Efficient Volume-Generation During the Simulation of NC–Milling

Georg Glaeser, Eduard Gröller*

Abstract – This paper presents an efficient and robust algorithm for the geometric determination of swept volumes during the simulation of NC–milling (three-axis machining and five-axis machining). The boundary Ψ of the volume swept by a cutter Φ is represented polygonally by using instantaneous helical motions to exactly determine the line of contact between Φ and Ψ.

Applying concepts of differential geometry allows a better and more efficient approximation of tool paths. Tool paths are explicitly calculated when a design surface Γ is to be milled along prescribed curves.

We also describe how to quickly determine a polygonized representation of the truncated material during the milling process by means of „Γ–buffering“. This polygon-oriented algorithm is perfectly suitable for Boolean subtractions and error assessment.

Index Terms – computer aided manufacturing, NC–milling, NC–verification, sweeps, solid modeling, Γ–Buffer.

1 INTRODUCTION

Numerically controlled (NC) milling technology is a production process where a rotating cutter is sequentially moved along prescribed tool paths in order to manufacture a free-form surface from raw stock.

Different approaches of the simulation of the process were introduced in the past years, each of which has advantages and drawbacks. Two major approaches can be developed: the exact, analytical approach and the approximation approach.

The main problem with the accurate approach (mostly done in CSG) is its computational expense. The cost of simulation using the CSG approach is reported to be $O(N^4)$ [JeHu89], where N is the number of tool movements. A complex NC program might consist of ten thousand movements, thus making the computation intractable.

In order to increase efficiency, a number of approximate simulation methods have been devised. The computational cost of this methods simplifies to O(N) [JeHu89].

1.1 Accurate techniques

In 1991, Kawashima et al [KaIt91] used a special geometric modeling method called „Graftree“ to speed up their solid modeling approach. A Graftree is a combination of an Octtree and CSG. At the leaf nodes of the Octtree CSG elements are „grafted“ onto the tree, if the corresponding spatial partition contains a boundary of the object. Using CSG modeling, their approach allowed accurate and precise representation of the workpiece and the tool, while the Octtree helped to decrease the number of ray-intersection calculations in rendering.

Implementations of special Boolean set operations were developed, which worked on Graftrees. Using the Boolean difference operator, NC milling was simulated.

Another approach to find the exact representation of the envelope of a swept volume was chosen by Sourin et al. [Sour96]. They describe the cutting tool analytically with the use of procedurally implemented time-dependent defining functions. The swept volume of the tool consists of the instances of the cutter in initial and final position together with the envelope of the moving cutter. Boolean set operations are defined by Rvachevs R–functions [Rvac74]. The milling process can be visualized by ray tracing the analytically defined scene, which is, however, a very time-consuming process.

1.2 Approximate techniques

In 1986, van Hook [Hook86] managed to get update rates of 10 cutting operations per second by using special hardware equipment. A solid model was milled by a cutting tool which followed an NC path. A real time shaded display of the solid model was achieved by using image-space Boolean set operations. The workpiece and the tool were converted into a quasi „dexel structure“ which is an extended Z–Buffer. Dexel (depth elements) represent thereby rectangular solids aligned to the viewing direction. The Boolean subtraction operator could be easily applied to these quasi one-dimensional representations of solid models. It was not necessary to use the computational-expensive and time-consuming Boolean difference operation in 3-space.

G. Glaeser is with the University of Applied Arts, A-1010 Vienna, Austria E-mail: gg@geometrie.tuwien.ac.at.
E. Gröller is with the Vienna University of Technology, A-1040, Vienna. E-mail: groeller@cg.tuwien.ac.at.

This first approach by Van Hook had two major problems: the z-direction vector of the dexel structure was limited to the viewing vector, and the whole simulation process had to be redone when changing the viewpoint.
These problems were solved by an extension of the method by Huang and Oliver in 1994 [HuOl94]. The dexel structure used had its own coordinate system, independent of the viewing vector. The view dependency of Hooks approach is also overcome by converting the dexel structure into a set of contour lines, which can be viewed from any direction. Huang and Oliver also introduced the possibility of error assessment to Hooks method. Deviations of the final workpiece from the design part are displayed with color coding.
In applications where fast response time is essential, evaluation of the tool envelope surfaces in object space and then scan converting these surfaces to a dexel representation is undesirable. Scan converting is a time consuming process since it may require triangulating the surface and the scan converting all the created triangles.
If a dexel representation of the sweeping object already exists, it is possible to apply the sweeping operation in image space directly. No further scan conversion is required in this case. In 1994, Hui [Hui94] has developed a method to speed up the creation of the swept volume by doing the sweep calculations in image space.
In 1991 Saito and Takahashi also used an extension of the Z–Buffer to simulate NC milling [SaTa91]. A so-called G–Buffer is used to store various information per pixel: depth, normal vector, patch or object identifier, and other patch information. Such a G–Buffer can be produced by conventional rendering techniques. A tool path for NC machining can be generated by simply scanning the G–Buffer for height. This method can only be used for three-axis machining without pocketing. Error assessment is also impossible.
A completely different approach (called „lawn mower“ method) was chosen by Jerard et al in 1989 [JeHu89]. The design surface is approximated by a polygonal mesh, where vectors (e.g., normal vectors or z-axis aligned vectors) are stored during the simulation at the mesh points. These surface vectors are intersected with a polygonal approximation of the tool. The vectors are shortened to the amount of the over- or undercutting error when the tool moves over them. Finally the mesh points are displaced by the remaining parts of the surface vectors and color coded by the amount of the error. The resulting polygonal surface can be viewed from any direction using conventional Z–Buffer rendering techniques. This approach is very useful to assess the different kinds of errors which may evolve during simulation. These possibilities are investigated in an additional paper by Jerard et al. [JeDr89].
Yang and Lee [YaLe96] developed a method suitable for wire–EDM (wire cut electric discharge machining), where cutting is only done at the sides of the workpiece. They modified the dexel approach to a cylindrical R–map, where dexels may be accessed by the height at the cylinder axis and a rotation angle. Cutting errors are again color coded.
Some other related work is given in [Chap83], [JoWi95], [LiEs96], [VoWa81], [WaWa86], and [WaKa95].

	representation of							
	workpiece	tool	error assessment	tool path	tool shape	viewpoint dependent	swept volume generation	number of axes
[Hook86] „dexel approach“	dexel				arbitrary	•	stamping in image space	3, 5 (slower)
[HuOl94] „extended dexels“	dexel		•	correction	arbitrary		stamping in dexel space	5
[JeHu89] „lawn mowing“	mesh with normal vectors	polygonal	•		ball or cylinder		explicit as polygons	3/5
[Kalt91] „Graftree“	Octtree / CSG		•		arbitrary		explicit as CSG	5
[SaTa91] „G–Buffer“	extended Z–Buffer		•	generation / verification	arbitrary	•	scanning of G–Buffer	3
[YaLe96] „wire EDM“	R–Buffer		•		wire		explicit as polygons	4 (wire–EDM)
[Sour96] “moving solid”	analytical				arbitrary		analytically, 4D functions	
our technique „Γ–Buffer“	Γ–Buffer	polygonal	•	generation / verification	arbitrary		scanning of Γ–Buffer explicit as polygons	3/5

Table 1: Overview of some (previous) work on NC milling

1.3 Our approach via Differential Geometry

This paper presents an efficient and robust algorithm for the geometric determination of swept volumes during the simulation of NC-milling. This is essential for the optimization of cutter shapes and cutter paths. Both the three-axis machining and the five-axis machining are covered.
First, we introduce an efficient algorithm for the polygonal representation of the boundary Ψ of the volume swept by the boundary Φ of a cutter (which is union of a surface of revolution and possible additional boundary circular disks). It takes advantage of the fact that in each moment any spatial motion can be interpreted as an instantaneous helical motion (the axis and

the parameter of this motion are determined). This allows to very efficiently calculate the line of contact between Φ and Ψ (called the characteristic curve *c* of Φ). Now Ψ can be interpreted as a surface swept by a curve *c*, which is much easier to polygonize.
The tool path is explicitly calculated when a design surface Γ is to be milled along a prescribed curve.
Second, we describe how to quickly determine a polygonized representation of the truncated material, during the milling process and after the cutter has been moved along all the prescribed paths. For this, the set of all the sweeps Ψ_k has to be 'subtracted' from the raw material. The polyhedra Ψ_k are of potentially complex shape and strongly tend to have self-intersections that are difficult to eliminate. For fast applications, they are not suitable for solid modeling.
Instead, we introduce a so-called „Γ-Buffer" (related to a G-Buffer). This polygon-oriented algorithm works in object space and is perfectly suitable for Boolean subtractions and error assessment. It creates a polygonized representation of the truncated material which can be displayed very efficiently.
To sum up, our approach has three major advantages:

- The consequent use of differential geometric results speed up the determination of the sweeps Ψ_k.
- The Γ-Buffer works very efficiently and produces polygonized representations of the workpiece. Additionally, error assessment is done without further computational expense.
- There are no limitations to the shape of the cutting tool.

2 SWEPT VOLUMES

Sweeping is a technique that creates new solids that are not simple Boolean constructions and combinations of solid primitives. In [Sour96], a method is described how to treat both solids modeled with set-theoretic operations and solids modeled with sweeping on the general base of the representation by a real function of three variables. This provides solutions for problems like the sweeping of CSG–solids, self-intersections, and set operations on sweeps. The method, however, is extremely computation-intensive.
It uses a result of differential geometry:
Let Φ be the boundary surface of a solid undergoing a motion, given by the implicit equation $f(x, y, z, t) = 0$ (where *t* is a motion parameter).Then Φ and its envelope Ψ have a line of contact *c* in common, determined by the additional equation

$$df(x, y, z, t) / dt = 0.$$

This line of contact is determined by a numerical algorithm. We now try to introduce some geometric considerations that help to determine *c* much faster, especially when Φ is a surface of revolution.

2.1 The Instantaneous Helical Motion $\Im$

A major theorem of spatial kinematics is that in any moment an arbitrary space motion can be replaced by an infinitesimal helical motion [Wund67]. (A helical motion is a rotation around an axis plus a proportional translation along this axis. The proportion factor is called parameter. The path curves of space points are helical curves.) Each spatial motion can thus be interpreted as an integral motion of an infinite number of helical motions. In general, the manifold of all instantaneous helical axes is a ruled surface.
In order to determine the corresponding helical motion $\Im$ (i.e., the helical axis and the helical parameter), we need two „neighboring" point-triples, i.e., two „snap shots" of a rigid triangle at the time *t* and $t + dt$ [Wund67].

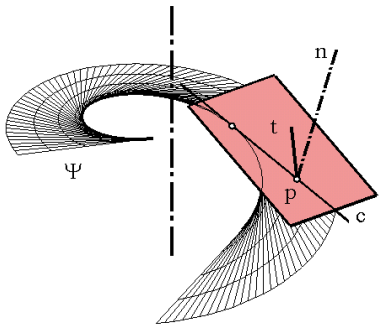


Fig.1

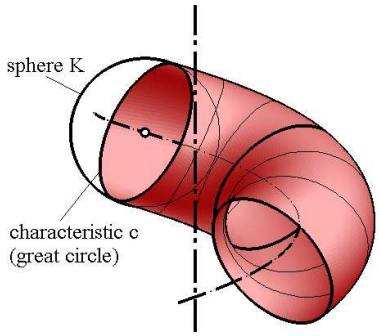


Fig.2

As a well-known application, consider a plane undergoing $\Im$ (Figure 1). It will in general envelope the tangent surface Ψ of a certain helix that is a path curve of $\Im$. The only tangent *c* of this helix in the plane is the so-called characteristic straight line *c* of the plane. For the points *p* of the straight line *c* we have the linear condition **t . n** = 0 (where **t** is the tangent vector of a point *p* undergoing $\Im$ and **n** is the constant normal vector of the plane).
Another well-known example for a characteristic curve *c* occurs when a sphere is undergoing a helical motion (Figure 2), sweeping a tubular helical surface. In this case, *c* is the great circle of the sphere the axis of which is the helical-tangent of its center.

2.2 The Line of Contact on a Polyhedron

Let Φ be a closed polyhedron (the polygonized boundary of a solid), undergoing a helical motion $\mathfrak{I}$. Let **F** be a polygonal face of Φ in the plane π. When the interior points of **F** are submitted to $\mathfrak{I}$, the dot product **t . n** of the corresponding helical tangent vector **t** and the normal vector **n** of π will vary linearly (which is easy to prove). We can distinguish three cases:

1. There are points inside the polygon with **t . n** = 0. In this case, the straight characteristic line of π intersects the polygon. Because of the linearity of the condition, the two points of c_π on the sides of the polygon can be found by means of linear interpolation.
2. **t . n** < 0 for all the points inside **F.** This is true for *all* interior points of the polygon, when it is true for all the vertices of the polygon.
3. **t . n** > 0 for all the points inside **F** (same condition as in case 2).

The characteristic polygon c_Φ of Φ now consists of one or more closed branches, the edges of which are either parts of straight characteristic lines c_π or edges that faces of type 2 and type 3 have in common [StEl96].

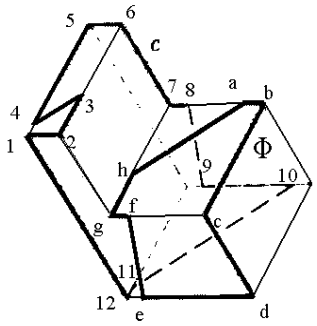


Fig.3

Figure 3 shows an example, where c_Φ consists of two branches *1, .. ,12* and *a, .. , h*. As one can see, characteristic polygons can be quite tricky, especially when the helical axis intersects the polyhedron.

2.3 The Line of Contact on a Surface of Revolution

During the milling process, the cutting tool is rotating around its axis, thereby generating a surface of revolution Φ. (The rotation of the cutting tool around its axis can be neglected for geometrical considerations). Thus, it is sufficient to consider this surface, which we will call „the cutter" henceforth. The motion of Φ is subsequently interpreted as an infinite sequence of instantaneous helical motions. Furthermore, we will always talk about the *relative motion* of Φ in regard to the workpiece (regardless of the fact that the workpiece itself might be undergoing a motion).

When the cutter is moved along, it is sufficient to know about the path curve of one point on the cutter-axis and the corresponding direction of the axis. The axis positions determine a set of instantaneous helical motions $\mathfrak{I}$, as will be explained in Section 3.

Now we determine the characteristic curve of the surface of revolution. To do so, we have two possibilities:

1. We approximate the surface Φ by a polyhedron and then we apply the above mentioned algorithm for the determination of the characteristic polygon on polyhedra. In some cases, this is a quite practicable way.
2. We use the fact that a surface of revolution can always be interpreted as a an envelope of a one-parametric manifold of spheres *K* the centers of which lie on the rotational axis of Φ (which may degenerate to planes perpendicular to the axis).

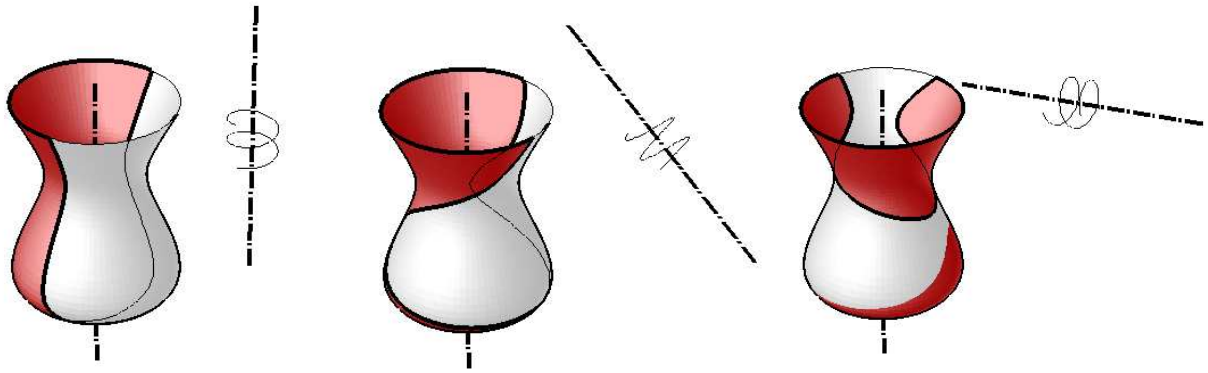


Fig.4

Each sphere *K* touches Φ along a parallel circle *k*. When *K* is undergoing the helical motion $\mathfrak{I}$, it sweeps a tubular surface Ψ_K. As we know, the line of contact c_K of *K* and Ψ_K is a *great circle* of *K* the axis of which is the helical tangent of the center of the sphere. The two possible intersection points s_1 and s_2 of *k* and the plane containing c_K are points on Φ that also belong to the swept volume Ψ of Φ, since the corresponding tangent planes of *K* and Φ are identical.

For a sufficient number of parallel circles on Φ, we can compute the points s_1 and s_2 very inexpensively. As long as each parallel circle carries two real points, the line of contact c consists of two separate branches. When the intersection points s_1 and s_2 are identical, the two branches of c „grow together". Since we deal with solids, parts of the bordering circles have to be included to c as well. Therefore, c consists of one or more closed branches with corner points at the bordering circles.
The shape of the characteristic curve c on a surface of revolution Φ varies considerably with the position of the instantaneous helical axis. When Φ is concave (Figure 4), c may consist out of several closed branches. Cutter shapes, however, are usually convex surfaces of revolution (Figure 5). In this case, c is always one closed curve.

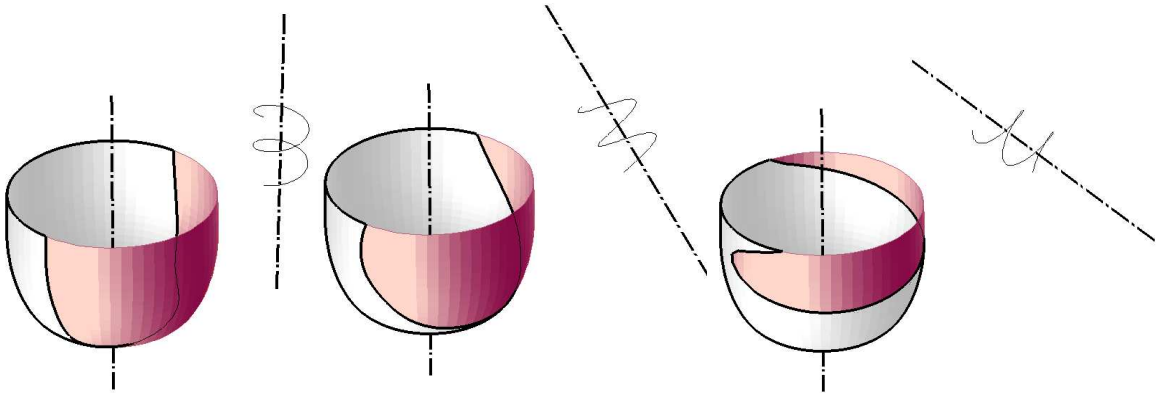


Fig.5

2.4 The Swept Volume of the Cutter

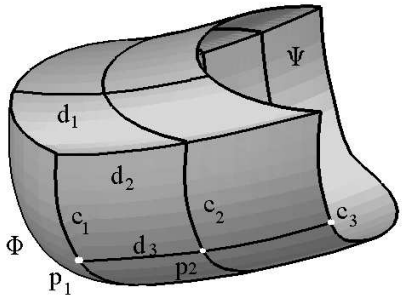


Fig.6a

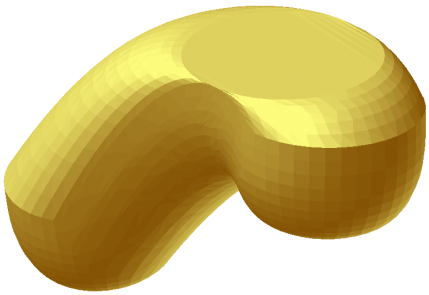


Fig.6b

When we calculate the line of contact c for a sufficient number of instantaneous helical motions, we get a one-parametric set of lines $\{c\}$ on the boundary Ψ of the volume swept by Φ. The question is now: how can we interpolate a second set $\{d\}$ of lines on Ψ so that Ψ comes to reality (i.e., to the volume swept by the cutter) as close as possible?
Surprisingly, corner points on c do *not* produce sharp edges on Ψ (though it looks like it in Figure 6a). This is because the plane spanned by the two tangents in a corner point is tangent plane of Φ. The only potential sharp edges of Ψ are produced by self-intersections and by parts of the bordering circles at the beginning and at the end of the sweeping (Figure 6b). Thus, a line d may also cross over the lines determined by all corner points.

To find a line d, we start from an arbitrary point p_1 on the first characteristic line c_1 (Figure 6a). The tangent t_1 of d lies in the corresponding helical tangent plane τ_1. As close as possible to t_1, we look for a new point $p_2 \in d$ on the second characteristic line c_2. The tangent t_2 of d in p_2 lies in the corresponding helical tangent plane τ_2. When t_2 coincides with the intersection point of t_1 and τ_2, the line segment p_1p_2 of d can be approximated by a Bezier curve (e.g. [Fole90]). Another approach (a rational model of the surface swept by a curve) is given in [JoWi95].
Figure 7 shows how two successive c-lines can have quite different behavior (details from Figure 6b). The image to the left shows that a self-intersection of the sweep Ψ occurs. The image to the right illustrates the not unusual case when the cutter's axis is parallel to the instantaneous helical axis and the cutter path reaches a relative maximum.

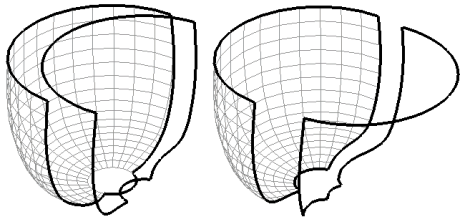


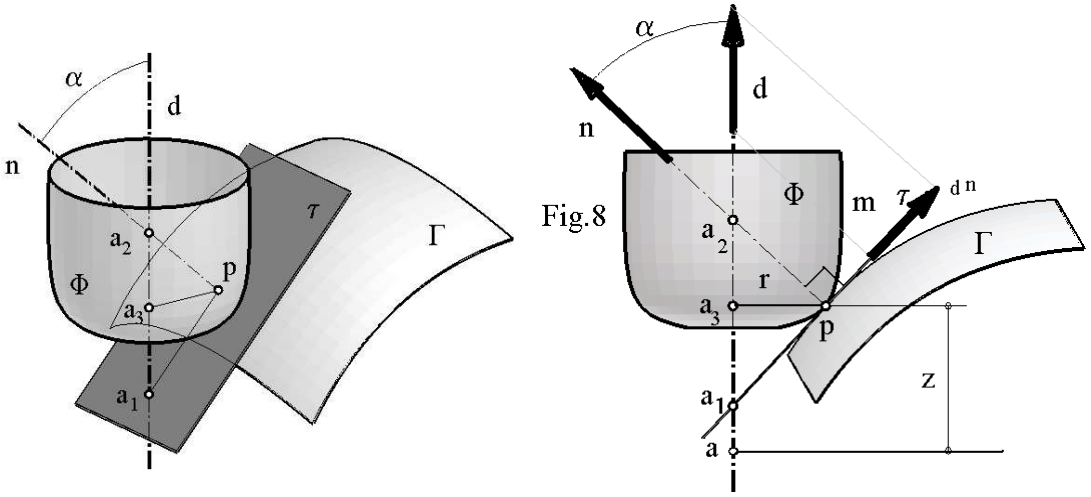
Fig.7

3 Tool Path Generation

Let Γ be the C^1–surface that has to be milled (the design surface). In each point $p \in \Gamma$ (position vector **p**), we have a surface normal **n** of Γ perpendicular to the tangent plane τ. The cutter Φ is considered to be a surface of revolution. When the direction vector **d** of the cutter axis *a* is given, it will in general be possible to find a point $p_0 \in \Phi$ where the normal of Φ has the direction of **n**. If not, a point p_0 on a bordering circle can be taken. We can move the cutter so that p_0 and *p* coincide. Now Φ and Γ touch in *p*. When Φ is convex, the correspondence $p \rightarrow p_0 = p_0(p, \mathbf{d})$ is well-defined.
Let α be the angle between **d** and **n**. It can also be interpreted as the angle between the tangent of the cutter's meridian and a base plane perpendicular to the cutter's axis (Figure 8). For further calculations, we parametrize the meridian of the cutter Φ by means of the angle α:

$$r = r(\alpha), \quad z = z(\alpha) \qquad (0 <= \alpha < \pi / 2).$$

We now look for points on the cutter's axis, provided that Φ is convex and *s* is a regular point on the cutter's meridian.



According to Figure 8, the position vectors to the three points a_1, a_2, a_3 on the cutter's axis are given by the vector equations

$$\mathbf{a}_1 = \mathbf{p} - r / (\sin\alpha \cos\alpha)\, \mathbf{d}^n, \quad \mathbf{a}_2 = \mathbf{p} + (r / \sin\alpha)\, \mathbf{n}, \quad \mathbf{a}_3 = \mathbf{a}_2 - (r / \tan\alpha)\, \mathbf{d}.$$

($\mathbf{d}^n$ is the normal projection of **d** on the tangent plane τ with the length sin α; **n** and **d** are normalized.) We can even describe a fixed reference point *a* ($\mathbf{a} = \mathbf{a}_3 - z\,\mathbf{d}$) on the cutter's axis:

$$\mathbf{a}(\mathbf{p}, \mathbf{d}) = \mathbf{p} + (r(\alpha) / \sin\alpha)\, \mathbf{n} - (z(\alpha) + r(\alpha) / \tan\alpha)\, \mathbf{d}.$$

The above equation describes a surface that carries the path curves of *a* during any milling process along the surface. In the case of 3-axis milling (where **d** is constant), such surfaces are called „general offset surfaces". They were introduced by Brechner [Brec92] and more carefully investigated by Pottmann [Pott97].

When we want to mill the surface Γ along an arbitrary line of contact $p(t) = \{p\}$, we can choose a set of corresponding axis directions $\mathbf{d}(t) = \{\mathbf{d}\}$. The parameter *t* is a motion (time) parameter. (For 3-axis milling, $\mathbf{d}(t)$ is constant). $p(t)$ and $\mathbf{d}(t)$ determine the motion of the cutter axis in a unique way. This allows to determine the instantaneous helical motion $\mathfrak{I}(t)$ as follows:

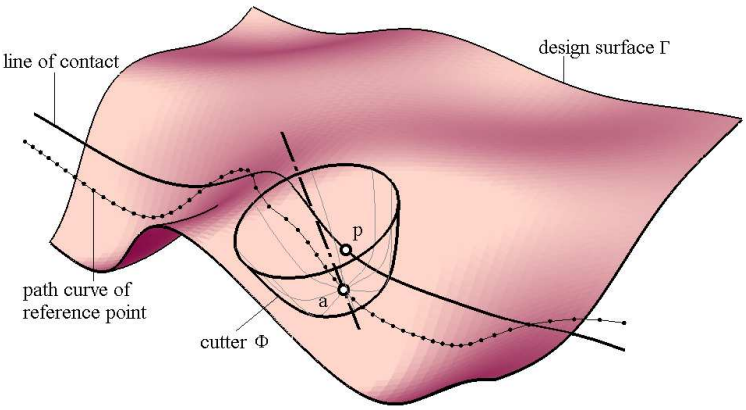


Fig.9

We consider three „neighboring" axes, corresponding to the times $t - \varepsilon$, t and $t + \varepsilon$ ($\varepsilon > 0$), where $a^{-\varepsilon}$, a and $a^{-\varepsilon}$ are the previously mentioned fixed points on these axes, and $\mathbf{d}^{-\varepsilon}$, **d** and $\mathbf{d}^{+\varepsilon}$ are the corresponding direction vectors. Let *c* be the circle

determined by $a^{-\varepsilon}$, a and $a^{+\varepsilon}$. Then we can define three Cartesian systems $\Sigma^{-\varepsilon}$, Σ and $\Sigma^{+\varepsilon}$ as follows. The origin is the fixed point, the *z*-direction is the corresponding axis-direction, and the yz-plane contains the corresponding tangent of the circle c. Thus we get a helical $\Im^{-\varepsilon}$ that transforms $\Sigma^{-\varepsilon}$ into Σ and another one that transforms Σ into $\Sigma^{+\varepsilon}$. When ε converges to zero, the circle *c* converges to the osculating circle of the path of *a*, and the two helical motions converge to one helical motion which may be interpreted as the corresponding instantaneous helical motion $\Im$.
Note that in the case of three-axis milling (where the neighboring axes are parallel) the helical axis has the same main direction. The helical motion $\Im$, however, is in general not degenerated to a simple rotation or translation!

Figure 9 shows the path of a fixed point on the cutter's axis when the cutter is moved along the design surface.

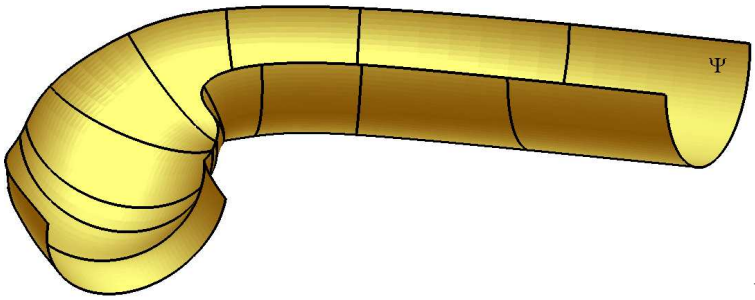


Fig.10

For a comparatively small number of instantaneous helical motions $\Im(t)$ ($t = t_1, t_2, ..., t_n$), we can now quickly determine a one-parametric set of characteristic curves $c(t)$. They are parameter lines on the surface of the swept volume Ψ. The time differences Δt should in general not be constant (Figure 10). As a rule of thumb, Δt can be increased when $\Im(t)$ is nearly a translation or rotation. For a pure translation, we only need the characteristic curve of the start position and of the final position.

4 The Γ-Buffer Representation of Surfaces

In [SaTa91], a method called "G–Buffering" for the representation of surfaces (function graphs) is introduced. A G–Buffer ("Geometry Buffer") is used to store various information per pixel: depth, normal vector, patch or object identifier, and other patch information. The method is viewpoint dependent and has some additional drawbacks, as mentioned in Section 1.
We now want to introduce a related object-space oriented Buffer that we will call Γ–Buffer. It will enable us to simulate the milling process, supporting Boolean subtractions very efficiently.

4.1 Γ-Representation of Function Graphs

Let the design surface be a function graph Γ, defined over a base rectangle $x \in [x_1, x_2]$, $y \in [y_1, y_2]$. The graph need not be represented by a mathematical function $z = f(x, y)$. It should rather be an arbitrary polygonized surface. (Depending on the complexity, the number of polygons may vary from only a few up to tens of thousands). In a top view on the *xy*–plane, the polygons need not obey any pattern like regular grid arrangement.

We now define a Γ–Buffer.
In terms of object-oriented programming, a Γ–Buffer is meant to be a class with the following members:

- A linear coordinate transformation from the rectangular base $\{ (x, y) \mid x \in [x_1, x_2], y \in [y_1, y_2] \}$ to the rectangle $\{ (u, v) \mid u \in [0, n_1], v \in [0, n_2] \}$ plus its inverse.
- A dynamic list of polygons (faces) with information about normal vector and vertices (in *xy*-coordinates) and physical properties.
- A large two-dimensional cell-structure ($n_1 \times n_2$ cells, $n_1, n_2 \approx 1000$), being placed as a fine grid over the base *uv*-rectangle. Each cell of the grid now contains an index that uniquely indicates the corresponding surface, two floating point *z*-values and a pointer to the polygon on the surface. The first *z*-value is used for the representation of the workpiece, the second one represents the design surface.
- Member functions for initializing, updating and displaying, read-write options for disk-storage and error-assessment.
- The most frequently used member function is to put a polygon into the polygon list and buffer its contents. It works as follows: The data of the polygon (vertices, normal etc.) are stored. Then the *xy*-coordinates of the vertices are transformed into the *uv*-base. The transformed polygon defines a plane γ. For each grid point inside the *uv*-projection of the polygon, the *z*-coordinate in γ is buffered in the *first* *z*-value of the corresponding cell (that means, it is stored when the existing *z*-value is larger than the current one). This can be done very quickly and is similar to Z–buffering (Z–buffering, however, is done in image space and is thus viewpoint dependent, whereas Γ–Buffering is not!). In addition to the *z*–values, we also store the pointer to the corresponding polygon and a unique index (> 0) for the surface.

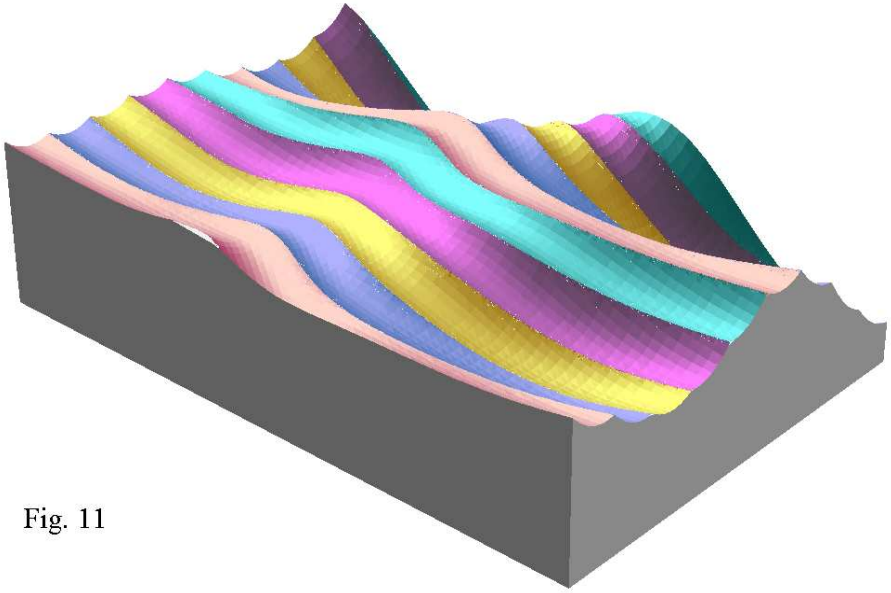


Fig. 11

The milling process can then be simulated by a Γ–Buffer in several steps:

1. The cells are initialized: The first *z*-value is initialized by a theoretical maximum number (e.g., $z = 10^{10}$). The surface indices are all initialized by zero. For quick error assessment, the second *z*-values have to be initialized: All the polygons of design surface Γ are put into the buffer in a modified way: The *z*-values of the polygon's plane γ are stored in the *second* *z*-values.
2. We buffer the polygons of the raw stock's surface (this might be the top plane of a simple box or any other surface).
3. We update the buffer with the polygons of the sweeps Ψ_k (each Ψ_k has a distinct surface number). The non-constant first *z*-value must never be smaller than the constant second one, otherwise the cutter over-cuts the design surface Γ. For the time being, we only buffer those polygons of the sweeps Ψ_k that are „on the lower side". (In Section 4.3 we will see how we can also do „pocketing").
4. At any time, we can display the contents of the Γ–Buffer in an efficient way, using a recursive algorithm (see implementation in Section 4.2).
5. At the end of our simulation (or at any time in between) we can estimate with high accuracy the amount of material that still needs to be removed: For each cell of the grid, we subtract the second *z*-value from the first one. This *z*-difference is proportional to the volume above the design surface.

4.2 Implementation of the Γ-Buffer

Before we extend Γ-buffering to general surfaces, we briefly want to discuss the implementation of a Γ-Buffer.
The fine grid over the *uv*-rectangle ($n_1 \times n_2$ grid points) makes it necessary to allocate vast amounts of memory. In our representation, we used grid sizes like $n_1 = n_2 = 1024$. (In order to accelerate the display-function, n_1 and n_2 should be dividable by 16 .)
Each cell of the grid contains a 2-byte integer (the index of the corresponding surface), two 4-byte floating point *z*-values and one 4-byte pointer to a polygon, which makes 14 bytes altogether.
Thus, the grid itself requires about 15 Mbytes RAM. Compared to this memory request, the polygons themselves need comparatively little space (approximately 1 Megabyte for 50,000 polygons).
The buffering process itself can be implemented very computation-inexpensively (on any platform that we used, even on a 200 MHz PC, thousands of polygons can be processed per second).
From time to time, the number of necessary polygons can be reduced by buffering all the polygons once again (this will not change the contents of the buffer). When the grid did not have to be updated by a polygon, this polygon can be removed from the Γ–buffer. In this way, even complicated buffers rarely exceed 20,000 polygons.

An important member function of the Γ–Buffer is the displaying of the contents.
The simplest – but unacceptably time-consuming – way is to display the $2 \times n_1 \times n_2$ triangles defined by the grid. This number can be reduced drastically without any loss of image quality, by using a recursive algorithm:
For a patch over say 16×16 grid points, we test whether all these grid points belong to the same surface (surface index!). If that is so, we can display the surface over the patch quickly by means of two triangles. If not, we subdivide the patch into four squares of 8×8 grid points and repeat the process. The recursion ends when the square size is 1×1. In this case, we display the patch in any case, provided the surface index is not zero. The latter enables us to display surfaces with non-rectangular top view and/or holes (Figure 12).

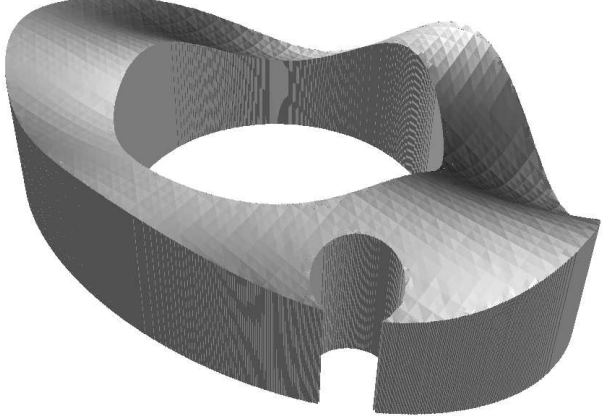


Fig.12

This recursive display algorithm produces an equally good result as when we display all the 2-3 million triangles that the Buffer potentially provides, especially when we work with smooth shading. In Figure 11, the reasonable number of 20,000 polygons had to be displayed. On a 200 MHz Silicon Graphics workstation (Indigo 2), the surface can be displayed several times a second.

4.3 Γ-Representation of General Surfaces

So far we only discussed design surfaces Γ and sweeps Ψ_k that are function graphs (i.e., surfaces with only one *z*-value over a base point (*x, y*)).
With slight modifications, we can cover surfaces with a maximum of two intersection points on any *z*-parallel line (i.e., line perpendicular to the ground plane). Then we can split the surface into two function graphs, and apply two separate Γ–Buffers to the scene. Figure 13 shows an non-trivial example for this (the *z*-direction is chosen in such a way that we fulfill the restriction to the number of intersection points in that direction).

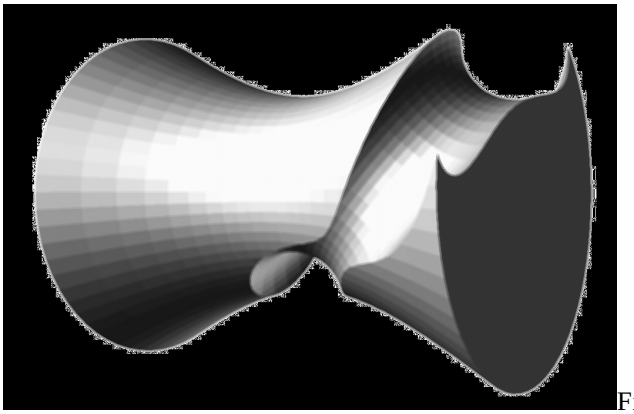


Fig.13

We now have to provide a more general solution in order to be able to do „pocketing“ and/or to mill general surfaces and/or to apply general 5-axis milling. For this purpose, we extend the definition of the Γ–Buffer:

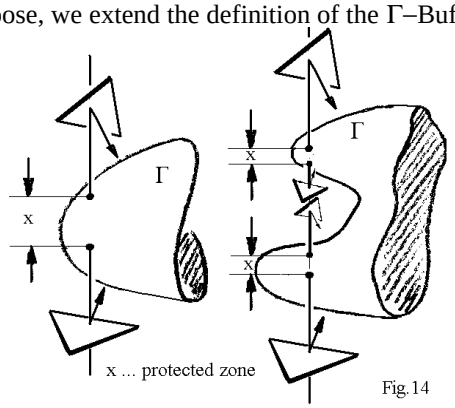


Fig.14

On a general surface, several *z*-values over a base point (*x, y*) are possible. Therefore, the grid cells must be extended. The idea is to store several „protected zones“. Figure 14 shows what is meant with a protected zone: It is a zone, where the cutter must not intersect the *z*-parallel through (*x, y*). When *m* is the maximum number of possible intersection points of Γ with an arbitrary *z*-parallel ray, the contents of the extended cell is now

- the current number *n* of protected zones ($n \leq m / 2$),
- the current number *k* of detected intersection points of the z-parallel ray with the polygons of the buffer ($k \leq m$),

- *m* floating point z-values z to these intersection points,
- *m* pointers to the corresponding intersecting polygons of the grid.
- For error assessment, we allso store *m* floating point *z*-values z_0 (minima and maxima of the protected zones of the design surface Γ). They indicate the 2*n* intersection points of the vertical ray above the grid point with Γ.

(Each extended grid cell now requires $12m + 2$ bytes altogether. For $m > 2$, this is quite memory intensive!)
The extended Γ–Buffer is filled and displayed in several steps:

1. The Buffer is allocated. The extended cells are initialized with $n = k = 0$.
2. The polygonized design surface Γ is stored. We process each polygon **P** of Γ as follows: We transform **P** into the (*u*, *v*)-base as usual. For each extended grid cell inside the (*u*, *v*)-projection of **P**, the corresponding z_0-value is stored. When the normal **n** is horizontal, i.e., when **P** is perpendicular to the (*u*,*v*)-plane, we only store points on the sides of **P**. –– After all the polygons of Γ have been proceeded, the *n* detected z_0-values are sorted. Alternately, the z_0-values indicate the beginning and the end of a protected zone.
3. The surface of the raw stock is put into the buffer similar to how the design surface was stored. The normal vectors of the polygons have to be oriented to the *outside* of the surface. We store *k* *z*-values plus additional information (pointer to the intersecting face).
4. The sweeps Ψ_k are buffered (the Γ–Buffer is updated). The normal vectors of the polygons are oriented to the *inside* of the surface. When it comes to the buffering of a polygon, we determine for each cell the corresponding *z*-value ζ in the polygon's plane. This value must never be inside a protected zone, otherwise the sweep over-cuts Γ. Let the oriented normal vector be turned downwards. When ζ is greater than a buffered maximum and smaller than the following minimum, the maximum is updated (including polygon pointer). Analogously a minimum is updated when the oriented normal vector is turned upwards.
5. The contents of the extended Γ–Buffer can be displayed in a very similar way as the contents of a conventional Γ–Buffer. We start to check quadrangular patches with size 16×16. For all grid cells in this patch we first check whether the levels *k* are the same. If that is so, we can display all the *k* parts of the buffered object as described above (locally, we have *k* Γ–Buffers at our disposal). If not, we recursively split the patch into 4 smaller patches. The recursion ends when the minimum patch size 1×1 is reached. Then this patch is displayed, when the level of at least two of the 4 neighboring grid points are the same. (Otherwise this might build up unwanted „curtains" from lower points to higher ones). Figure 15shows an example of a workpiece with pocketing by means of a ball cutter. In this case, we had a maximum of two protected zones ($m = 2$).

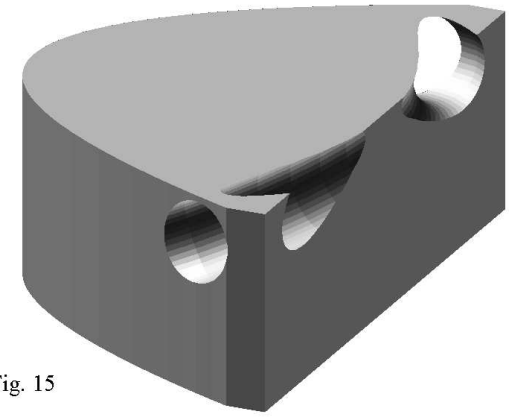


Fig. 15

Although the computational expense and the request of memory increase for complex surfaces, Γ–buffering still is robust and works comparatively fast.

5 Conclusion and Future Work

The simulation of milling processes can be done effectively and without restrictions with the algorithms described in this paper. When a workpiece is stored in a Γ–Buffer, it can be shaded in very short time (several times a second on graphics workstations and – depending on the complexity of the surface – little more than one second on a 200 MHz Pentium PC with graphics hardware). The milling process itself can also be done very computational-inexpensively by making intensive use of the geometric results explained in this paper. The simulation is very accurate since it uses results from differential geometry.

Since these algorithms are very robust, it will now be possible to continue with further geometrical research towards improvements of NC-milling.

Here are two important questions, to which a final answer is not in sight so far (but to which the simulator can quite effectively verify or falsify various suppositions for a number of samples):

- How are the lines of contacts to be chosen in order to minimize the number of path-sequences? For example, it is conjected that such lines might be the „relative principal curvature lines“ of „generalized offset surfaces“ ([Pott97])
- Which shape of the cutter is best for a specific surface? E.g., for a certain class of cutters (and 3-axis milling of a function graph) it is possible to prove the following theorem: If the cutter does not intersect the graph locally (in a small area around any point of contact), there will be no under-cuttings. To determine the optimal cutter among the possible ones is an important challenge.

In general, it can be said that 5-axis milling has so many degrees of freedom that it will require lots of research to get a better understanding of what is possible and what is not.

6 ACKNOWLEDGMENTS

This research has been supported by the Austrian Science Foundation through project P11357-MAT. The authors thank Helmut Pottmann and Hellmuth Stachel for critical comments and Andreas König for his help with the introductory section.

REFERENCES

[Brec92] E. Brechner, "General Offset Curves and Surfaces", in R.E.Barnhill, ed., "Geometry Processing for Design and Manufacturing". SIAM, Philadelphia, pp. 101-121 (1992).

[Chap83] I. T. Chappel, „The use of vectors to simulate material removed by numerically controlled milling“, *Computer Aided Design*, Vol. 15, No. 3, pp.156-158, (1983).

[Fole90] J. Foley van Damm, „Computer Graphics. Principles and Practice“, *Addison-Wesley* (1990).

[Hook86] T. van Hook, "Real Time Shaded NC Milling Display", *Computer Graphics*, Vol. 20, No. 4, (*Proc. SIGGRAPH '86*), pp.15-20 (1986).

[Hui94] K.C., Hui, “Solid sweeping in image space- application in NC simulation” , *The Visual Computer* , Vol.10, pp.306-316 (1994).

[HuOl94] Y. Huang, and J.H. Oliver, "NC Milling Error Assessment and Tool Path Correction", *Computer Graphics Proceedings*, Conference Proceedings July 24-19, 1994, (*Proc. SIGGRAPH '94*), pp. 287-294.

[JeDr89] R.B. Jerard, R.L. Drysdale, K. Hauck, J. Magewick, and B. Schaudt, "Methods for Detecting Errors in Numerically Controlled Machining of Sculptured Surfaces", *IEEE Computer Graphics and Applications* , 0272-1716/89, pp.26-39 (1989).

[JeHu89] R.B Jerard, S. Z. Hussaini, R.L. Drysdale, and B. Schaudt, "Approximate methods for simulation and verification of numerically controlled machining programs", *The Visual Computer* , Vol. 5, pp.329-348 (1989).

[JoWi95] J.K. Johnstone, and J.P. Williams, "A rational model of the surface swept by a curve ", *EUROGRAPHICS’95, Vol.14, pp.77-88* (1995).

[KaIt91] Y. Kawashima, K. Itoh, T. Ishida, S. Nonaka, and K. Ejiri, "A flexible quantitative method for NC machining verification using a space-division based solid model", *The Visual Computer* , Vol. 7, pp.149-157 (1991).

[LiEs96] Ch. Liu, and D.M. Esterling et al. „Dimensional Verification of NC Machining Profiles Using Extended Quadtrees“, *Computer Aided Design*, Vol. 28, No. 11, pp.845-842 (1996).

[Pott97] H. Pottmann, "General Offset Surfaces", *Neural, Parallel & Scientific Computations*, 1997, to appear.

[Rvac74] V.L. Rvachev, "Methods of Logic Algebra in Mathematical Physics", *Kiev: Naukova Dumka Publishers*, p259, (1974).

[SaTa91] T. Saito, and T. Takahashi, "NC Machining with G-Buffer method", *Computer Graphics*, Vol. 25, No. 4, (*Proc. SIGGRAPH '91*), pp.207-216 (1991).

[Sour96] A.I. Sourin, and A.A. Pasko, "Function Representation for Sweeping by a Moving Solid", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 2, No. 2, March 1996, pp.11-18 (1996)

[StEl96] H. Stachel, and A. Elsonbaty, "Generating solids by Sweeping Polyhedra", 1996. *Proc. 7th ICECGDG*, Cracow 1996, Vol.I, pp. 245-249 (to appear in *Journal for Geometry and Graphics 1* (1997)).

[VoWa81] H.B. Voelker, and K.K. Wang, "The Role of Solid Modeling for Swept Volume of Moving Solids." *SAE Technical Paper no. 810195*, Feb. 1981.

[WaKa95] S.W. Wang and A. Kaufmann, "Volume Sculpting" *ACM Symposium on Interactive 3D-Graphics, Monterey CA*., pp.151-156.

[WaWa86] W.P. Wang and K.K. Wang, "Geometric Modeling for Swept Volume of Moving Solids." *IEEE Computer Graphics and Applications*, 6, 12, 1986, pp.8-17.

[Wund67] W. Wunderlich, "Darstellende Geometrie II", BI-Hochschultaschenbuch, Mannheim 1967.

[YaLe96] M. Yang and E. Lee, "NC verification for wire-EDM using an R-map", *Computer Aided Design*, Vol. 28, No. 9, pp.733-740 (1996).