Int J Adv Manuf Technol (2011) 53:325–338
DOI 10.1007/s00170-010-2817-x

ORIGINAL ARTICLE

Constant cusp toolpath generation in configuration space based on offset curves

Steffen Hauth · Claus Richterich · Lothar Glasmacher · Lars Linsen

Received: 4 January 2010 / Accepted: 24 June 2010 / Published online: 14 July 2010

Abstract Constant cusp is a common strategy for generating tool paths in many NC machining applications. Cusps need to be regulated to ensure high precision without wasting machining efforts. Constant cusp strategies frequently operate on NURBS surfaces or triangular meshes and, thus, have to deal with the issues of patch-boundary oscillations or long, stretched triangles. To avoid these issues, one can operate in a pre computed configuration space (c-space). The c-space is given in form of a regular quadrilateral heightfield mesh, which may be adaptively subdivided, where the slope is large. This simple data structure is memory efficient and is widely used in CAD/CAM frameworks. In this paper we introduce an algorithm for creating a constant cusp tool path with the help of a given c-space. The constant cusp algorithm iteratively produces curves in the c-space by fitting a tube around the current curve and intersecting the tube with the c-space mesh to detect the subsequent curve. As tool paths are handed to the machine controller in form of point sequences, it suffices to operate on piecewise linear curves. The tube becomes a concatenation of cylinders, which we derive using geometric considerations. In each iteration of the constant cusp algorithm, intersection points of the cylinders with the not yet traversed part of the mesh are detected and checked for their validity. The validity check can efficiently remove global or local self-intersections of the new curve by just deleting the respective points. In a final step, the detected intersection points are connected to form constant cusp tool paths. Dealing with piecewise linear curves, we achieve low computation times for real-world data sets.

Keywords Configuration space · Constant cusp · Offset curves · Tool path

S. Hauth (✉) · L. Linsen
School of Engineering and Science, Jacobs University Bremen gGmbH, Bremen, Germany
e-mail: s.hauth@jacobs-university.de

L. Linsen
e-mail: l.linsen@jacobs-university.de

C. Richterich
ModuleWorks GmbH, Aachen, Germany

L. Glasmacher
Fraunhofer Institut für Produktionstechnologie, Aachen, Germany

1 Introduction

In modern manufacturing practice, computerized numerical control (CNC) machine tools are commonly used to produce a variety of functioning surfaces and molds. CAD/CAM software tools are essential for generating free form surfaces and tool paths for the desired application (milling, grinding, polishing, finishing, etc.). Several strategies for generating tool paths have been developed [2, 9, 14, 17, 25]. One important strategy, especially for finishing applications, is the constant cusp approach [2, 14, 17]. Constant cusp algorithms are based on algorithms for creating offset curves. These algorithms for offset curves are difficult to implement due to the particular properties of offset curves. For instance, when computing an offset curve to an original curve, the exact offset curve has a significantly higher degree. Also, offset curves are, in general, not rational anymore [13, 22]. Furthermore, local and global self-

intersections can appear, which are typically hard to detect and to resolve [10, 33].

The nomenclature in this field is not unique and differs depending on the authors. In this paper we use the term *cusp* rather than the term *scallop* for the material leftover. Constant cusp strategy denotes a processing strategy, which results in a workpiece with constant material leftover over the whole surface. Constant cusp correlates with a constant *step over*, which is the distance between adjacent parts of the tool path. Figure 1 illustrates the terms. If the step over is sufficient small, the tool path results in a constant cusp tool path. Our experiments show, that the step over should be smaller than the radius of the tool to have adequate results.

Researchers worked for years on constant cusp strategies (see Section 2), but still not all commercial CAD/CAM-Software have an implementation of a constant cusp algorithm. Regarding a study from CIMdata, Inc. [6], the two widely used CAx programs are MasterCam and NX Unigraphics, and these programs do not provide a constant cusp strategy. However, a plugin exists, which provides constant cusp strategy for MasterCam. This appears to be caused by the fact that the existing approaches are not flexible enough, not fast enough, not stable enough, or very hard to implement. In this paper, we present a new algorithm for constant cusp strategy, which is easy to implement and flexible in terms of the geometry of the workpiece.

Choi et al. [3] proposed to operate in a configuration space (c-space). The c-space represents all valid configurations of the 3-axis machining tool by describing the positions of the tool tip. Note that the tool tip is not necessarily the contact point of the tool with the workpiece such that the c-space is not identical to a description of the workpiece's surface. The c-space is neither represented by triangles nor by non-uniform rational b-spline (NURBS) but by a quadrilateral mesh, whose structure may be adaptively refined, if their slope is too steep. However, the c-space is typically generated by taking a NURBS or triangular mesh surface representation as an input and applying the respective conversion. In Section 3, we give a detailed description of the c-space representation. This representation has the advantage of being simple and allowing for memory and time efficient implementations which makes it amenable for large objects. Furthermore, the conversion to c-space eliminates the disadvantages of operating directly on NURBS surfaces and triangular meshes. NURBS surfaces have the disadvantage that the modeled surfaces, although they might look beautiful to the designer, often exhibit some discrepancies where patches come together leading to oscillations in the surface. Triangular meshes suffer from the disadvantage that they frequently contain long, stretched triangles leading to artifacts in the tool paths. Hence, the conversion to and operating in c-space makes our approach robust against bad properties in the representation of the input surface.

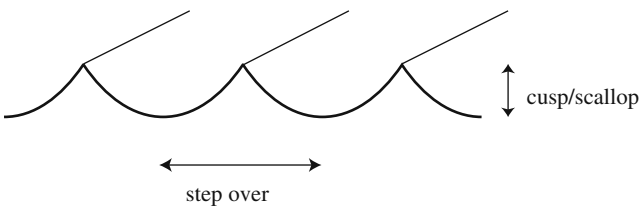


Fig. 1 Side view of the workpiece after applying a constant cusp strategy with a ball end tool. After the process the workpiece provides not a plane surface, but the typically cusp pattern. The height of the material leftover is called *cusp* or *scallop* and the distance between adjacent parts of the tool path is called *step over*. In this paper, we use the term cusp rather than scallop

State of the art CNC machines use for processing the so called *G1* Mode, here the CAD/CAM software provides the tool path as a piecewise linear curve to the CNC machine [8, 37–40]. In this respect the maximum length of the single linear pieces depends on the tool, its diameter, and the desired accuracy of the workpiece. The modes *G60* and *G64* are implemented in the CNC machines for smoothing the provided piecewise linear tool path [8, 34]. Due to this fact, it is a common and memory efficient strategy to use an (adaptive) quadrilateral mesh to represent the c-space. Operating in this c-space answers the question of how to discretize the curves. We only consider the intersecting points of the curve with the (adaptive) quadrilateral mesh and obtain a piecewise linear curve.

Unlike in existing approaches for constant cusp tool paths, we calculate the tool path in c-space. This guarantees valid position for the tool and makes gouge and collision detection obsolete. To compute offset curves to a given (discretized) curve, we put a cylinder with its radius being the given offset around each curve segment, stitch the cylinders together to form a tube around the curve, and intersect the c-space with the tube. Connecting the intersection points results in the next curve. To assure that the intersection points are valid, we restrict our search to valid regions only. This approach is not only fast and memory efficient, but it also allows for a simple detection and removal of local and global self-intersections of the offset curve. Iterating this process to cover the entire c-space of the workpiece provides the desired tool paths. We give a detailed description of the algorithm in Section 4, discuss results in Section 5, and discuss the c-space in Section 6.

2 Related work

Researchers have worked for years to improve the algorithms for creating offset curves. In the 1980s, researchers tried to avoid the problem of high degree to approximate either the curve itself or its offset curve by a curve with a lower degree [7, 15, 16, 18, 32]. As it became hard to improve the state of the art algorithms, the field of research turned more theoretical. Farouki et al. discussed the fundamental difficulties of exact offset curve computation [11, 12].

Another problem is to solve the global and local self-intersections of offset curves [10, 22, 33]. Lee et al. [22] applied a plane sweep algorithm to detect all global and local self-intersections of planar offset curves. However, this approach is hard to implement. Elber et al. [10] located self-intersections by checking whether the tangent field of the starting curve and its offset curve have opposite directions. This approach is limited to detecting local self-intersections. Seong et al. [33] proposed an algorithm for trimming global and local self-intersections by using distance maps.

Algorithms for creating constant cusp tool paths or implementations for offset curves algorithms can be classified by the surface representation: algorithms that work on surfaces represented by NURBS and algorithms that work on triangular meshes [1, 26, 35]. Mitchell et al. [26] introduced the first method for creating offset curves on triangular meshes. Further research studies tried to improve the implementation of this algorithm [1, 35].

Both kinds of algorithms have disadvantages if the purpose is to apply the algorithm to a wide field of applications: NURBS surfaces generated by CAD programs frequently exhibit small oscillations between single simple components. They are so small that designers barely see them. However, they are a big problem for the algorithms. For triangular meshes, triangles with one angle being small can occur. Those stretched triangles may cause bad numerical errors [1, 28]. Although those problems are well known in industry, the problem is not well documented in literature. This problem, however, makes it hard to apply existing algorithms in CAx systems.

The majority of constant-cusp algorithms in literature concentrate on constant-cusp only and not on the shape of the tool path, which results in zigzag shaped tool paths [20, 23, 24, 36]. Park and Choi [5] introduced a constant cusp algorithm for pockets and improved it in later publications [30, 31]. This algorithm takes the countour of the pocket into account and creates a non-zigzag tool path. Due to the fact that the algorithm is designed for pockets only and hence the algorithm creates only a 2D tool path, the field of its application is limited. Lee developed an algorithm, which used the offset curve approach for creating a constant-cusp tool path, with the common problem of local and global self-intersections [21]. Yuwen et al. proposed to use a harmonic parametrisation of a triangular surface for creating iso-parametric tool paths [41]. Since it is not possible to guarantee a perfect harmonic parametrization, it results not always in a contour parallel constant cusp tool path.

Choi et al. proposed the idea to transfer the method of configuration space (c-space) for robot path planning to CNC machine tool path generation [3, 4]. How exactly a c-space for tool path generation is built is not documented in literature. One has to transfer the methods for robot path planning, e.g., the state-of-the-art method by Kavraki [19], to tool path generation. Kavraki takes a discretized representation of the moving object—here the tool—and the obstacles—here the workpiece—and uses the discrete convolution operation in combination with the fast Fourier transform to determine the c-space. To our knowledge, no approach exists that describes how to create a constant cusp tool path in c-space, neither a zigzag nor an offset approach. We propose such a constant cusp strategy in c-space based on offset curves.

3 C-space

A set $\mathcal{Q}$ is called a configuration space (c-space) for a system, if every element of $\mathcal{Q}$ corresponds to a valid configuration of the system and each configuration of the system can be identified with a unique element of $\mathcal{Q}$ [27].

A *valid configuration* in our case refers to a valid position of the tool, described by the tool location, where the tool is in the desired position. The c-space approach provides valid configurations for tool paths. Hence, by creating a tool path in the c-space instead of on the designed surface, the position of the tool tip in the c-space offers an accurate orientation and a correct tool contact point [3]. Figure 2 illustrates the difference (gray) between the c-space and the designed surface, i.e., the workpiece (white). Here, the c-space is the top of the grey shaded part and the space above it. For 3-axis machining the c-space can be represented as a height field over the workpiece (see Fig. 3).

Compared to classical approaches, expensive gouge checks to adjust the tool path for complex surfaces are obsolete. In Fig. 4 an example illustrates the benefit. Consider, in a first processing step a tool is used with a large diameter, which cannot reach all parts of the sur-

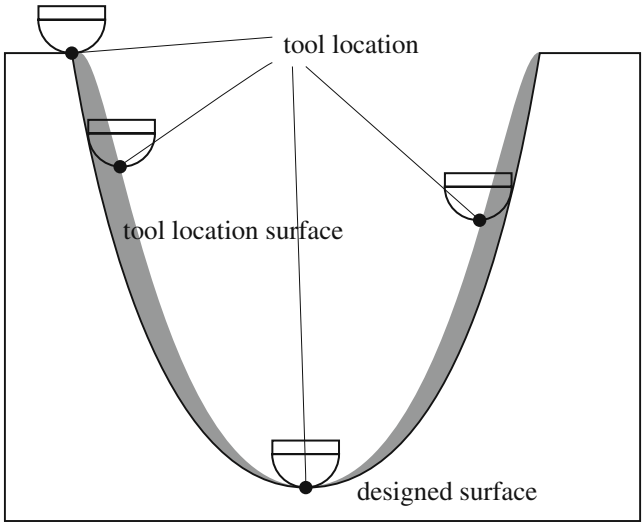


Fig. 2 Designed surface and c-space created for a ball end tool. The example positions show the ball end tool for 3-axis processing

face. Creating tool paths direct on the surface may lead to unreachable positions, which require corrections.

One can create the c-space by discretizing—with respect to the desired tolerance—the tool and the workpiece and convolute the tool kernel with the workpiece. The discretizing makes it possible to use the fast fourier transformation for the convolution operation [19].

The input of the control unit of a state of the art NC machine is a piecewise linear curve or a sequence of discrete points [37–40]. As a consequence, there is no need to create continuous smooth curves for a tool path, since they would be discretized anyway to make it readable for the control unit. This observation gives the motivation for operating on a discrete representation of the c-space. We store the c-space as a height field mesh in a quad tree. If the differences of the z values of the corners of a quadrilateral is larger than a threshold, we subdivide the quadrilateral into four sub quads.

Figure 5 shows a side view of a discrete c-space. The discrete points sample the c-space on a regular grid. If the z value of two adjacent points is larger than a threshold, we locally subdivide the regular grid such that the mesh can be stored in a quad tree data structure. Figure 6 shows an example.

The discrete representation of the c-space is memory efficient. Hence, implementations of our algorithm can operate on large models by using standard computer hardware. Furthermore, since we discretize the tool path by considering only points of the tool path that intersects the mesh, the algorithm becomes faster.

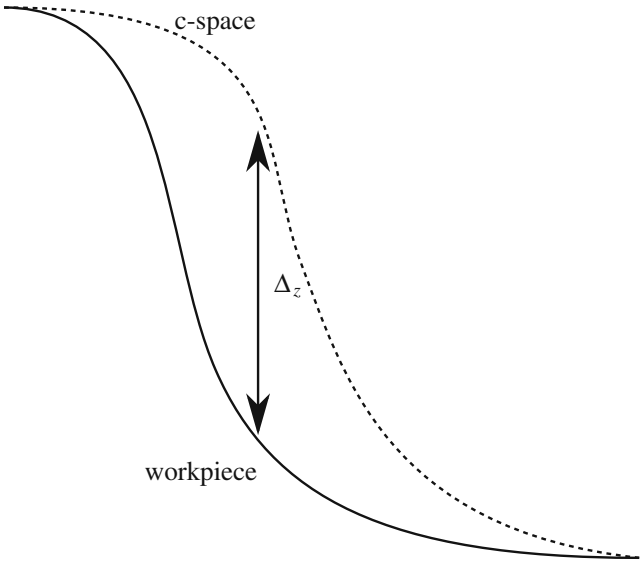
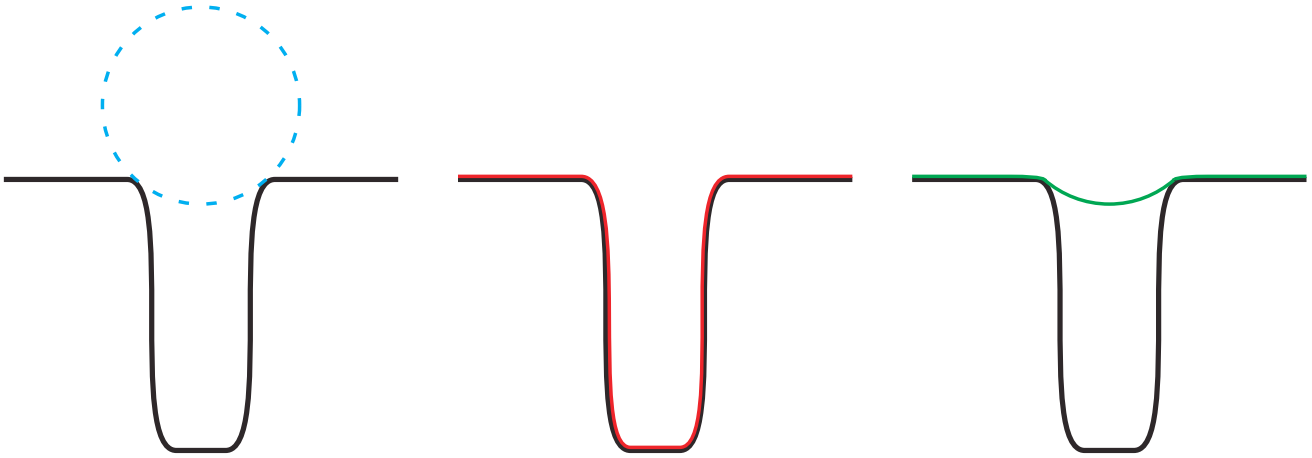


Fig. 3 The c-space (*dashed line*) can be represented by a height field over the workpiece (*solid line*)

4 Algorithm

Our algorithm for generating constant cusps on a c-space iteratively generates curves in a piecewise linear representation. The curves are approximations to offset curves of a given starting curve and generated by calculating the intersection between a tube with constant user given radius around the starting curve and the mesh. In observations from our experiments we figured out, that the radius of the tube should be smaller than the half of the radius of the tube to achieve good results. Hence, if the user desires a larger radius for the tube, for generating the next curve intermediate steps are required. The starting curve is a closed intersection free curve, which is projected onto the net. In general it is the contour of the mesh, but it also can be any user given curve. Initially, the part of the mesh to the curve's interior is set as valid and the exterior as invalid. In each iteration, an offset curve to the most recently generated curve (or the starting curve) is being added. Recapitulatory, each iteration consists of three steps: chopping the c-space, intersecting the chopped c-space with a tube around the curves, and connecting the appropriate intersection points.

The c-space is given in form of an adaptively refined quadrilateral mesh and the starting curve is given as a polygonal line whose vertices lie on the edges of the quadrilateral mesh. Intersecting the quadrilateral mesh with a tube around the curve results in intersection points on both sides of the curve. Since it is desired to have intersection points only on the inside or outside of



(a) A tool (illustrated by the dashed circle) with a large diameter cannot process the narrow part of the workpiece.

(b) Classical approaches without c-space first create the toolpath on the workpiece, although it cannot be processed as calculated,...

(c) ...and correct the toolpath with an expensive gouge check afterwards.

Fig. 4 Sketch of a 2D sideview of an workpiece example with a narrow hole to illustrate the benefit using a c-space. Consider a tool, which diameter is larger than the diameter of the hole (see Fig. 4a). A classical aproach without c-space creates first a tool path on the surface (see Fig. 4b) and correct it with gouge check afterwards to the correct tool path (see Fig. 4c). This example illustrates a case, where it is very expensive to correct the tool path with gouge checking. Calculating the tool path in c-space make this expensive step obsolete

the curve in each iteration of our algorithm we have to modify the mesh, such that only points on the desired side are computed. Hence, we trim the mesh such that only the desired part of the mesh remains.

Around each linear piece of a curve a surrounding tube reduces to a cylinder. For piecewise linear curves, the tube can be built by appropriately stitching the respective cylinders together. These cylinders are used for computing the intersections with the trimmed mesh.

For simplicity, we explain the three steps of our algorithm on a uniform quadrilateral mesh. For adaptively refined quadrilateral meshes the steps are analogous.

Our algorithm calculates offset curves to an initial curve with an offset of $2d$ between adjacent curves. Assuming a ball end tool on a plane surface. The distance

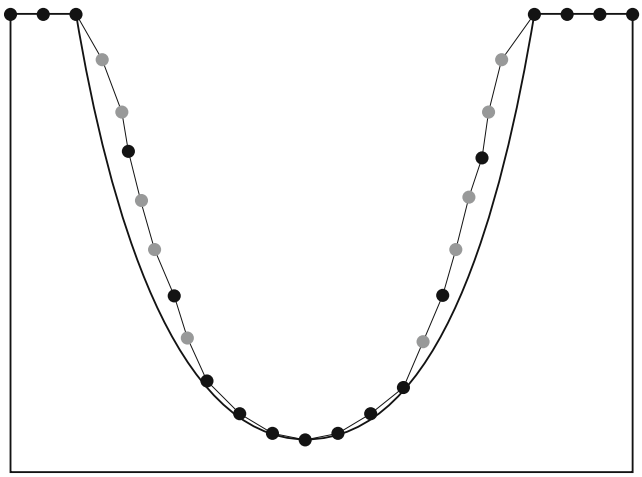


Fig. 5 Side view of c-space representation. The c-space is represented by discrete points lying on a regular grid connected to a mesh. If the z value of two adjacent points is too large, we insert further points in between. The points on the regular grid are *colored black* and the inserted points are *colored gray*

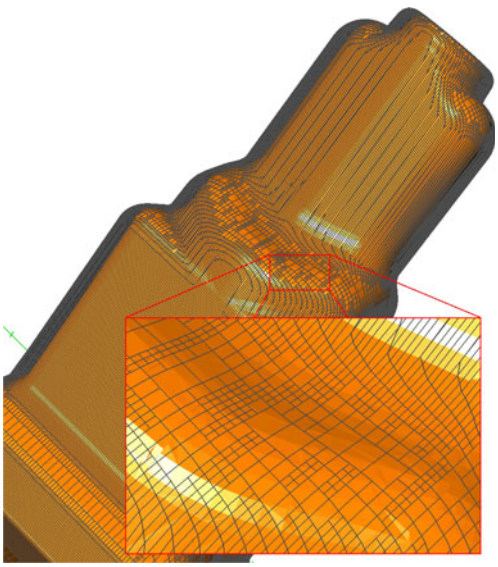


Fig. 6 C-space represented by an adaptive quadrilateral mesh

between the tool tip and the cusp vertex, projected to the surface, can be described as

$$d = r\sin(\alpha),$$

where r is the radius of the tool and α is the angle between the cusp vertex, the tool center point and the tool tip (see Fig. 7). The height of the cusp can be described by

$$s = r - r \cdot \cos(\alpha).$$

Using

$$\sin^2 + \cos^2 = 1$$

results in

$$s = r - r \cdot \sqrt{1 - \frac{d^2}{r^2}},$$
$$= r - \sqrt{r^2 - d^2}.$$

and

$$(r - s)^2 = r^2 - d^2,$$
$$d^2 = 2sr - s^2,$$
$$d = \sqrt{2sr - s^2}.$$

Hence, for a desired cusp length s, we determine the required offset $2d$.

4.1 Trimming the mesh

Creating one offset curve for a closed curve on a surface is not always unique. In general two curves exist, one on the inside and one on the outside. The same problem exists for the mesh. To circumvent the ambiguity, we mark the mesh in the inside of the starting curve as valid and the outside as invalid.

For this step we need an extension of the c-space data type, that allows us to split each edge into two parts: a valid and an invalid part. Thus, for the subsequent step of the algorithm only the valid part is considered. Assuming that the constant cusp algorithm is supposed to cover the entire mesh, initially all edges are valid, the invalid part is zero, and the starting curve is the boundary curve of the mesh. Figure 8 depicts the configuration at some iteration step. The mesh on the outside of the curve being removed. The subsequent step of the algorithm only considers the part of the mesh which lies inside the curve. This trimming step not only makes the computation of the intersection points with the surrounding tube unique, but also halves the computation costs.

4.2 Intersection of the c-space with the tube

Since the curve is piecewise linear, the tube can be approximated by a sequence of cylinders. The cylinders are bounded by two planes, whereas two adjacent cylinders share one bounding plane (see Fig. 9). As the cylinders have constant radii, they stitch together continuously. Consequently, the computation of the intersection of the c-space mesh with the tube surrounding the given curve reduces to calculating the intersection points between cylinders and lines. The cylinders form the tube and the lines are the edges of the quadrilaterals in the mesh structure.

For calculating the intersection points, we have to derive the bounding planes $\mathscr{E}_i$ and $\mathscr{E}_{i+1}$ for each cylinder $\mathscr{C}_i$ (see Fig. 9). Knowing that the points p_i of the line

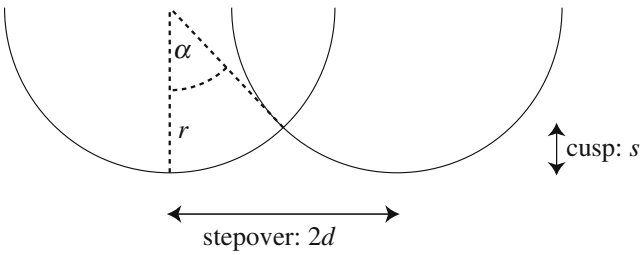


Fig. 7 This figure illustrates the relation between step over and the cusp height. For a desired cusp height s an stepover of $2d = 2\sqrt{2sr - s^2}$ is required

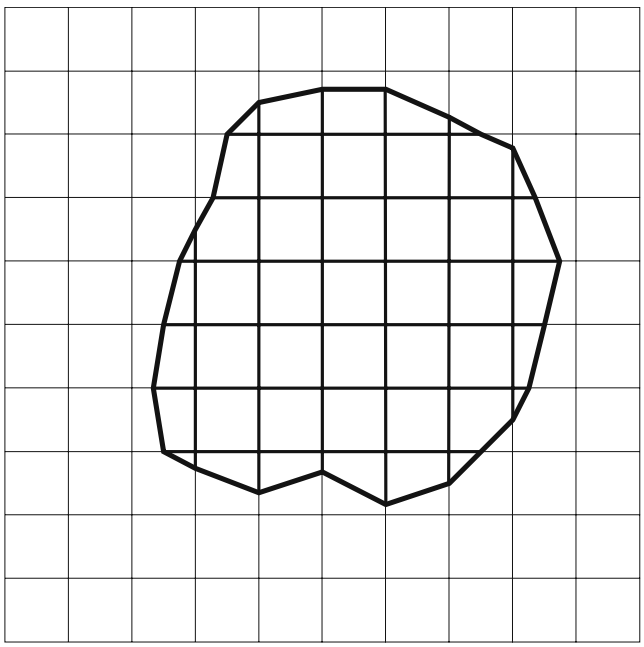


Fig. 8 The figure illustrates the trimmed mesh. Here, we want the curve to grow into the inside and hence we trimmed the mesh on the outside. So only the inner part will be considered

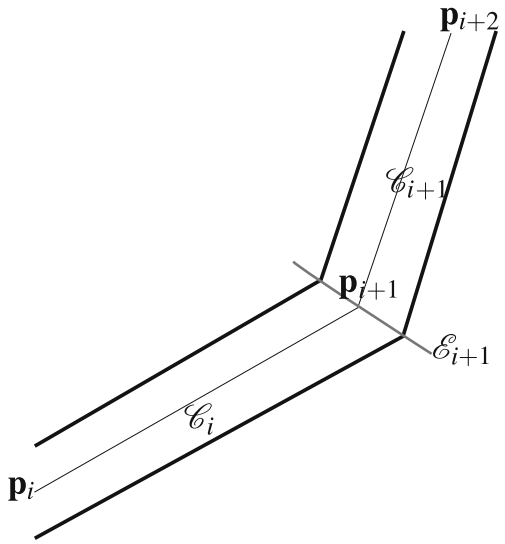


Fig. 9 Cylinders around several line segments form the tube for intersecting the c-space. The cylinders are bounded by planes. The adjacent cylinders $\mathscr{C}_i$ and $\mathscr{C}_{i+1}$ share the bounding plane $\mathscr{E}_{i+1}$

segments describing the curve lie in the bounding plane $\mathscr{E}_i$, what is left is to compute the normal vector of plane $\mathscr{E}_i$. The normal vector $\mathbf{n}_i$ of the plane $\mathscr{E}_i$ is derived by

$$\boldsymbol{d}_i := \boldsymbol{p}_i - \boldsymbol{p}_{i-1},$$

$$\boldsymbol{v}_i := \boldsymbol{d}_{i+1} - \boldsymbol{d}_i,$$

$$\boldsymbol{u}_i := \boldsymbol{v}_i \times \boldsymbol{d}_{i+1},$$

$$\boldsymbol{n}_i := \boldsymbol{u}_i \times \boldsymbol{v}_i.$$

This geometric consideration and the respective notations are depicted in Fig. 10. $\mathbf{d}_i$ describes the vector along the line segment and $\mathbf{v}_i$ the bisecting line of the

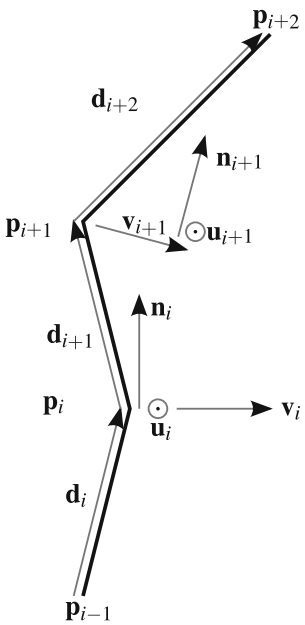


Fig. 10 For calculating the bounding plane for the cylinders, we need to derive its normal vectors $\mathbf{n}_i$

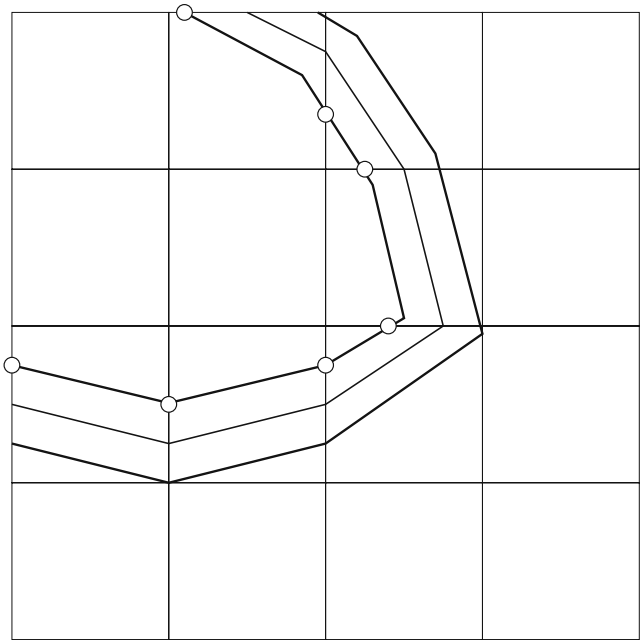


Fig. 11 After setting up the tube the white points result from the intersection of the tube with the mesh. Since in the preceding step we chopped the mesh, we only obtain the points on the desired side of the curve and not on both sides

angle between adjacent line segments. If $\mathbf{d}_i$ and $\mathbf{d}_{i+1}$ are collinear, we set $\boldsymbol{n}_i := \boldsymbol{d}_{i+1}$.

Having computed all the bounding planes, we have a complete representation of the tube and, hence, the intersection points with the mesh can be calculated. See [29] for details of calculating intersecting points between line and cylinder. Figure 11 shows an example of calculated intersecting points. Since we chopped the mesh in the preceding step, we ensure that the calculated points are on the desired side of the curve.

A common issue in calculating offset curves are self-intersections. Figure 12 illustrates the difference

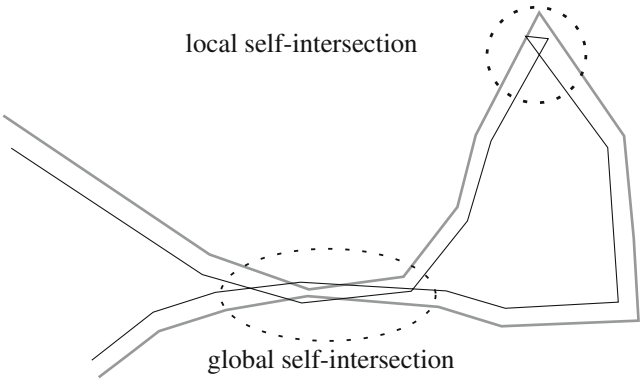


Fig. 12 Local and global self-intersections. Both kinds of intersection can occur while calculating offset curves and have to be solved

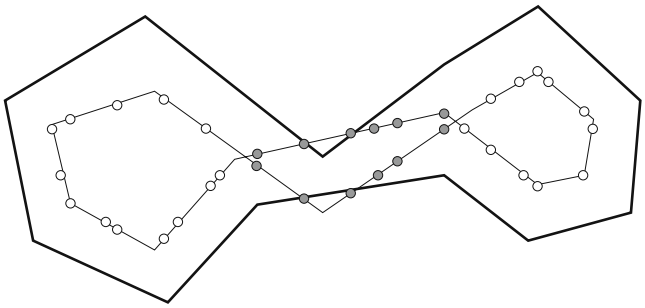


Fig. 13 To avoid the need of solving global and local self-intersections we remove all points, which lie inside the tube. The invalid points are marked *grey*. So the remaining points result in two independent curves

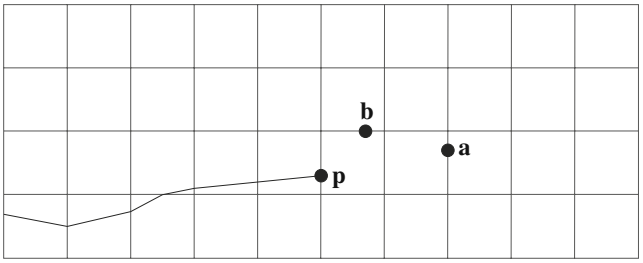


Fig. 15 In a valid structure, each line segment of the to be generated linear curve representation lies inside one quadrilateral, i.e., start and end point of the segment lies on an edge of the quadrilateral. Point **a** does not lie in the same quad as point **p**, i.e., continuing with point **a** would destroy the structure. Only continuing with point **b** keeps the structure of the resulting curve valid

between global and local self-intersections. Global self-intersections are those where two parts of the curve intersect, whereas local self-intersections are those where the curve exhibits a small loop. Finding and solving the global and local self-intersections is, in general, not trivial [10, 22, 33]. However, our approach allows for a simple check. In order to make the testing for global and local self-intersections obsolete, we check for each point, if the point lies inside the chopping tube. Each point is checked against all part of the tubes surrounding the piecewise linear segments. Consider segment $\mathbf{d}_{i+1}$ presented in the example in Fig. 10. The part of the tube surrounding this segment is described by a cylinder with radius $r \in \mathbb{R}$ and limited by the plane through $\mathbf{p}_i$ with normal $\mathbf{n}_i$ and plane through $\mathbf{p}_{i+1}$ with normal $\mathbf{n}_{i+1}$. All the points in the tube are invalid and are removed. Connecting the remaining points results in self-intersection free curve(s). Figure 13 shows an example. All the grey marked points lie inside the tube and are removed. Considering only the white points results in two separate curves without local or global self-intersection.

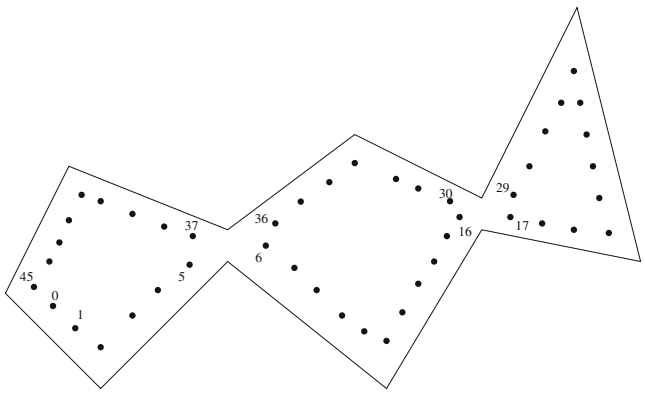


Fig. 14 Processing the points in the order of the indexing, the first index on the stack becomes 6 and we continue with point 37 until we reach the last point 45. Then, we continue with point 6 and go on until 16. We put 17 on the stack and continue with 30 until we reach 36. Finally, we connect the last part from 17 to 29

4.3 Connecting the points

The set up of the tube and of the mesh ensures that the calculated intersection points form a valid curve. Each resulting curve segment remains within one quadrilateral, i.e., it does not span over multiple quadrilaterals. The fact that points can become invalid and are consequently removed, prohibits the simple method of just taking point by point in the given order to connect them forming a piecewise linear curve (see Fig. 13).

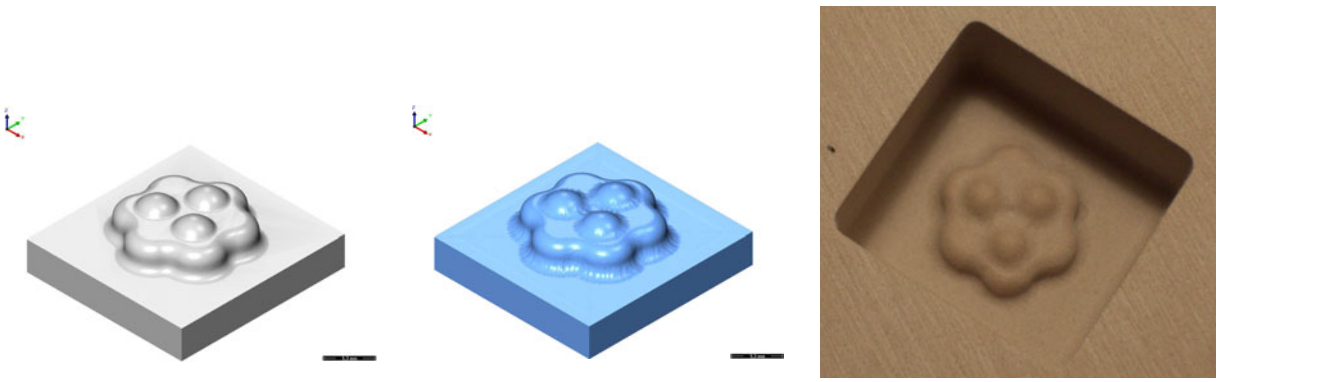
Instead for each point (in the given order) it is tested whether taking this point as the subsequent one keeps the structure valid, i.e., whether the point shares a common quadrilateral with the preceding one. If taking the point destroys the structure, we search for the next valid point, skip all points in between, put the index of the structure destroying point into a stack, and continue with the valid point. During one run, multiple points can be put on the stack. After finishing the first curve, we start a new one with the top most index from the stack and proceed until the stack is empty. Figure 14 illustrates the processing of the points. Figure 15 illustrates two opportunities to continue connecting the points, but only point **b** keeps the structure valid, since this point lies on the same quadrilateral as point **p**.

5 Results

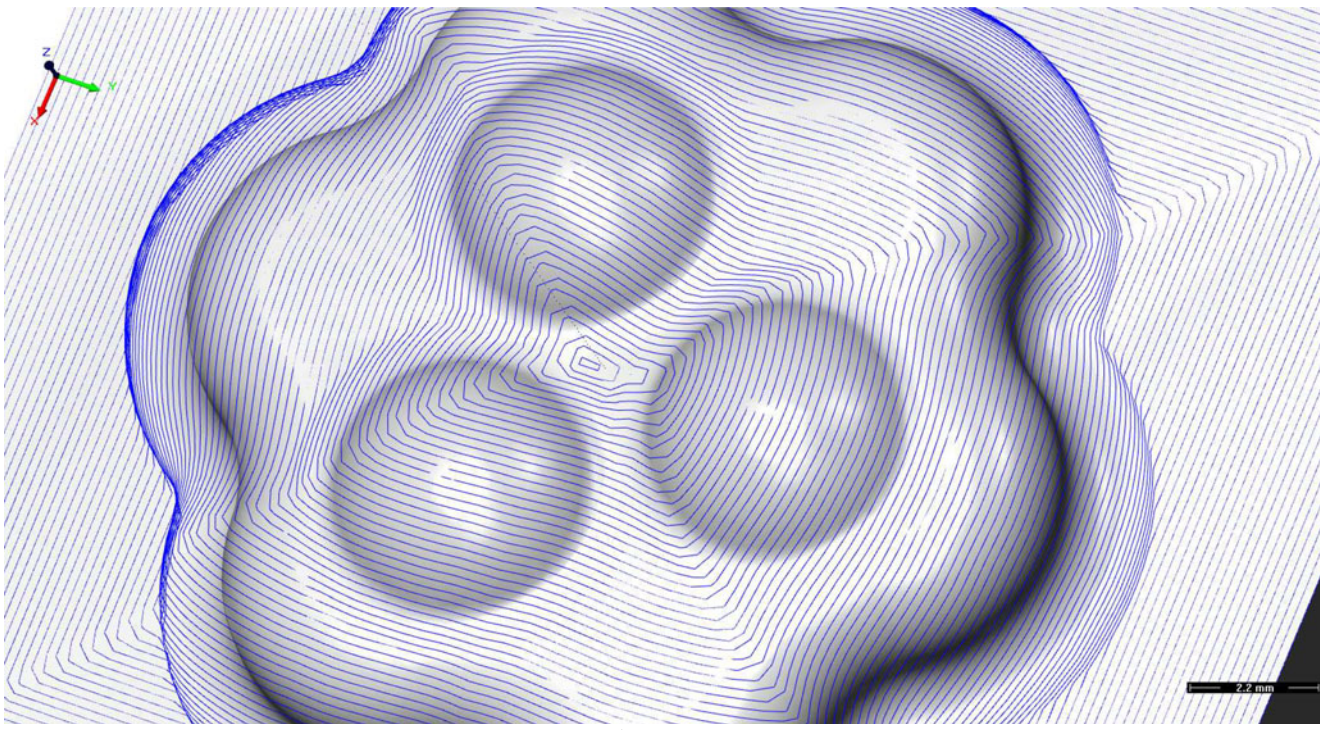
We developed the chop by tube algorithm for implementing the constant cusp strategy and tested the algorithm on real world and synthetic examples. The

examples provide a wide range of features, like round corners, sharp corners and vertical surface patches. For the implementation we used a framework provided by ModuleWorks GmbH. The framework offers function to convert the tool path into machine readable code with respect to DIN 66025-2 and offers the opportunity to simulate the process. For example see the book from Choi and Jerard [2] for more details about simulating. All remaining information, which is required to implement the algorithm, is mentioned in previous sections. All the calculating steps (creating the c-space and calculating the tool path) were executed on a PC with an Intel(R) Core(TM) Duo CPU with 2.2 GHz, 3 GB of RAM with Windows Vista as Operating System. In our implementation we use as input workpieces represented as triangle mesh and generate the c-space according to the method by Kavraki [19] as mentioned in Section 3.

In Fig. 16, we show the results of the processing steps, starting from the CAD model of the workpiece in

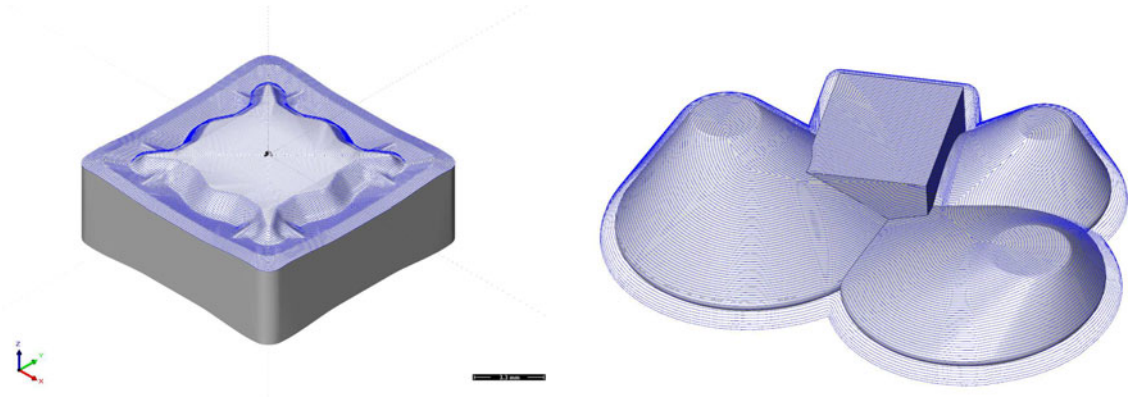


(a) Workpiece (c) Simulated result (d) Result



(b) Toolpath

Fig. 16 Processing of a real world workpiece. Figure 16a shows the designed workpiece. Figure 16b shows the calculated tool path. The tool path indicates the position of the tool tip. Figure 16c shows the results of the simulated milling process and Fig. 16d shows the resulting workpiece milled in wood



(a) Second workpiece (b) Synthetic example

Fig. 17 Overview of the tool path on two examples. The first one in Fig. 17a is a real-world example and the second one in Fig. 17b is a synthetic example

Fig. 16a, the resulting tool path in Fig. 16b, the results of simulating the milling process in Fig. 16c (generated by the framework), and finally the resulting workpiece in Fig. 16d. Note that the depicted tool path in all examples represents the positions of the tool tip and not of the tool contact point.

The rectangular socket underneath the workpiece has a size of 25 × 25 mm, which can be considered as a kind of bounding box. The object is processed with a 3-axis machine from top and milled into a block. One can see the block only in Fig. 16d which is not displayed in the remaining figures. We used a ball end tool with a diameter of 2.5 mm. For this setup we created the mesh for the c-space with a grid size of 0.01 mm. Creating the whole tool path took 2.3 s.

Figures 17 and 18 show two examples in an overview and a close up view. The first example is a real world example. Here the socket has a size of 120 × 120 mm and the tool path is generated for a ball end tool with a diameter of 1 mm. The mesh for the c-space has a grid size of 0.005 mm.The second example is a synthetic example. Here the socket (bounding box) has a size of 123 × 163 mm and the tool path is generated for a ball end tool with a diameter of 10 mm. The mesh for the c-space has a grid size of 0.01 mm and the calculation took 24.9 s.

We used several comercial programms to process Fig. 17 to compare oure algorithm and implementation with existing ones. The results are presented in Table 1. It is not easy to compare different programms, since they have specific features, which cannot be compared. Powermill displays the tool path while calculating, which slows down the calculating process. Cimatron cannot work with stl files, while all other programms took a triangle mesh as input.

6 Discussion of c-space properties and their impact on the algorithm

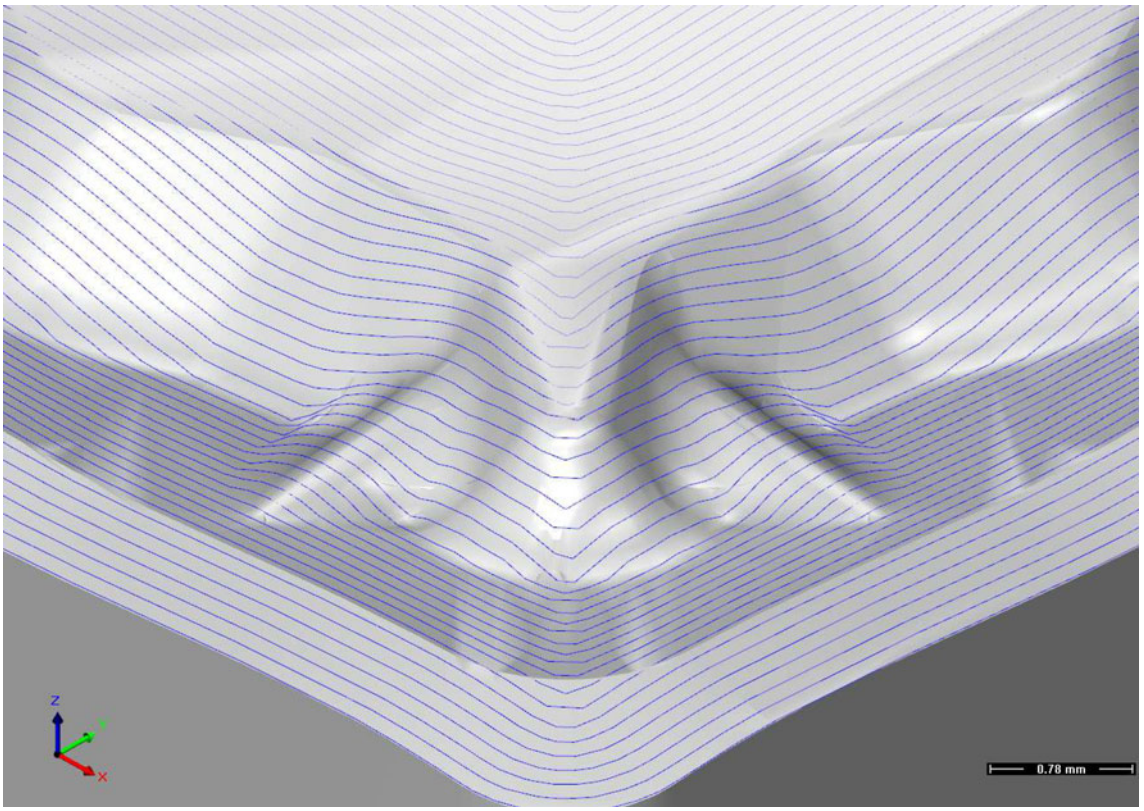
Our algorithm creates offset curves in c-space, which induces curves on the surface of the workpiece. In the following, we investigate whether the curves on the surface also have a constant distance like the curves in the c-space. For simplicity, we consider a ball end tool. Similar observations can be made for other tools. Furthermore, we assume a slice of the workpiece—and the corresponding c-space—perpendicular to a computed offset curve in point $c(\mathbf{p})$ and intersecting the subsequent offset curve in point $c(\mathbf{q})$, see Fig. 19. Within the slice, we observe two curves representing the workpiece and the c-space. The points $\mathbf{p}$ and $\mathbf{q}$ lie on adjacent curves on the workpiece and the points $c(\mathbf{p})$ and $c(\mathbf{q})$ lie on the corresponding adjacent offset curves in the c-space.

Since we are considering a slice, the workpiece can be represented by a function $f : \mathbb{R} \to \mathbb{R}$. Let $f(t), t \in \mathbb{R}$, describe the workpiece within the slice, $\mathbf{n}(t)$ its 2D normal, and $r \in \mathbb{R}$ the radius of the tool. The c-space $c(t)$ can be described by

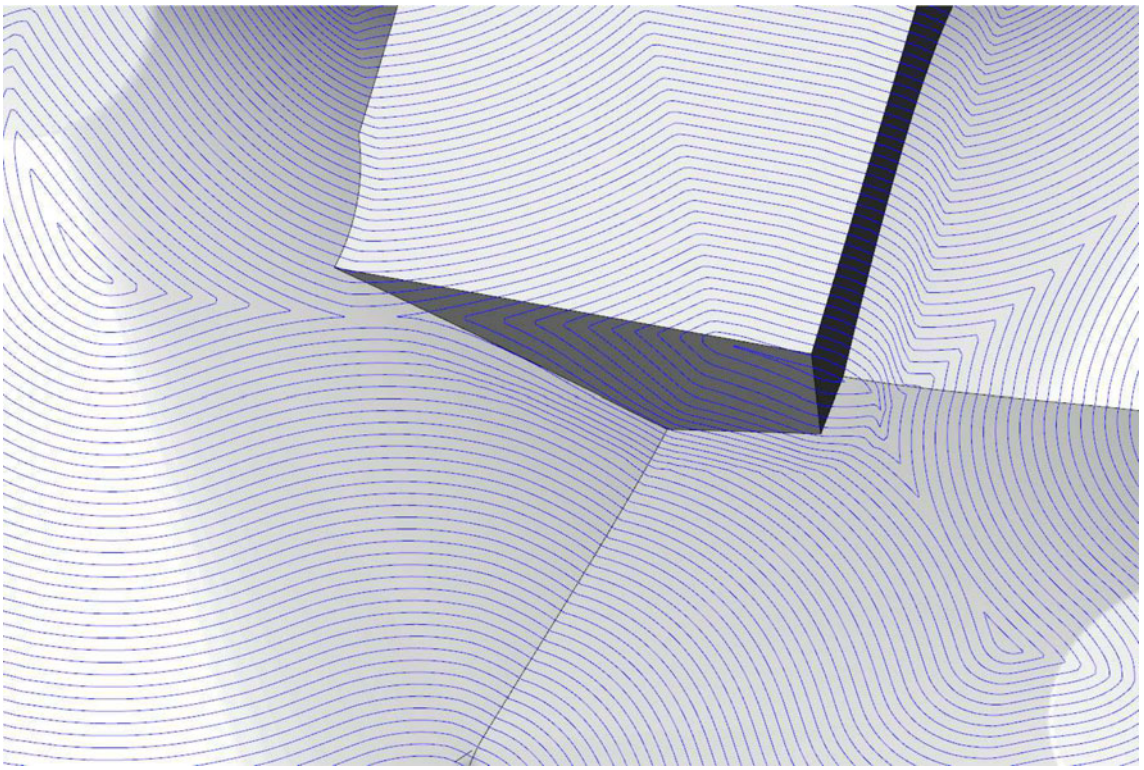
$$c(t) := f(t) + r\mathbf{n}(t) - r\mathbf{e}_z,$$

where $\mathbf{e}_z$ is the second unit vector. In Fig. 20, the workpiece is represented by the solid line and the corresponding c-space by the dashed line. This relationship illustrates: If the workpiece has no curvature, then offset curves with a constant distance on the workpiece corresponds to offset curves in c-space with the same constant distance.

Next, we want to consider the case depicted in Fig. 21 consisting of two adjacent linear pieces with an angle α between them. Let $c(\mathbf{p}_1)$ be the corresponding point in



(a) Second workpiece close up



(b) Synthetic example close up

Fig. 18 Close up of the examples in Fig. 17

Table 1 Comparing our implementation with existing comercial software for processing the workpiece in Fig. 17

Programm	Time (s)	Memory usage (MB)	Remarks
Our implementation	24	84	
Powermill	115	45	Tool path is displayed during calculation.
Cimatron	339	367	Cimatron cannot compute on STL files.
Mastercam	81	176	

The table gives an overview of the consumed time, the used memory before the algorithm is started, and the maximum consumed memory

the c-space of the point $\mathbf{p}_1$ on the workpiece as shown in Fig. 21 and let $d(\cdot, \cdot)$ be the distance between two points on the workpiece or in the c-space. As described above $d(\mathbf{p}_1, \mathbf{p}_2) = d(c(\mathbf{p}_1), c(\mathbf{p}_2))$, since the points lie on the same linear piece. The critical area on the workpiece is between points $\mathbf{p}_3$ and $\mathbf{p}_4$. This area corresponds to one single point in the c-space. So the error ε is exactly the distance $d(\mathbf{p}_3, \mathbf{p}_4)$. In general, we obtain the same error $\varepsilon = |d(\mathbf{p}_1, \mathbf{p}_5) - d(c(\mathbf{p}_1), c(\mathbf{p}_5))|$ for any two points each lying on one of the linear pieces. If the angle between the two linear pieces is α and the radius of the tool is r, the error can be described by

$$\varepsilon = 2r \tan\left(\frac{\pi - \alpha}{2}\right).$$

A common model does not have sharp angles between its planar elements, except around feature lines. In general, if a continuous model is converted to a piecewise linear model, the angles are bounded.

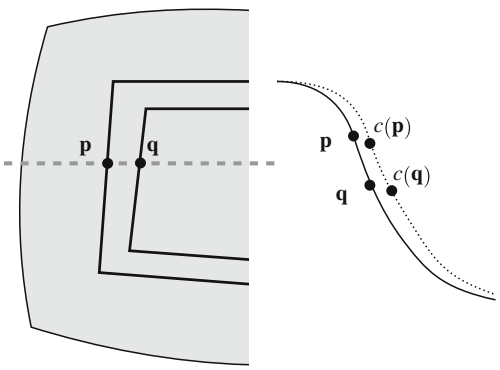


Fig. 19 We consider a slice of the workpiece along the *dashed line* (*left*) with two points from different offset curves. The c-space within the slice is shown with a *dotted line* (*left*), whereas the workpiece within the slice is shown with a *solid line*. The difference of the distance of the points in the c-space $d(c(\mathbf{p}), c(\mathbf{q}))$ and on the workpiece $d(\mathbf{p}, \mathbf{q})$ represents the error

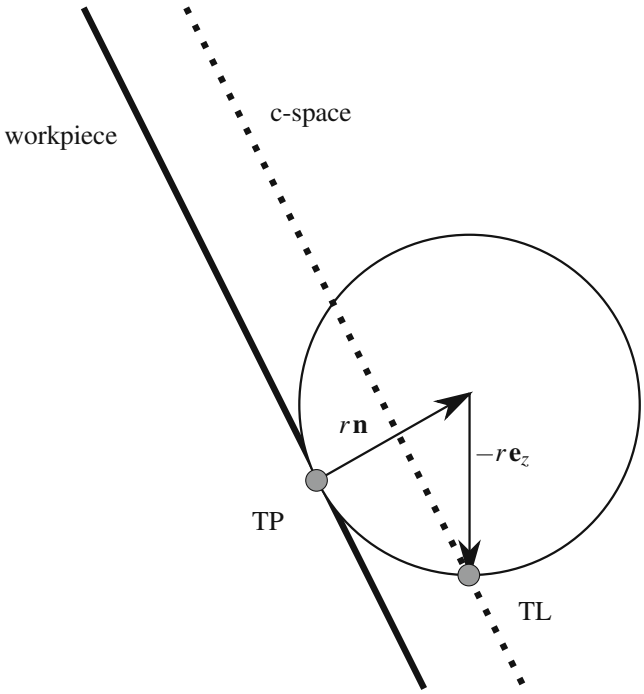


Fig. 20 This figure shows the relation of the workpiece or the tool center point (*TP*) to the c-space or the tool location point (*TL*). Starting from the workpiece in direction of the normal of the amount r—radius of the tool—and then in direction $-\mathbf{e}_z$ of the amount r provides the corresponding location in the c-space

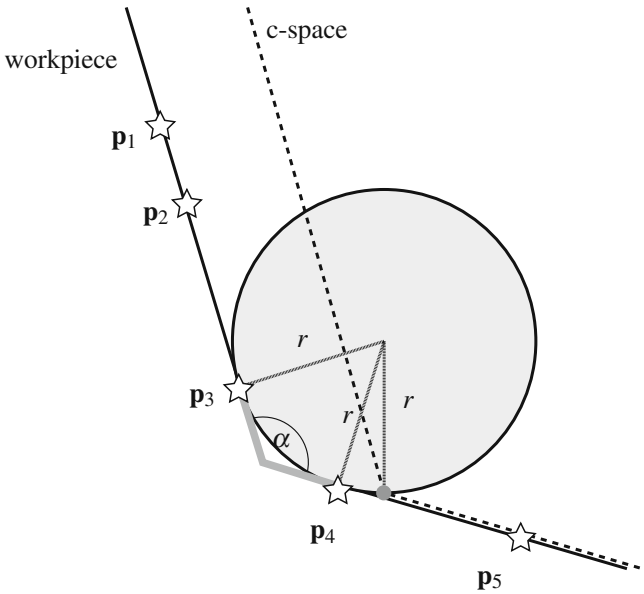


Fig. 21 All points on the workpiece between $\mathbf{p}_3$ and $\mathbf{p}_4$ correspond to one single point in the c-space. With $d(\cdot, \cdot)$ as distance function between two points on the workpiece or in the c-space the error in this example can be described by $\varepsilon = d(\mathbf{p}_3, \mathbf{p}_4)$ and hence $|d(\mathbf{p}_1, \mathbf{p}_5) - d(c(\mathbf{p}_1), c(\mathbf{p}_5))| = \varepsilon$, where $c(\mathbf{p}_1)$ and $c(\mathbf{p}_5)$ are the corresponding points in the c-space to the points $\mathbf{p}_1$ and $\mathbf{p}_5$ on the workpiece

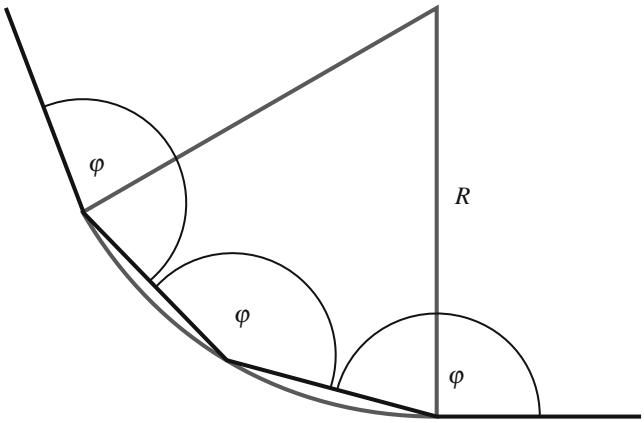


Fig. 22 Assuming no feature lines, the angle between two adjacent linear pieces should be close to π. So a blend with radius R between two linear pieces is represented by several linear segments

Figure 22 shows an example. The circle is represented by line segments and for its enclosed angles φ holds that

$$\cos(\pi - \varphi) > \gamma, \tag{1}$$

where γ is a suitable constant (e.g., $\gamma := 0.95$).

In general, for processing a workpiece the radius of the tool r must be smaller than the smallest curvature circle radius R of the workpiece. Moreover, the distance between two adjacent curves of the offset curves $2d$ must be smaller than the diameter of the tool. These considerations lead to the relation

$$d < r < R.$$

The length x of a linear segment that contributes to the representation of a circle is given by

$$\begin{aligned} x &= \sqrt{R^2(1-\cos(\pi-\varphi))^2 + R^2\sin(\pi-\varphi)^2} \\ &= \sqrt{2}R\sqrt{1-\cos(\pi-\varphi)}. \end{aligned}$$

The maximum number of angles m between two points on the workpiece is given by

$$\begin{aligned} m &= \frac{d}{x} \\ &= \frac{d}{\sqrt{2}R\sqrt{1-\cos(\pi-\varphi)}}. \end{aligned}$$

This leads us to the maximum error ω between two corresponding distances on the workpiece and the c-space. The maximum error is the product of the number of angles and the error given by two adjacent segments with the angle φ. So the maximum error can be described by

$$\begin{aligned} \omega &= m \cdot \varepsilon \\ &= \frac{d}{\sqrt{2}R\sqrt{1-\cos(\pi-\varphi)}} \cdot 2r\tan\left(\frac{\pi-\varphi}{2}\right) \\ &\overset{(1)}{=} \frac{\sqrt{2}dr}{R\sqrt{1+\gamma}}, \end{aligned}$$

when the bounding of φ in Eq. 1 holds. When γ is close to one, we obtain

$$\omega \approx \frac{dr}{R}.$$

The derived error has been derived for one exemplary slice or position. Hence, ω is not necessarily constant over the whole curve. In general, the offset curves in the c-space do not lead to offset curves on the workpiece. However, depending on the surface, they lead to a tolerance band for the offset curves with a bounded and quite small diameter.

The intended application for this c-space set up are 3-axis applications. However, one can also handle 5-axis applications by moving the tool tip to the tool contact point and using the difference of tool center point and tool contact point, i.e., the tool is oriented such that it is perpendicular to the surface. However, such a 5-axis approach does not provide the opportunity to handle undercuts.

7 Conclusion and future work

In this paper we presented a novel approach to the constant cusp strategy for c-space set-ups. With the c-space approach we avoid problems with oscillations as they occur in NURBS surfaces and with long, stretched triangles as they occur in triangular meshes. This change to c-space makes our method robust and allows us to apply it to a large field of NURBS or triangular surfaces. Furthermore, the memory-saving mesh representation of the c-space enables us to use large meshes on standard computers and adapting the grid size of the mesh enables us to use any desired precision. Hence, this approach allows the user to have a handle for adjusting the trade-off between precision and computation time. Since the structure of the c-space works with piecewise linear curves, we achieved low computation times for real-world settings.

The algorithm is based on determining the valid part of the c-space, putting a tube with desired radius around the current piecewise linear curve, and computing intersections of the tube with the valid c-space part.

Connecting the points leads to the desired tool paths, while self-intersections of the resulting offset curves can easily be detected and removed.

In the real world, many workpieces have "obstacles" (e.g., holes) that need to be considered in the constant cusp algorithm. These are areas, that cannot be handled in the same processing step as the surrounding area. For future work, we want to extend our current algorithm, such that it can deal with such obstacles. Precisely, the obstacles are given as bounding curves by the user. The input will be an outer curve, which can include further curves, that mark areas to be left out.

Acknowledgements We would like to thank ModuleWorks GmbH for providing us with their framework for implementing our algorithm and simulating the milling process.

Furthermore, we would like to thank David Plater for the constructive discussions.

References

1. Bommes D, Kobbelt L (2007) Accurate computation of geodesic distance fields for polygonal curves on triangle meshes. In: VMV, pp 151–160
2. Choi BK, Jerard RB (1998) Sculptured surface machining: theory and applications, 1st edn. Springer-Verlag Gmbh
3. Choi BK, Kim DH, Jerard RB (1997) C-space approach to tool-path generation for die and mould machining. Computer Aided Des 29:657–669
4. Choi BK, Ko K (2003) C-space based capp algorithm for freeform die-cavity machining. Computer Aided Des 35:179–189
5. Choi BK, Park SC (1999) A pair-wise offset algorithm for 2d point-sequence curve. Computer-Aided Des 31:735–745
6. CIMdata I (2008) http://www.cimdata.com
7. Coquillart S (1987) Computing offset of b-spline curves. Computer Aided Des 19(6):305–309
8. Deutsches Institut für Normierung e.V. (1988) DIN 66025-2. http://www.din.de
9. Dragomatz D, Mann S (1997) A classified bibliography of literature on NC milling path generation. Computer Aided Des 29(3):239–247
10. Elber G, Cohen E (1991) Error bounded variable distance offset operator for freeform curves and surfaces. Int J Comput Geom Appl 1:67–78
11. Farouki R, Neff C (1990) Algebraic properties of plane offset curves. Computer Aided Geom Des 7:101–127
12. Farouki R, Neff C (1990) Analytic properties of plane offset curves. Computer Aided Geom Des 7:83–99
13. Farouki RT, Neff CA (1990) Algebraic properties of plane offset curves. Computer Aided Geom Des 7:101–127
14. Hoffmann CM (1989) Geometric and solid modeling: an introduction (The Morgan Kaufmann series in computer graphics and geometric modeling). Morgan Kaufmann Publishers, US
15. Hoschek J (1985) Offset curves in the plane. Computer Aided Des 17(2):77–82
16. Hoschek J (1988) Spline approximation of offset curves. Computer Aided Geom Des 5:33–40
17. Hoschek J, Lasser D (1993) Fundamentals of computer aided geometric design. A.K. Peters, Ltd
18. Hoschek J, Wissel N (1988) Optimal approximate conversion of spline curves and spline approximation of offset curves. Computer Aided Des 20(8):475–483
19. Kavraki L (1995) Computation of configuration-space obstacles using the fast fourier transfom. IEEE Trans Robot Autom 11:408–413. doi:10.1109/70.388783
20. Kim T (2007) Constant cusp height tool paths as geodesic parallels on an abstract riemannian manifold. Computer Aided Des 39:477–489
21. Lee E (2003) Contour offset approach to spiral tool path generation with constant scallop height. Computer Aided Des 35:511–518
22. Lee I, Kim M, Elber G (1996) Planar curve offset based on circle approximation. Computer-Aided Des 28(8):617–630
23. Lee SG, Yang SH (2002) CNC tool-path planning for high-speed high-resolution machining using a new tool-path calculation algorithm. Int J Adv Manuf Technol 20:326–333
24. Lin RS, Koren Y (1996) Efficient tool-path planning for machining free-form surfaces. J Eng Ind 118:20–28
25. Marshall S, Griffiths J (1994) A survey of cutter path construction techniques for milling process. Int J Prod Res 32(12):2861–2877
26. Mitchell JSB, Mount DM, Papadimitriou CH (1987) The discrete geodesic problem. SIAM J Comput 16(4):647–668. doi:10.1137/0216045
27. Murray RM, Li Z, Sastry SS (1994) A mathematical introduction to robotic manipulation. CRC Press
28. Ortner S (2005) The keys to successful high-speed machining. http://www.moldmakingtechnology.com/articles/120501.html
29. Paeth AW (ed) (1995) Graphics gems V. Morgan Kaufmann
30. Park SC, Choi BK (2001) Uncut free pocketing tool-paths generation using pair.wise offset algorithm. Computer-Aided Des 33:739–746
31. Park SC, Chung YC, Choi BK (2003) Contour-parallel offset machining without tool-retractions. Computer-Aided Des 35:841–849
32. Pham B (1988) Offset approximation of uniform b-splines. Computer Aided Des 20(8):471–474
33. Seong JK, Elber G, Kim MS (2006) Trimming local and global self-intersections in offset curves/surfaces using distance maps. Computer Aided Des 38:183–193
34. Siemens (2006) SINUMERIK 840D sl/840Di sl/ 840D/840Di/ 810D Grundfunktionen: Bahnsteuerbetrieb, Genauhalt, LookAhead (B1)
35. Surazhsky V, Surazhsky T, Kirsanov D, Gortler SJ, Hoppe H (2005) Fast exact and approximate geodesics on meshes. ACM Trans Graph 24(3):553–560. doi:10.1145/1073204.1073228
36. Suresh K, Yang DCH (1994) Constant scallop-height machining of free-form surfaces. J Eng Ind 116:253–259
37. Weck M (1988) Werkzeugmaschinen Band 1. Maschinenarten, Bauformen und Anwendungsgebiete. Springer-Verlag, Berlin Heidelberg
38. Weck M (1997) Werkzeugmaschinen, Fertigungssysteme 2. Konstruktion und Berechnung. Springer-Verlag, Berlin Heidelberg
39. Weck M (2006) Werkzeugmaschinen 4: automatisierung von maschinen und anlagen. Springer-Verlag, Berlin Heidelberg
40. Weck M, Brecher C (2006) Werkzeugmaschinen 3: mechatronische systeme: vorschubantriebe, prozessdiagnose. Springer-Verlag, Berlin Heidelberg
41. Yuwen S, Dongming G, zhenyuan J, Haixia W (2006) Isoparametric tool path generation from triangular meshes for free-form surface machining. Int J Adv Manuf Technol 28, 721–726 (2006)