described in the present paper, but has a similar appearance as the spirals described by Held and Spielberger (2009) and Held and de Lorenzo (2018).

We refer to Held and Spielberger (2009) for a detailed discussion of the benefits of spiral tool paths in high-speed machining compared to various other tool path patterns and more information on CNC milling in general.

We describe an alternative construction of spirals that also satisfy the previously mentioned properties of the construction of.

\(^{2} \) A plane curve is \( G^1 \) continuous or tangent continuous if a parameterization of the curve by arclength is differentiable.
Held and Spielberger (2009). In practice, it is very common that there is one or more islands in the pocket that should be avoided by the cutter, for instance, if there are areas of material that should not be machined to the same depth. It is only described by Held and Spielberger (2009). In practice, it is very common that one can obtain with spirals that morphs a point to the pocket boundary, see Fig. 6(b) for an example. This is our biggest new contribution. We exploit the fact that the Voronoi diagram of a pocket with an island consists of exactly one cycle and trees rooted at that cycle to define a wave that starts at the boundary of the island at time 0 and propagates outwards, reaching the outer boundary at time 1. This wave is used to define our spiral tool path. We shall demonstrate natural applications of this method to traverse the boundaries monotonically forward in clockwise direction. These methods would for instance not work for a pocket with an island such as in Fig. 7. Our method works with no assumption on the geometry of the pocket and the island.

The paper is based on the author’s experiences while developing the morphed spiral strategy for the Autodesk<sup>®</sup> CAM products (HSMWorks, Inventor HSM<sup>®</sup>, and Fusion 360<sup>®</sup>). The morphed spiral seems to be quite popular among the users. Due to the abundant number of real-world parts that have been available during the development, we guarantee that it is possible to make an efficient industrial-strength implementation of the algorithms described here.

We use Held’s VRT<sup>®</sup> library for the computation of Voronoi diagrams (Held, 2001). All figures in the paper are automatically generated using our implementation of the algorithms.

The rest of the paper is structured as follows: In Section 2, we describe our basic method for making a spiral that morphs a point to the boundary in a simply-connected pocket. Section 3 describes how the method is adapted to a pocket with one island. Using that method, we describe in Section 4 an alternative spiral in simply-connected pockets which will be superior to the one from Section 2 in many cases. In Section 5, we show how to construct at spiral around arbitrarily many islands by first connecting the islands into one island. Finally, we conclude the paper in Section 6 by suggesting some future paths of development of spiral tool paths.

### 2. Computing a spiral in a pocket without islands

In this section we describe a method to compute a spiral in a given simply-connected 2D pocket \( P \), see Fig. 2(c). In practice, the boundary of a pocket is often described by line segments and more advanced pieces of curves, such as circular arcs, elliptic arcs, and splines. However, it is always possible to use a sufficiently accurate linearization of the input, so we assume for simplicity that \( P \) is a polygon.

Our algorithm first constructs a polyline spiral, see Fig. 2(b). The polyline spiral must respect the stepover \( \delta \), i.e., the distance from every point to the neighboring revolutions and the distance from the outermost revolution to the boundary of \( P \) at most \( \delta \). In Section 2.8 we devise a method for rounding the polyline spiral to get a \( C^1 \) continuous spiral consisting of line segments and circular arcs.

The paper is based on the author’s experiences while developing the morphed spiral strategy for the Autodesk<sup>®</sup> CAM products (HSMWorks, Inventor HSM<sup>®</sup>, and Fusion 360<sup>®</sup>). The morphed spiral seems to be quite popular among the users. Due to the abundant number of real-world parts that have been available during the development, we guarantee that it is possible to make an efficient industrial-strength implementation of the algorithms described here.

We use Held’s VRT<sup>®</sup> library for the computation of Voronoi diagrams (Held, 2001). All figures in the paper are automatically generated using our implementation of the algorithms.

The rest of the paper is structured as follows: In Section 2, we describe our basic method for making a spiral that morphs a point to the boundary in a simply-connected pocket. Section 3 describes how the method is adapted to a pocket with one island. Using that method, we describe in Section 4 an alternative spiral in simply-connected pockets which will be superior to the one from Section 2 in many cases. In Section 5, we show how to construct at spiral around arbitrarily many islands by first connecting the islands into one island. Finally, we conclude the paper in Section 6 by suggesting some future paths of development of spiral tool paths.

2. Computing a spiral in a pocket without islands

In this section we describe a method to compute a spiral in a given simply-connected 2D pocket \( P \), see Fig. 2(c). In practice, the boundary of a pocket is often described by line segments and more advanced pieces of curves, such as circular arcs, elliptic arcs, and splines. However, it is always possible to use a sufficiently accurate linearization of the input, so we assume for simplicity that \( P \) is a polygon.

Our algorithm first constructs a polyline spiral, see Fig. 2(b). The polyline spiral must respect the stepover \( \delta \), i.e., the distance from every point to the neighboring revolutions and the distance from the outermost revolution to the boundary of \( P \) at most \( \delta \). In Section 2.8 we devise a method for rounding the polyline spiral to get a \( C^1 \) continuous spiral consisting of line segments and circular arcs.

The paper is based on the author’s experiences while developing the morphed spiral strategy for the Autodesk<sup>®</sup> CAM products (HSMWorks, Inventor HSM<sup>®</sup>, and Fusion 360<sup>®</sup>). The morphed spiral seems to be quite popular among the users. Due to the abundant number of real-world parts that have been available during the development, we guarantee that it is possible to make an efficient industrial-strength implementation of the algorithms described here.

We use Held’s VRT<sup>®</sup> library for the computation of Voronoi diagrams (Held, 2001). All figures in the paper are automatically generated using our implementation of the algorithms.

The rest of the paper is structured as follows: In Section 2, we describe our basic method for making a spiral that morphs a point to the boundary in a simply-connected pocket. Section 3 describes how the method is adapted to a pocket with one island. Using that method, we describe in Section 4 an alternative spiral in simply-connected pockets which will be superior to the one from Section 2 in many cases. In Section 5, we show how to construct at spiral around arbitrarily many islands by first connecting the islands into one island. Finally, we conclude the paper in Section 6 by suggesting some future paths of development of spiral tool paths.

2. Computing a spiral in a pocket without islands

In this section we describe a method to compute a spiral in a given simply-connected 2D pocket \( P \), see Fig. 2(c). In practice, the boundary of a pocket is often described by line segments and more advanced pieces of curves, such as circular arcs, elliptic arcs, and splines. However, it is always possible to use a sufficiently accurate linearization of the input, so we assume for simplicity that \( P \) is a polygon.

Our algorithm first constructs a polyline spiral, see Fig. 2(b). The polyline spiral must respect the stepover \( \delta \), i.e., the distance from every point to the neighboring revolutions and the distance from the outermost revolution to the boundary of \( P \) at most \( \delta \). In Section 2.8 we devise a method for rounding the polyline spiral to get a \( C^1 \) continuous spiral consisting of line segments and circular arcs.

The paper is based on the author’s experiences while developing the morphed spiral strategy for the Autodesk<sup>®</sup> CAM products (HSMWorks, Inventor HSM<sup>®</sup>, and Fusion 360<sup>®</sup>). The morphed spiral seems to be quite popular among the users. Due to the abundant number of real-world parts that have been available during the development, we guarantee that it is possible to make an efficient industrial-strength implementation of the algorithms described here.

We use Held’s VRT<sup>®</sup> library for the computation of Voronoi diagrams (Held, 2001). All figures in the paper are automatically generated using our implementation of the algorithms.

The rest of the paper is structured as follows: In Section 2, we describe our basic method for making a spiral that morphs a point to the boundary in a simply-connected pocket. Section 3 describes how the method is adapted to a pocket with one island. Using that method, we describe in Section 4 an alternative spiral in simply-connected pockets which will be superior to the one from Section 2 in many cases. In Section 5, we show how to construct at spiral around arbitrarily many islands by first connecting the islands into one island. Finally, we conclude the paper in Section 6 by suggesting some future paths of development of spiral tool paths.

2. Computing a spiral in a pocket without islands

In this section we describe a method to compute a spiral in a given simply-connected 2D pocket \( P \), see Fig. 2(c). In practice, the boundary of a pocket is often described by line segments and more advanced pieces of curves, such as circular arcs, elliptic arcs, and splines. However, it is always possible to use a sufficiently accurate linearization of the input, so we assume for simplicity that \( P \) is a polygon.

Our algorithm first constructs a polyline spiral, see Fig. 2(b). The polyline spiral must respect the stepover \( \delta \), i.e., the distance from every point to the neighboring revolutions and the distance from the outermost revolution to the boundary of \( P \) at most \( \delta \). In Section 2.8 we devise a method for rounding the polyline spiral to get a \( C^1 \) continuous spiral consisting of line segments and circular arcs.

The paper is based on the author’s experiences while developing the morphed spiral strategy for the Autodesk<sup>®</sup> CAM products (HSMWorks, Inventor HSM<sup>®</sup>, and Fusion 360<sup>®</sup>). The morphed spiral seems to be quite popular among the users. Due to the abundant number of real-world parts that have been available during the development, we guarantee that it is possible to make an efficient industrial-strength implementation of the algorithms described here.

We use Held’s VRT<sup>®</sup> library for the computation of Voronoi diagrams (Held, 2001). All figures in the paper are automatically generated using our implementation of the algorithms.

The rest of the paper is structured as follows: In Section 2, we describe our basic method for making a spiral that morphs a point to the boundary in a simply-connected pocket. Section 3 describes how the method is adapted to a pocket with one island. Using that method, we describe in Section 4 an alternative spiral in simply-connected pockets which will be superior to the one from Section 2 in many cases. In Section 5, we show how to construct at spiral around arbitrarily many islands by first connecting the islands into one island. Finally, we conclude the paper in Section 6 by suggesting some future paths of development of spiral tool paths.
necessary and sufficient for the stepover to be respected among all neighboring wavefronts. Therefore, we choose $p_0$ as the point in $VD$ that minimizes the longest distance to a leaf in $VD$. That is a unique point traditionally known as the center of $VD$. Handler (1973) gives a simple algorithm to compute $p_0$ in time proportional to the size of $VD$. The center will most likely not be a node in $VD$, but an interior point on some edge. In that case, we introduce a node at $p_0$ by splitting the edge into two edges.

2.3. Our representation of $VD$

We consider $VD$ as a directed, rooted tree with the node $\text{Root}$ at $p_0$ being the root. For a node $n$ in $VD$, we let $\text{Pt}[n]$ be the position of the node $n$. Let $VD[n]$ be the subtree rooted at node $n$. For a node $n \neq \text{Root}$, we store a pointer $\text{ParentE}[n]$ to the edge having end node $n$. We say that edge $\text{ParentE}[n]$ is the parent edge of node $n$ and of any edge having start node $n$. We also store an array.

Fig. 1. (a) The Voronoï diagram. (b) The Voronoï diagram enriched with equidistantly placed segments perpendicular to long edges. (c) The final diagram $VD$ where double edges going to concave corners of $P$ are replaced by their angle bisector.

Fig. 2. Wavefronts (a), polyline spiral (b), and the final $C^1$ continuous spiral (c) in the same polygon $P$. The diagram $VD$ is in gray.
ChildEs[n] of the edges going out of n sorted in counterclockwise order with the edge following ParentE[n] being the first. For Root, the choice of the first child edge does not matter. For each edge e, we store pointers Start[e] and End[e] to the start and end nodes of e. We also store an index \( i = \text{IndexInStart}[e] \) such that ChildEs[Start[e]][\( i ] = e. If e is an edge, we say that Start[e] and End[e] are incident to e and that e is incident to Start[e] and End[e]. For an edge \( e_i \) and node \( n \) incident to \( e_1 \), we let NextCCW(\( e_i, n \)) = \( e_2 \), where \( e_2 \) is the edge after \( e_1 \) among the edges incident to \( n \) in counterclockwise order. The function NextCCW can be implemented so that it runs in constant time using the values defined here.

Using NextCCW, we can traverse all of \( VD \) in the counterclockwise direction in linear time. We start setting \((n, e) = (\text{Root}, \text{ChildEs}[\text{Root}][0]) \). In each iteration, we let \( n \) be the other node incident to \( e \) and then set \( e = \text{NextCCW}(e, n) \). We stop when we have traversed every edge, i.e., when \((e, n) = (\text{Root}, \text{ChildEs}[\text{Root}][0]) \) at the end of an iteration. Note that each edge \( e \) is visited twice, once going down the tree \( VD[\text{Start}[e]] \) and once going up.

2.4. Defining the movement of the wave

In the following, we outline the model which we define the movement of the wave. Let \( \text{Hgt}[n] \) for each node \( n \) be the length of the longest path from \( n \) to a leaf in \( VD[n] \). All the \( \text{Hgt} \) values can be computed in linear time by traversing \( VD \) once. For each node \( n \), we define the time \( \text{TmNd}[n] \) of the wave reaches \( n \). We set \( \text{TmNd}[\text{Root}] = 0 \). We also define the speed \( \text{VNd}[n] \) that the wave has when it reaches \( n \). We set \( \text{VNd}[\text{Root}] = \text{Hgt}[\text{Root}] \). The wave starts at the root at time \( t = 0 \) and travels with constant speed \( \text{VNd}[\text{Root}] \) on the paths to the farthest leaves in \( VD \). (Due to our choice of the starting point \( p_0 \), there will always be at least two paths from \( p_0 \) to a leaf with the maximum length.) Hence, it reaches those leaves at time \( t = 1 \). On all the shorter paths, we make the wave slow down so that it reaches each other leaf at time \( t = 1 \). Generally, when the speed needs to be decreased along some path, we let the speed be a piecewise linear function of time, so that the first 25% of the path is used to slow down and the speed is thereafter constant.

When the movement of the wave is defined, we may define \( \text{GetPt}(e, t) \) as the point on edge \( e \) with \( \theta(\text{GetPt}(e, t)) = t \), where \( \text{TmNd}[\text{Start}[e]] \leq t \leq \text{TmNd}[\text{End}[e]] \).

2.5. Constructing the wavefronts

We make a spiral with \( r = \frac{TtWfLng}{\text{node}} \) revolutions, where \( \delta = 0.95 - \delta \). The number 0.95 is a hyper-parameter that we have found to work well in practice. We use the slightly smaller step-over \( \delta \) so that the maximum distance between two neighboring revolutions is smaller than \( \delta \). That gives more flexibility to smooth the spiral later on as described in Section 2.8.

Using a number closer to 0 instead of 0.95 results in more revolutions and thus a longer spiral, while a number closer to 1 increases the curvature of the resulting spiral, since there will not be as much freedom to smooth the spiral with circular arcs and at the same time respecting the stepover. We set \( \Delta = 1/r \) and compute a wavefront for each of the times \( \{0, 1/r, 2/r, \ldots, r\Delta\} \). The two-dimensional array \( Wf \) stores the wavefronts so that the wavefront at time \( \Delta \) is the array \( Wf[i] \). Wavefront \( i \) is constructed by traversing \( VD \) once and finding every point on \( VD \) with time \( \Delta i \) in counterclockwise order.

Let \( e \) be an edge we have not visited before, and let \( n = \text{Start}[e] \) and \( m = \text{End}[e] \). There is a corner of wavefront \( i \) on \( e \) if \( \text{TmNd}[n] < \Delta i \leq \text{TmNd}[m] \). If that is the case, we add \( \text{GetPt}(e, \Delta i) \) to the end of \( Wf[i] \). We also make one corner in \( Wf[0] \) for each of the child edges of the root, \( \text{ChildEs}[\text{Root}] \), and all these corners are copies of the point \( \text{Pt}[\text{Root}] \). Using this construction, there is exactly one corner on each wavefront on each path from \( \text{Root} \) to a leaf of \( VD \).

For each corner \( Wf[i][w] \), we store the length of the part of the wavefront up to the corner, i.e., \( WfLng[i][0] = 0 \) and \( WfLng[i][w] = \sum_{j} ||Wf[i][j] - Wf[i][j - 1]|| \), for \( w > 1 \). Here \( || \cdot || \) is the Euclidean norm. We also store the total length of \( Wf[i] \) as \( TtWfLng[i] \).

We introduce a rooted tree with the wavefront corners as the roots. See Fig. 3(a). The parent of a corner \( Wf[i][w], i > 0 \), is the unique corner \( Wf[i - 1][pWf[i][w]] \) on wavefront \( i - 1 \) on the path from \( Wf[i][w] \) to \( \text{Root} \). We store \( pWf \) as \( PaWf[i][w] \), i.e., the parent of \( Wf[i][w] \) is \( Wf[i - 1][PaWf[i][w]] \).

![Fig. 3.](image-url) The construction of a polyline spiral in a polygon P: (a) The wavefronts in green and blue arrows from each wavefront corner \( Wf[i][w] \) to its parent \( Wf[i - 1][PaWf[i][w]] \). The diagram \( VD \) is in gray. (b) The polyline spiral in black obtained by interpolating between the wavefronts. The purple arrows are from each corner \( Sp[i] \) of the spiral to its parent \( Sp[Pa[i]] \).
Since the distance from each wavefront corner to each of its children is at most \( \delta' \), we get that the distance from a point on one wavefront to the neighboring wavefronts is at most \( \delta' \). Furthermore, since the wave is moving with positive speed towards the leaves of \( VD \) and the faces into which \( VD \) divides \( P \) are convex, neighboring wavefronts do not intersect each other. From the order in which the corners of a wavefront are constructed, it is also clear that a wavefront does not intersect itself.

### 2.6. Interpolating between the wavefronts

We construct a polyline spiral stored as an array \( Sp \). For each \( i = 1, \ldots, r \), we construct one revolution of the spiral by interpolating between wavefront \( i-1 \) and wavefront \( i \). Every corner of the spiral is a point on \( VD \). There is exactly one spiral corner on the path in \( VD \) from each wavefront corner \( Wf[i][w] \) to its parent \( Wf[i][pw] \). Assume for now that we know how to choose the actual corners of the polyline spiral. We shall get back to this shortly.

The first corner \( Sp[0] \) is on the root node of \( VD \), and for every other corner \( Sp[s], s > 0 \), we store a parent index \( Pa[s] \), such that the parent \( Sp[Pa[s]] \) is the first corner we meet on the path in \( VD \) from \( Sp[s] \) to the root. Fig. 3(b) shows the resulting polyline spiral and the parents of each corner. We define the spiral such that the distance between a spiral corner and its parent is at most \( d_0 \). It follows that the distance from a point on the polyline spiral to the neighboring revolutions is at most \( d_0 \).

Here we describe how to define the \( Pa \)-pointers. When we have constructed a spiral corner \( Sp[s] \) which is on the path from \( Wf[i][w] \) to its parent wavefront corner \( Wf[i-1][pw] \), we know that the first spiral corner on the path from \( Wf[i][w] \) to the root is \( Sp[s] \), and we store this information as \( Pa[Sp[i][w]] = s \). Therefore, when we have made a new spiral corner \( Sp[r] \) on the path from \( Wf[i+1][w] \) to its parent \( Wf[i][pw] \), the parent of \( Sp[r] \) is defined to be \( Pa[r] = Pa[Sp[i][w]] \). By doing so, the corner \( Sp[Pa[r]] \) will be the first spiral corner on the path from \( Sp[r] \) to the root.

In the following we describe how to choose the corners of the polyline spiral. We assume that we have finished the revolution of the spiral between wavefronts \( i-2 \) and \( i-1 \) and we show how to make the revolution between wavefronts \( i-1 \) and \( i \). While constructing the revolution between wavefronts \( i-2 \) and \( i-1 \), we have stored for each wavefront corner \( Wf[i-1][w] \) the index...
Long edges on \( P \) lead to long faces in the Voronoi diagram so that the wave is not moving towards the boundary of \( P \) in a natural way. It may result in degenerate one-dimensional wavefronts. Therefore, we add extra edges from long edges of the Voronoi diagram to the boundary in the following sense. Let \( l_1 \) and \( l_2 \) be two leafs of \( \VD \) which are neighbors on \( \partial P \). Such a pair of nodes are on the same or on two neighboring corners of \( P \). Assume the latter, so that there is a segment \( S \) on \( \partial P \) from \( l_1 \) to \( l_2 \) and a face \( f \) of the Voronoi diagram to the left of \( S \). Let \( s = \sqrt{\|P_1[l_2] - P_1[l_1]\|^2} \) be the vector from \( l_1 \) to \( l_2 \), \( d = |s| \) be the length of \( S \) and \( m = \delta/d \). We want to subdivide \( f \) into \( m \) faces. Let \( p_i = P_1[l_1] + s \cdot i/m, i = 1, \ldots, m - 1 \), be interpolated points on \( S \). Let \( h_i = p_{i+1} - p_i \) be the half-line starting at \( p_i \) with direction \( s \), where \( s \) is the counterclockwise rotation of \( s \). For each \( i = 1, \ldots, m - 1 \), we find the first intersection point between \( h_i \) and the Voronoi diagram. Assume the intersection for some \( i \) is a point \( q \) on an edge \( e \). If the smallest angle between \( h_i \) and \( e \) is larger than 50°, we split \( e \) into two edges by introducing a node at \( q \) and add a segment from that node to a new node \( p_{i+1} \). If the smallest angle is less than 50°, the Voronoi diagram is moving fast enough towards the boundary so that the wavefronts will be fine in that area without adding any additional edges.

The other modification we perform has to do with the concave corners of \( P \), which are the corners where the inner angle is more than 180°. Each concave corner \( c \) on \( P \) leads to a face in the Voronoi diagram of all the points in \( P \) being closer to \( c \) than to anything else on the boundary of \( P \). Therefore, there are two edges \( e_1 \) and \( e_2 \) of the Voronoi diagram with an endpoint on \( c \). We have found that we get a better spiral if we remove these edges and instead add an edge following the angle bisector of the edges, i.e., we follow the bisector from \( c \) and find the first intersection point \( q \) with the Voronoi diagram and add an edge from \( q \) to \( c \). The reason that this process improves the resulting spiral is that the wavefronts will resemble \( P \) more because they will have just one corner corresponding to the corner \( c \) (on the bisector edge on \( P \)), instead of two (one on each of \( e_1 \) and \( e_2 \)). We can only do this manipulation if the resulting faces are also convex. That is checked easily by computing the new angles of the manipulated faces and it seems to be the case almost always.

### 2.8. Rounding the polyline spiral

In this section, we describe a possible way for smoothing the polyline spiral to get a \( C^1 \) continuous spiral. See Fig. 5 for a comparison between the rounded and unrounded spiral from Fig. 2 using this method. Note that some of the arcs of the rounded spiral round multiple corners of the polyline spiral.

![Fig. 5. The spirals from Fig. 2 together. The polyline spiral is blue and the rounded spiral is black.](image-url)
For each corner on the polyline spiral, we substitute a part of the spiral containing the corner with a circular arc which is tangential to the polyline spiral in the endpoints. That gives a spiral which is $G^1$ continuous, i.e., having no sharp corners. For each index $i$, let $s_i = S_{p[i]}[S_{p[i] + 1}]$ be the segment from $S_{p[i]}$ to $S_{p[i] + 1}$ and $v_i = S_{p[i] + 1} - S_{p[i]}$ be the vector from $S_{p[i]}$ to $S_{p[i] + 1}$. Each arc has the start point $p_i$ on some segment $s_i$ and the endpoint $q_i$ on another segment $s_b$, $a < b$, so that the arc substitutes the part of the polyline spiral from $p_i$ to $q_i$. We say that the arc rounds the corners $a + 1$ to $b$. We call the arc tangential if it is counterclockwise and its center is on the intersection of the half-lines $\overrightarrow{p_i q_i}$ and $\overrightarrow{q_i q_i'}$ or it is clockwise and its center is on the intersection of the half-lines $\overrightarrow{p_i q_i}$ and $\overrightarrow{q_i q_i'}$. It follows that if each corner is substituted by a tangential arc with positive radius, the resulting spiral is $G^1$ continuous.

We store a pointer $\text{Arc}[i]$ to the arc that substitutes the corner $S_{p[i]}$. The same arc can substitute multiple consecutive corners, so that $\text{Arc}[i] = \text{Arc}[i + 1] = \ldots = \text{Arc}[i + k - 1]$. In that case, when $\text{Arc}[i - 1] \neq \text{Arc}[i] \neq \text{Arc}[i + k]$, the neighbors of $\text{Arc}[i]$ are $\text{Arc}[i - 1]$ and $\text{Arc}[i + k]$. Two different arcs must substitute disjoint parts of the polyline spiral for the rounded spiral to be well-defined.

We subdivide each segment $s_b$ at a point $p_k \in s_b$ such that an arc ending at $s_b$ must have its endpoint at the segment $S_{p[k]}$ and an arc beginning at $s_b$ must have its startpoint on the segment $p_s S_{p[a + 1]}$. The point $p_k$ is chosen as a weighted average of $S_{p[a]}$ and $S_{p[a + 1]}$ so that the arc rounding the sharpest of the corners $S_{p[a]}$ and $S_{p[a + 1]}$ gets most space. Let $\phi_k \in (-\pi, \pi]$ be the angle at corner $S_{p[a]}$ of the polyline spiral. We set $\omega_k = \frac{\pi - \phi_k}{2 \pi - \phi_k}$ and choose $p_k$ as $p_k = (1 - \omega_k) \cdot S_{p[a]} + \omega_k \cdot S_{p[a + 1]}$.

We keep a priority queue $Q$ (Cormen, Leiserson, Rivest, & Stein, 2009) of the arcs that can possibly be enlarged. After each enlargement of an arc, the resulting spiral respects the stepover $d$ and no self-intersections have been introduced. We say that an arc with these properties is usable. Initially, we let each corner be rounded by a degenerated zero-radius arc, and $Q$ contains all these arcs. Clearly, these initial zero-radius arcs are usable by definition. We consider the front arc $A$ in $Q$ and try to find another usable arc $A'$ that substitutes a longer chain of the polyline spiral. The new arc $A'$ normally has a larger radius than $A$. If possible, we choose $A'$ so that it also substitutes one or, preferably, two of the neighbors of $A$. Assume that we succeed in making a usable arc $A'$ substituting both $A$ and its neighbors $B$ and $C$. Now $A'$ rounds the union of the corners previously rounded by $A$, $B$, and $C$. For all these corners, we update the Arc-pointers and we remove $A$, $B$, and $C$ from $Q$. We add $A'$ to $Q$. For every corner that $A'$ rounds, we also add the arcs rounding the children and parents of that corner to $Q$, since it is possible that these arcs can now be enlarged. If no larger usable arc $A'$ is found, we just remove $A$ from $Q$. The rounding process terminates when $Q$ is empty.

We test if an arc is usable by measuring the distance to the arcs rounding the child and parent corners. That is easily done using elementary geometric computations.

Here we describe how to find the largest usable arc rounding the corners $a + 1$ to $b$, $a < b$. We find the possible radii of tangential arcs beginning at a point on $p_s S_{p[a + 1]}$ and ending at a point on $S_{p[b]} p_b$ using elementary geometry. There might be no such arcs, in which case we give up finding an arc rounding these corners. Otherwise, we have an interval $[r_{\text{min}}, r_{\text{max}}]$ of radii of tangential arcs going from $p_s S_{p[a + 1]}$ and ending on $S_{p[b]} p_b$. We check if the arc with radius $r_{\text{min}}$ is usable. If it is not, we give up. Otherwise, we check if the arc with radius $r_{\text{max}}$ is usable. If it is, we use it. Otherwise, we make a binary search in the interval $[r_{\text{min}}, r_{\text{max}}]$ after the usable arc with the largest radius. We stop when the binary search interval has become sufficiently small, for instance $0.01 \cdot d$, and use the arc with the smallest number in the interval as its radius.

The order of the arcs in $Q$ is established in the following way: We have found that giving the arc $A$ the priority $P(A) = r(A)/r(\text{Cmax}) + 1/s(A)$ gives good results, where $r(A)$ is the radius of $A$, $s(A)$ is the size of the subtended angle from the center of $A$ in radians, and $r(\text{Cmax})$ is the radius of the maximum circle contained in $P$. The front arc in $Q$ is the one with the smallest $P$-value. We divide by $r(\text{Cmax})$ to make the rounding invariant when $P$ and $d$ are scaled by the same number. $r(\text{Cmax})$ can be obtained from the Voronoi diagram of $P$, since the largest inscribed circle has its center on a node in the diagram. If $s(A) = 0$, we set $P(A) = \infty$, since there is no corner to round.

We see that arcs with small radii or large subtended angles are chosen first for enlargement. In the beginning when all the arcs in $Q$ have zero radius, the arcs in the sharpest corners are chosen first because their degenerated arcs have bigger subtended angles –
even though the radius is zero, we can still define the start and end angle of the arc according to the slope of the segments meeting in the corner and thus define the subtended angle of the arc.

If two or three arcs are substituted by one larger arc each time we succeed in making a larger arc, we are sure that the rounding process does terminate, since the complexity of the spiral decreases. However, it is often not possible to merge two or three arcs, but only to make a larger arc rounding the same corners as an old one. The rounded spiral gets better, but we cannot prove that the process terminates. In practice, we have seen fast termination in any tested example. A possible remedy could be only to allow the process does terminate, since the complexity of the spiral we succeed in making a larger arc, we are sure that the rounding corner and thus define the subtended angle of the arc.

3. Computing a spiral in a pocket with an island

In many practical applications, the area to be machined is not simply-connected but has one or more “islands” that should not be machined. It might be because there are physical holes in the part or areas of a thicker layer of material not to be machined in the same depth. Therefore, assume that we are given a polygon \( P \) and a single polygonal island \( I \) in the interior of \( P \). In Section 5 we suggest a method to deal with multiple islands. By \( P \setminus I \), we denote the closed set of points which are in the interior or on the boundary of \( P \) but not in the interior of \( I \). We want to compute a spiral which is contained in \( P \setminus I \) such that the user-defined step-over \( \delta \) is respected between (i) two consecutive revolutions, (ii) the boundary of \( I \) and the first revolution, and (iii) the boundary of \( P \) and the last revolution. We also require that the spiral is \( C^1 \) continuous and has no self-intersections. See Fig. 6(b) for an example.

As in the case with a simply-connected pocket, we use a wave model to construct the spiral. We imagine a wave that has exactly the shape of \( I \) at time 0 and moves towards the boundary of \( P \), so that at the time 1, it has the shape of \( P \). We explain how to define a polyline spiral. The spiral should be rounded by a method similar to the one described in Section 2.8.

3.1. The Voronoi diagram of a pocket with an island

We use the Voronoi diagram of the set of line segments of \( P \) and \( I \). As in the case with no islands, we modify the diagram slightly. Let \( VD = VD(P \setminus I) \) be the modified polygon. Like the true Voronoi diagram, the modified diagram \( VD \) has the following properties:

1. \( VD \) is a connected, plane graph contained in \( P \setminus I \),
2. each leaf of \( VD \) is on the boundary of \( P \) or \( I \),
3. there is at least one leaf of \( VD \) on each corner of \( P \) and \( I \),
4. all the faces into which \( VD \) divides \( P \setminus I \) are convex,
5. \( VD \) contains exactly one cycle, the cycle is the locus of all points being equally close to the boundaries of \( I \) and \( P \), and \( I \) is contained in its interior.

The diagram \( VD \) is consisting of one cycle \( C \), which we also denote by the central cycle, and some trees growing out from \( C \), see Fig. 6(a). Each of the trees grows either outwards and has all its leaves on the boundary of \( P \) or inwards and has all its leaves on the boundary of \( I \). The general idea is to use the method described in Section 2 to define wavefronts in each of these trees separately. Afterward, we interpolate between the neighboring wavefronts in each tree and connect the interpolated pieces to get one contiguous spiral.

As in the case of a polygon without an island, we enrich the Voronoi diagram and remove double edges going to concave corners as described in Section 2.7.

We want the trees to be symmetric in the sense that there is a tree \( PT_n \) with root \( n \in C \) and leaves on the boundary of \( P \) if and only if there is a tree \( IT_n \) with root \( n \in C \) and leaves on the boundary of \( I \). If for a node \( n \in C \) we only have one of the trees, say \( IT_n \), we add an edge from node \( n \) to the closest point on the boundary of \( P \) and let \( PT_n \) be the tree consisting of that single edge. It follows from the properties of the Voronoi diagram that the added edge does not intersect any of the other edges.

We store \( C \) as a vector \( \{n_0, \ldots, n_{c-1}\} \) of the nodes on \( C \) in clockwise order, such that there are trees \( IT_n \) and \( PT_n \) for each \( i = 0, \ldots, c - 1 \). A root node is a node \( n \in C \). We let \( T_n = PT_n \cup IT_n \) be the union of the two trees rooted at node \( n \in C \) and consider \( T_n \) as a tree rooted at node \( n \).

3.2. Defining the movement of the wave

On each tree \( T_n \) we now define a wave model similar to the one described in Section 2.1. The wave starts at time \( t = 0 \) on the leaves on the boundary of \( I \) and moves through \( T_n \) so that it hits the leaves on the boundary of \( P \) at time \( t = 1 \). We compute the wavefronts from \( n \) and towards \( I \) and \( P \).

Let the preferred time of a root node \( n \) be \( t_n = \frac{1}{\text{IslHgt}[n] + \text{BndHgt}[n]} \), where \( \text{BndHgt}[n] \) and \( \text{IslHgt}[n] \) is the length of the longest path
to a leaf in $PT_n$ and $IT_n$, respectively. Similarly, if the time $TmNd[n]$ has already been defined, we let the preferred speed of $n$ be $v_n = \max\left\{ \frac{t_{\max}}{TmNd[n]}, \frac{d}{1 - TmNd[n]} \right\}$.

A naive method to define the times and speeds of a root $n$ is to set $TmNd[n] = t_n$ and $VelNd[n] = v_n$. That will minimize the number of revolutions and give the most equidistant wavefronts on each tree $T_n$. However, the abrupt changes in time and speed along the central cycle $C$ results in a spiral which curves a lot. Instead, we might smooth the times and speeds in the root nodes around $C$. This can be done in many different ways, and we shall leave the details for the reader. See Fig. 7.

3.3. Creating wavefronts

For a given root node $n \in C$, we want at least $s_n = \frac{t_{\max}}{d}$ revolutions of the spiral in the tree $IT_n$ in order to respect the stepover $\delta' = 0.95 \cdot \delta$. Similarly, we want $s_n = \frac{d}{1 - TmNd[n]}$ revolutions in $PT_n$. Therefore, the time between two revolutions should be at most $\Delta_n = \min\left\{ \frac{t_{\max}}{s_n}, \frac{d}{s_n} \right\}$. Hence, we let $\Lambda = \min_{n \in C}\{\Delta_n\}$ be the minimum over all such values. We let the number of revolutions be $r = \lceil 1/\Lambda \rceil$ and set $\Delta = 1/r$.

Each tree $T_n$ contains an interval of the corners of each wavefront $i$. The corners of the subset are points on $IT_n$ if $t \leq TmNd[n]$, otherwise they are on $PT_n$. Let $r_n = \lceil \frac{t_{\max}}{d} \rceil$. The wavefronts $i = 0, \ldots, r_n$ are on $IT_n$, while wavefronts $i = r_n + 1, \ldots, r$ are on $PT_n$. As explained in Section 3.2, we do not use the same time $TmNd[n]$ for every node $n$ on $C$. Therefore, we may get wavefronts crossing $C$ and wavefronts crossing each other. We shall later explain how to avoid that the polyline spiral has self-intersections.

We suggest to store the wavefront corners in a two-dimensional array for each of the trees $IT_n$ and $PT_n$. The wavefront corners on $IT_n$ of one wavefront are stored in an array $Isl\, Wf[i][j]$, where index $j$ corresponds to wavefront $i = r_{n - j + 1}$. In $PT_n$, the array $Bnd\, Wf[i][j]$ stores corners of wavefront $i = j + r_n$. Hence, the corners of each wavefront is stored locally in each tree $T_n$.

In $IT_n$, the parents of the corners of wavefront $i = 0, \ldots, r_n - 1$ are corners of wavefront $i + 1$. In $PT_n$, the parents of the corners of wavefront $i = r_n + 2, \ldots, r$ are corners of wavefront $i - 1$. Therefore, all parents are on the wavefront one step closer to the root $n$. In both $IT_n$ and $PT_n$, we introduce fake wavefront corners at the root $n$ stored in the arrays $Isl\, Wf[n][0]$ and $Bnd\, Wf[n][0]$, respectively, which are the parents of the corners in the arrays $Isl\, Wf[n][1]$ and $Bnd\, Wf[n][1]$. Thus, these fake corners are not corners on wavefront $i$ for any $i = 0, \ldots, r$, but are merely made to complete the tree of parent pointers between corners of neighboring wavefronts.

We also need an array $Wf\, Ln$ containing global information about the length of each wavefront crossing all the trees $\{T_n\}$ in
order to do interpolation between the wavefronts later. We have \( \text{WfLn}[i][0] = 0 \) for every wavefront \( i \). If \( c_m \) and \( c_{m+1} \) are the \( m \)'th and \( (m+1) \)'st corners on wavefront \( i \), respectively, we have \( \text{WfLn}[i][m] = \text{WfLn}[i][m] + \| c_{m+1} - c_m \| \). Notice that \( c_m \) and \( c_{m+1} \) can be corners in different, however neighbouring trees \( T_n \) and \( T_m \) and hence stored in different arrays. \( \text{TtlWfLn}[i] \) stores the total length of wavefront \( i \).

### 3.4. Interpolating between wavefronts

We interpolate between two wavefronts \( i - 1 \) and \( i \) in each tree \( T_n \) separately, but using the same technique as in Section 2.6. If \( i \leq r_n \), we interpolate between the wavefront fragments stored in \( \text{IslWf}[n][j] \) and \( \text{IslWf}[n][j+1] \), where \( j = r_n - i + 1 \), using the values of the length of wavefront \( i - 1 \) stored in \( \text{WfLn}[i-1] \). If \( j > r_n + 1 \), we interpolate between \( \text{BndWf}[n][j-1] \) and \( \text{BndWf}[n][j] \), where \( j = i - r_n \), using the values stored in \( \text{WfLn}[i] \). A special case occurs when \( i = r_n + 1 \), i.e., when we are interpolating between the first wavefront on each side of the root node \( n \). In that case, let \( t = (i-1)\Delta + \frac{\text{WfLn}[i][0]}{\text{TtlWfLn}[i]} \Delta \), when \( \text{BndWf}[n][1][0] \) is the \( m \)'th corner on wavefront \( i \). If \( t \leq \text{TmNd}[n] \), we interpolate between \( \text{IslWf}[n][0] \) and \( \text{IslWf}[n][1] \). Otherwise, we interpolate on the other side of \( C \), that is, between \( \text{BndWf}[n][0] \) and \( \text{BndWf}[n][1] \). The convexification process described in Section 2.6 can be used in each tree \( T_n \) separately.

We may store the interpolated spiral in a one-dimensional array \( \text{Sp} \). Before we add the first interpolated revolution to \( \text{Sp} \), i.e., the one between wavefront 0 and 1, we add wavefront 0 to \( \text{Sp} \), that is, all the corners of \( I \). Likewise, after the final revolution between wavefronts \( r-1 \) and \( r \), we add wavefront \( r \), which is all the corners of \( P \). These are used to ensure that the distance from the first and last revolution to the boundaries of \( I \) and \( P \), respectively, does respect the stepover when rounding the spiral.

For every corner \( \text{Sp}[s] \) we have a pointer \( \text{Pa}[s] \) such that \( \text{Sp}[\text{Pa}[s]] \) is the first spiral corner we meet when traveling from \( \text{Sp}[s] \) to the root \( n \) of the tree \( T_n \) containing \( \text{Sp}[s] \). The parents are not defined for the spiral corners closest to the root node \( n \). Therefore, make parent dependencies across \( C \) such that the parent nodes are on the same side of \( C \) as \( I \). We want all the parent pointers to be towards the island \( I \). Therefore, we reverse all the pointers between pairs of corners in each tree \( IT_n \). Now, the parent pointers are defined for all spiral corners except for the ones on the boundary of \( I \). See Fig. 8 for an example of a polyline spiral around an island and red arrows indicating the parent pointers. Notice that a corner can have multiple parents, but the parents are consecutive and can thus be stored using two indices.

---

**Fig. 9.** The same pocket and island as in Fig. 7. The cycle \( C \) is blue, the other edges of \( \mathcal{D} \) are gray. The interpolated spiral is black, and we have not introduced extra spiral corners on \( C \) to avoid self-intersections. In order to make the intersection problems appear, we have not smoothed the times and speeds around \( C \) as described in Section 3.2, but merely used the preferred time and speed of each node. (b) is a close-up of (a) of the area in the red rectangle. In (c), we have introduced new corners on \( C \) when the spiral jumps from one side of \( C \) to the other when the union of the two faces on each side of \( C \) is not convex.
In Section 2.6, we stated that since there are no intersections between different wavefronts, the polyline spiral has no self-intersections when there is no island in \( P \). The wavefronts do not intersect in that case due to the convexity of the faces into which the diagram \( VD(P) \) subdivides \( P \). When there is an island \( I \) in \( P \), there are two kinds of faces into which \( VD(P \setminus I) \) subdivides \( P \setminus I \), see Fig. 9. Some faces, like \( a_1 \) in the figure, are bounded by edges of one tree \( T_n \) while some are between two trees, like \( a_1 \) and \( a_2 \). The latter kind is bounded by edges of two neighboring trees \( T_n \) and \( T_m \) and an edge \( e \) from \( n \) to \( m \) on \( C \), where \( n \) and \( m \) are neighboring nodes on \( C \). The first kind of faces is similar to the faces in Section 2, so here we do not worry about self-intersections of the spiral. The second, however, can lead to wavefronts crossing each other and therefore also a self-intersecting spiral as is the case in the figure when the spiral jumps over \( C \) and crosses \( a_2 \). If the union of the faces on each side of \( e \) is convex, like \( a_1 \) and its neighboring face on the other side of \( C \), there is no problem. It can easily be tested if the union of the faces is convex by considering the angles of the union at nodes \( n \) and \( m \). If it is not, we introduce a new corner on the edge on \( C \) whenever the spiral jumps from one side of \( C \) to the other. The new corner is an interpolation of nodes \( n \) and \( m \) using the time of the spiral in the last corner in tree \( T_n \). Fig. 9(c) shows the result of introducing the extra corners. Our experience is that these intersection problems occur very rarely when the times and speeds have been appropriately smoothed around \( C \). Our experience is that these intersection problems occur very rarely if the time and speed of the wave have been smoothed around \( C \) as described in Section 3.2.

4. The skeleton method for a spiral in a pocket without islands

The method from Section 2 is mainly applicable if the polygon \( P \) is not too far from being a circle. If \( P \) is very elongated or branched, the distance between neighboring revolutions will often be much less than the maximum stepover. Therefore, the tool path will be unnecessarily long and the cutting width will vary a lot. That leads to long machining time and an uneven finish of the part. See Fig. 10 for an example. In such cases, we construct a skeleton in \( P \), which is

![Fig. 10. Comparison of the basic spiral method from Section 2, figure (a), with the improved skeleton method from Section 4, figure (b). Note that the spiral obtained from the skeleton method is significantly shorter and that the distance between neighboring revolutions is varying much less than when using the basic method.](image)

![Fig. 11. (a) The diagram \( VD(P) \) of the polygon \( P \) from Fig. 10 in green, where the edges chosen for the skeleton are black. (b) The diagram \( VD(P \setminus I) \) of \( P \) with the skeleton considered an island \( I \). The cycle \( C \) is blue and the remaining edges are green. The resulting spiral from Fig. 10(b) is included in gray.](image)
an island $I$ with zero area. We then use the method from Section 3 to make a spiral from the island to the boundary. It does not matter for the construction of the spiral that the island $I$ has zero area. See Fig. 10 for a comparison between the basic method from Section 2 with the skeleton method described here when applied to the same polygon.

We choose the sketch as a connected subset of the edges of the diagram $VD = VP(P)$. We traverse $VD$ once starting at the root and decide for each edge whether to include it in the skeleton. If an edge from node $n$ to $m$ is not included, we do not include anything from the sub-tree $VD[m]$. For any node $n$, let $d(n)$ be the length of the shortest path from $n$ to a leaf in $VD[n]$, and let $D = \max_{m \in VD} d(n)$. We have found that the following criteria for including an edge $e$ from node $n$ to $m$ gives good results. We require all the criteria to be satisfied.

1. The longest path from $n$ to a leaf in $VD[n]$ goes through $m$ or $l[e] + H_l[m] \geq 1.5 \cdot D$, where $l[e]$ is the length of edge $e$.
2. The length of the spanned boundary of $m$ is larger than $2 \cdot D$.
3. $H_g[m] \geq D$.

Criterion 1 is to avoid getting a skeleton that branches into many short paths. Therefore, we only make a branch which is not following the longest path from $n$ if it seems to become at least $0.5 \cdot D$ long (taking criterion 3 into account). When criterion 2 fails, it seems to be a good indicator that an edge is not a significant, central edge in $VD$, but merely one going straight to the boundary. Criterion 3 ensures that we do not get too close to the boundary. If we did, we would get very short distances between the neighboring revolutions there. If criterion 3 is the only failing criterion, we find the point $p$ on $e$ such that $H_g[m] + \|p - p + m\| = D$ and include the edge from node $n$ to $p$ in the skeleton.

If the polygon is close to being a circle, the method described here results in a very small skeleton, and we get a better spiral using the basic method from Section 2 in that case. This can be tested automatically by falling back to the basic method if the circumference of the skeleton is less than, say, 5% of the circumference of $P$.

5. Computing a spiral in a pocket with multiple islands

The method from Section 3 works only for polygons with a single island. If there are many islands $I_0, \ldots, I_{n-1}$ in a polygon $P$, we may connect them with bridges in a tree structure to form one big connected island, see Fig. 12. The basic idea of reducing the number of islands by connecting them is also used by Chuang and Yang (2007) and Held and Spielberger (2014). Our method is a variation of the minimum spanning tree algorithm of Dijkstra (1959). We choose the bridges as edges in the Voronoi diagram of the area $P \setminus \bigcup_{i=0}^{k-1} I_i$. The algorithm creates an array of the edges to use as bridges. We keep a growing set $s$ of the nodes of the Voronoi diagram that we have connected by bridges so far. We first find one central node $n_0$ and $s$ is only containing $n_0$ in the beginning. We use Dijkstra’s algorithm to make all shortest paths from nodes in $s$. When we reach an island $I_i$ whose corners are not in $s$, we use the shortest path to that island as a bridge and add the nodes on the shortest path and the corners on $I_i$ to $s$. We implement the algorithm so that bridges starting at nodes close to $n_0$ are preferred over nodes far from $n_0$. That makes the bridges grow from the center node $n_0$ out in every direction. If we did not choose the bridges in this careful way, the resulting connected island would possibly make unnecessarily long dead ends that would require many revolutions to fill out by the spiral.

6. Conclusion

We have described methods for the computation of spiral tool paths suitable for many shapes of pockets for which no previously described algorithms yield equally good results. Our main contribution is the new possibility of making a spiral morphing an island in a pocket to the boundary of the pocket.

Our algorithms work under the assumption that the input is polygonal. An obvious improvement is to generalize to input consisting of line segments and circular arcs, as done by Held and Spielberger (2009), using ArcVRONI by Held and Huber (2009) to compute the Voronoi diagrams for such input.

Further work could be focused on developing a hybrid method of the skeleton method and the method for handling one or more islands. Also, one could work on automatic subdivision of complex pockets into pockets more suitable for spiral machining. Given the new techniques described in this paper, this problem seems...
different from the one handled by Held and Spielberger (2014) and Patel and Lalwani (2017).

Conflict of interest

None.

Acknowledgments

I would like to thank my colleague at Autodesk, Niels Woo-Sang Kjærsgaard, for numerous helpful discussions while I did the development of the spiral algorithms and for many useful comments on this paper. My thanks also go to Autodesk in general for letting me do this research and publish the result.

References


