Precise Global Collision Detection in Multi-Axis NC-Machining

Oleg Ilushin[a*], Gershon Elber[b†], Dan Halperin[c‡], Ron Wein[c§]

[a]Department of Applied Mathematics, Technion

[b]Department of Computer Science, Technion

[c]School of Computer Science, Tel-Aviv University

Abstract

We introduce a new approach to the problem of collision detection in multi-axis NC-machining. Due to the directional nature (tool axis) of multi-axis NC-machining, space subdivision techniques are adopted from ray-tracing algorithms and are extended to suit the peculiarities of the problem in hand. We exploit the axial-symmetry inherent in the tool's rotational motion to derive a highly precise polygon-tool intersection algorithm which, combined with the proper data structure, also yields efficient computation times. Other advantages of the proposed method is the separation of the entire computation into a preprocessing stage that is executed only once, allowing more than one toolpath to be efficiently verified thereafter, and the introduced ability to test for collisions against arbitrary shaped tools such as flat-end or ball-end, or even test for interference with the tool holder or other parts of the NC-machine.

Keywords: NC-machining, 5-axis machining, collision detection and verification, space subdivision, ray tracing, lower envelopes.

1. Introduction

In recent years, evermore complex surfaces have been evolving in various engineering processes and designs, driving the demand for more efficient and accurate machining of such surfaces [**?**]. 5-axis machining offers many advantages over traditional 3-axis machining, such as faster machining times, better tool accessibility and improved surface finish. Yet, there are still difficult geometric problems to solve in order to take full advantage of 5-axis machining. One of the most critical problems in machining sculptured surfaces is collision detection and avoidance.

Research in the area of 5-axis machining has been mostly focused on generating proper cutter toolpaths, optimizing tool orientation (for maximal material removal rates and minimal scallop heights), and solving local interference problems (gouging). Verma [19] offers an image-space approach to simulating multi-axis NC milling machines that use a dexel-buffer structure for the workpiece and tool representations, and simulates material removal by boolean subtraction between the two. In this representation a dexel is the basic volume element represented by a rectangular box prolongated along the positive z direction. Müller et al. [16] extend the idea of dexel volumes to multi-dexel volumes. In this approach more than one dexel model is used for representation of a solid, with each model having different direction of dexels. In this manner the difficulty of unequal sampling densities dependent on the slope of the machined surface relative to the direction of the dexels is overcome. In [7], Elber and Cohen derive the boundaries of accessible (gouge-free) regions of a freeform surface with respect to some given check surface by projecting the check surface onto the given surface along a given orientation field. Elber [6] introduced a toolpath optimization method, in which he classifies the surface into convex, concave and saddle-like regions. As a result, applying a flat-end cutting tool to convex regions and a ball-end tool for other regions yields better material removal rates and smaller scallop heights. At the same time this method guarantees gouge-free milling. Yet another approach to the tool orientation optimization was suggested by Lee et al. in [14], where they search the configuration space for optimal tilt and yaw angles of the tool in order to minimize cusp heights in the resulting surface.

While some algorithms were developed to avoid global collision between the tool and the machined part, in 5-axis machining [21,15] these methods do not allow for a general form of a tool and assume a cylindrical approximation for it. The first method, [21], utilizes convex hulls in order to quickly find the feasible set of tool orientations, and in the case of a collision, a correction vector is calculated in the direction opposite to the surface normal vector, at the interfering point. In [15], the collision detection is integrated into the toolpath generation stage. Once collision is detected, the collision vector is derived from the center of collision curve – the curve of intersection of the machined part and the cylindrical approximation of the tool, in the direction perpendicular to the tool axis. This is used later to calculate the correction vector. [17] and [4] allow more general representation of tool geometry for the purpose of collision detection in 5-axis machining. They use a point-cloud representation for the work-

*olegi@tx.technion.ac.il

†gershon@cs.technion.ac.il

‡danha@tau.ac.il

§wein@tau.ac.il

piece, a Constructive Solid Geometry (CSG) representation for the tool and an efficient bounding volumes hierarchy, thereby reducing the interference problem to simple point inclusion queries. However, this representation tends to lose efficiency and requires a substantial amount of memory as the number of sampled points increases, which is the case when one requires a good approximation for the machined part. Also, interference between the tool and other parts of the NC-machine, such as clamping devices, the rotating table and the spindle, is not handled in these methods. [?] offers a toolpath verification method based on the sweep plane algorithm. Parallel slices of the workpiece, the reference part and the tool geometry are computed, with constant intervals. The resulting geometry of the part being machined is obtained by performing the intersection between these slices. This method allows general tool geometry and supports collision detection between the workpiece with the tool, tool holder and fixtures. The precision of such approach, however, depends on the distance between the slices and on the behavior of the surface.

In this work, we aim to develop an efficient and precise algorithm for global collision detection and avoidance in a 5-axis machining context. We expect to take into consideration not only the cutting tool and the machined part, but also other parts of the NC-machine, such as clamping devices and rotating table, as well as impose no restrictions on the shape of the tool. We offer, for the first time to the best of our knowledge, to take advantage of the inherent axis-symmetry of the problem of multi-axis NC-machining.

The paper is organized as follows. In Section 2, we define the problem of collision detection in multi-axis machining and introduce our approach. Section 3 gives some background on lower envelopes and ray tracing – tools that we utilize in our algorithm. Section 3.1 and its three subsections elaborate on the description of the lower envelope construction algorithm. In Section 3.2, we examine computer graphics rendering techniques which we hereby adapt toward efficient extraction of a set of potentially intersecting polygons. In Section 4, we describe the data structure used in our algorithm. Section 5 presents a fast tool-polygon interference testing algorithm. Section 6 introduces some experimental results of our algorithm and finally, in Section 7, we give concluding remarks and outline directions for extending this work.

2. Algorithm Overview

The general collision detection problem can be stated as follows:

> Given a workpiece, a tool, the tool position and orientation, detect and possibly correct any interference, if exists, between the tool and the workpiece or between other parts of the NC-machine, such as the chuck and spindle, the rotating table, clamping devices, etc.

In this work, we assume that the workpiece and the parts of the NC-machine that are to be checked for collision are given in a polygonal representation. As for the tool, arbitrary polyline representation is allowed, as will be shown.

At the preprocessing stage, a data structure will be constructed, which we will refer to as the Line-Distance Query (LDQ) data structure. For a given polygonal model, the following operations should be efficiently performed using the LDQ data structure, at runtime:

1. Given a cutter position and tool orientation (a ray), find the set of polygons that are close to the tool by computing, for example, their inclusion or intersection with a bounding cylinder around the tool. Alternatively, use a set of concentric cylinders and cones that will closely approximate the tool, the spindle and other rotating parts.
2. Given local small changes in the geometry of the workpiece (due to the nature of the machining process), tool position or tool orientation, propagate these changes into the LDQ data structure.

Once the preprocessing stage has been completed and the LDQ data structure has been constructed, we initiate the query, for the given cutting position and tool orientation, and collect the set of polygons that are close to the tool axis with respect to some approximation of the tool mentioned above (such as bounding cones and cylinders). These polygons may originate from the workpiece as well as from the parts of the NC-machine and serve as potential candidates for a precise interference test with the cutting tool.

Given this potentially interfering set, we derive planar hyperbolic segments which originate from the radial projection of the triangles' points around the tool's axis onto a plane, for each triangle in the set. These hyperbolic segments are then tested for intersections with the tool's profile and all other rotating parts such as the chucks and spindle. Such intersections will identify the collision between the rotating parts of the NC-machine and the workpiece or other stationary parts such as a fixture or the base table. Alternatively, we calculate the *lower envelope* of the radial projection of all the polygons in the set with respect to the tool's axis, where the projection is onto a plane through this axis. The lower envelope is a planar contour, which is then checked for intersection with the tool. We discuss the advantages of each of these two possibilities in Section 5.

Notice that even though the surfaces as well as the tool profile are represented using linear functions (polygons and polylines), the radial projection gives rise to hyperbolic arcs as well (see Section 5 for details). Our precise machinery can efficiently cope with second order curves; see more details below.

Among the advantages of the presented method are:

- The preprocessing stage needs to be performed only once and is independent of the toolpath (several toolpaths could be verified without the need to rebuild the LDQ data structure).

- A collision test is independent of the initial orientation of the model. This dependency is a significant drawback of many contemporary voxel-based alternatives, which favor major axes.
- The proposed approach is precise to within machine precision.
- No restrictions are posed on the shape of the tool and collision tests for all the rotating parts of the NC-machine are supported.

Devising an efficient LDQ data structure is of great importance in this approach and has a significant impact on the algorithm. Among the possible choices one can consider are an Oriented Bounding Box (OBB) hierarchy or k-DOP, used in [17,4].

3. Background

In this section we briefly discuss two tools that will serve us in our algorithm. In Section 3.1, the idea of a lower envelope is presented whereas in Section 3.2, optimization methods in computer graphics ray tracing are considered.

3.1. Lower Envelopes

Lower envelopes are ubiquitous in computational geometry and will be used here to speed up collision-detection queries for complex tools.

Definition

> Given a collection $\mathcal{C} = \{C_1, \ldots, C_n\}$ of bounded planar x-monotone curves, which can be viewed as univariate functions $C_i(x)$ defined on an interval, the *lower envelope* of $\mathcal{C}$, denoted $\mathcal{L}(\mathcal{C})$, is the pointwise minimum of these functions $\mathcal{L}(\mathcal{C})(x) = \min C_i(x)$, taken over all functions defined at x.

Let us assume that our curves are well-behaved — that is, two curves intersect at k points at most, where k is a constant, or else they are considered to be overlapping. We can then compute the planar arrangement [11] $\mathcal{A}(\mathcal{C})$ of the curves in $\mathcal{C}$, namely the planar subdivision induced by $\mathcal{C}$, whose complexity is $O(n^2)$, and examine the unbounded face of this arrangement. However, we could also compute the lower envelope directly. The complexity of the lower envelope is $O(\lambda_{k+2}(n))$, where $\lambda_s(n)$ is the maximum length of a Davenport–Schinzel sequence [18] of n elements with order s. For small values of s, $\frac{\lambda_s(n)}{n}$ is an extremely slowly growing function of n and the complexity of the lower envelope is "almost" linear. For example, if $\mathcal{C}$ contains just line segments, then $k = 1$ and $\lambda_3(n) = O(n\alpha(n))$, where $\alpha(n)$ is the inverse of Ackermann's function. In our case, we deal with line segments and special hyperbolic segments, such that each pair of curves may intersect at most twice, so the complexity of the lower envelope is $\lambda_4(n) = O(n2^{\alpha(n)})$ (see [18] for more details). This suggests that we can do much better than the $O(n^2)$ algorithm.

Indeed, we use a divide-and-conquer approach. Namely we divide $\mathcal{C}$ into two subsets of equal size, compute the lower envelope of each half recursively and finally merge the two envelopes together. This algorithm clearly runs in $O(\lambda_{k+2}(n)\log n)$ time. (A slightly faster but more involved algorithm, which is also more difficult to implement, is proposed by Hershberger [13].)

Next we describe the data structure we use to represent $\mathcal{L}(\mathcal{C})$ and how it is constructed.

3.1.1. Constructing the Minimization Diagram

A natural representation of the lower envelope $\mathcal{L}(\mathcal{C})$ is by a *minimization diagram*. By this we mean that, given a collection $\mathcal{C}$, we can subdivide the x-axis into maximal intervals, such that the identity of the curve(s) contributing to the lower envelope, above each point in the interval is the same. Notice that the curves are not assumed to be in a general position (i.e., it is possible that more than two curves in $\mathcal{C}$ intersect at a common point, two (or more) arcs may overlap[1], etc.), so this diagram must be carefully defined and constructed.

The minimization diagram $\mathcal{M}(\mathcal{C})$ consists of *minimization vertices* and *minimization edges* (*vertices* and *edges* for short). All the components are stored in a doubly-linked list, where the vertices are sorted by their increasing x-value, and each edge is stored between its two end-vertices. Each vertex v is associated with a point $p(v)$, and each edge e stores a set $C(e)$ of the curves that constitute the lower envelope between the two end-vertices of the edge (it is also possible that $C(e) = \emptyset$). See Figure 1 for an illustration.

To construct the minimization diagram of a collection $\mathcal{C}$ we will use the following procedure:

1. Remove all vertical segments from $\mathcal{C}$ (we will consider these segments only after we are done with the rest). We denote the resulting set by $\hat{\mathcal{C}}$.
2. If $|\hat{\mathcal{C}}| \leq 1$, the construction of the lower envelope is trivial. At any other case:
   - Partition the set into two subsets $\mathcal{C}_1, \mathcal{C}_2$ of equal sizes.
   - Execute step 2 on each of the subsets to compute $\mathcal{M}(\mathcal{C}_1)$ and $\mathcal{M}(\mathcal{C}_2)$.
   - Merge the two minimization diagrams by simultaneous traversal on the two lists of vertices.
3. Sort the vertical segments by their increasing x-values and merge them with the diagram $\mathcal{M}(\hat{\mathcal{C}})$.

The complexity of step 1 is obviously linear and of step 3 is $O(n\log n)$, while for step 2 the running time obeys $T(n) = 2T(\frac{n}{2}) + O(\lambda_{k+2}(n))$, so the entire process takes $O(\lambda_{k+2}(n)\log n)$ time.

[1] Allowing overlaps should not be misconstrued as allowing an arbitrary number of intersection points between a pair of curves. The constant upper bound on the number of intersections is needed in order to guarantee a good bound on the complexity of the envelope and hence on the running time of the algorithm to compute it. Partial overlap between curves does not effect these bounds.

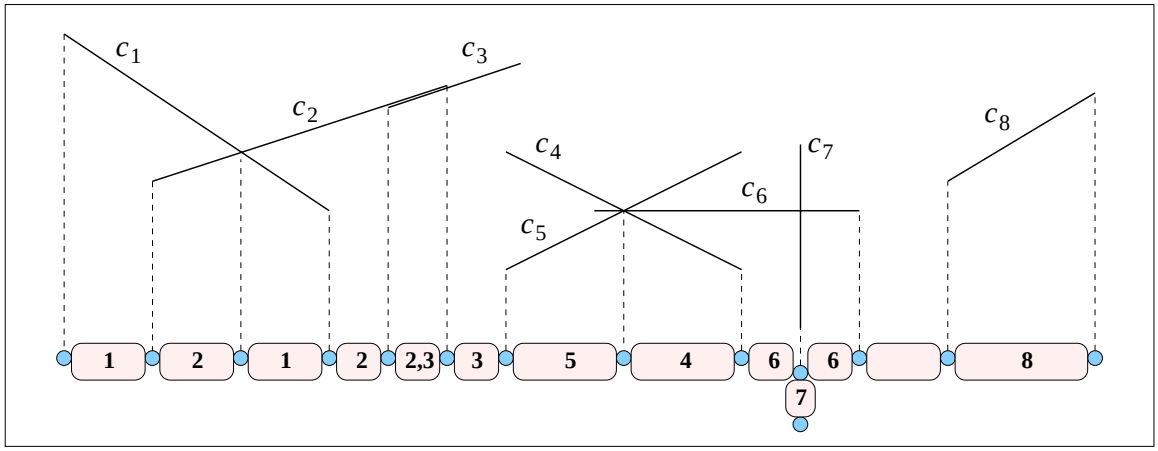


Figure 1. The minimization diagram representing the lower envelope of a set of line segments. Vertices are represented as lightly shaded circles and the list of segments associated with each minimization edge is shown in the rounded rectangles between the vertices. Notice the overlap between c_2 and c_3 and the representation of the vertical c_7.

3.1.2. The Lower Envelope Traits

We have developed a CGAL [1] package that computes the lower envelope of a given set of curves, represented as a minimization diagram, using the algorithm given in the previous section. As CGAL employs the generic programming approach (see [5] for more details), the geometry involved in the algorithm is entirely separated from its topological constructions.

This set of geometric predicates and constructions is a subset of the geometric requirements needed by CGAL's arrangement package [8,12], and should be provided by a so-called *traits* class. Such a traits class is used as a parameter for the arrangement template to instantiate arrangements of curves of a specific type, and can similarly be used to instantiate the lower-envelope template.

Although the line segments and hyperbolic arcs we have to deal with are a special case of conic arcs (see [20] for more details), it is possible to simplify the geometric predicates and constructions involved in the implementation of the traits class if we use the special structure of the hyperbolas we obtain in our case.

We are interested in segments of the upper portion of canonical hyperbolas of the form:

$$\alpha x^2 - y^2 + \beta x + \gamma = 0 \ , \tag{1}$$

which are defined by the x-coordinates x_1 and x_2 of the segment's endpoints. The endpoints of the hyperbolic arc are given by (x_i, y_i), where:

$$y_i = C(x_i) = \sqrt{\alpha x_i^2 + \beta x_i + \gamma} \ . \tag{2}$$

Thus, the 5-tuple $\langle \alpha, \beta, \gamma, x_1, x_2 \rangle$ completely characterizes the canonical hyperbolic arc.

We can see that the traits class for this family of curves can be easily implemented using elementary algebra, for example:

- Given two x-monotone arcs $C = \langle \alpha, \beta, \gamma, x_1, x_2 \rangle$ and $C' = \langle \alpha', \beta', \gamma', x_1', x_2' \rangle$ and their intersection point p, we can conclude which curve is above the other immediately to the right of p if we compare the two slopes, or first-order derivatives of the two curves. Thus, we have to compare $\frac{2\alpha x(q)+\beta}{C(x(q))}$ and $\frac{2\alpha' x(q)+\beta'}{C'(x(q))}$.
- Given two x-monotone arcs $C = \langle \alpha, \beta, \gamma, x_1, x_2 \rangle$ and $C' = \langle \alpha', \beta', \gamma', x_1', x_2' \rangle$, we can compute all their intersection points by solving the quadratic equation:

  $$\alpha x^2 + \beta x + \gamma = \alpha' x^2 + \beta' x + \gamma' \ . \tag{3}$$

  Also notice that no overlaps may occur between C and C', unless, of course, the underlying hyperbolas of the two hyperbolic arcs are the same.

Recall that we need to support line segments as well. This is quite simple as we can write the equation of the underlying line $y = ax + b$ of the segment[2] as:

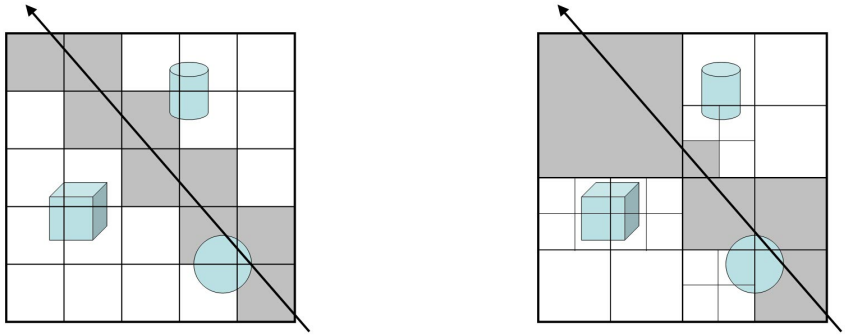
$$a^2 x^2 - y^2 + 2abx + b^2 = 0 \ . \tag{4}$$

Thus we can represent the line segment by the hyperbolic arc $\langle a^2, 2ab, b^2, x_1, x_2 \rangle$.

3.2. The Ray Tracing Connection

A very common task in computer graphics is rendering of three-dimensional scenes. Ray tracing is one of the most popular rendering techniques in use that can produce high-quality realistic images; see for example [10]. Ray tracing simulates camera photography by shooting rays through each pixel in the image into the scene and then tracing these rays recursively as they intersect and interact with the geometry, in reflection and refraction directions. Ray-object intersection calculation is considered the most expensive operation in ray-tracing. While theoretically each ray can

[2] In case of a vertical segment we use a different representation, but we omit the details here.



(a) Uniform Grid (b) Octree

Figure 2. Ray tracing space subdivision schemes. Ray-object intersections are applied only to those objects contained in the voxels along the ray (grey voxels).

intersect any polygon in the scene, in practice rays typically intersect only a small fraction of the geometry. So optimizations are generally required. Among the ray-tracing optimization schemes, Uniform Space Subdivision (Uniform Grid) and Octrees are most frequently used; see Figure 2. These two techniques subdivide the object space into cells or voxels, and in each voxel register the information about the geometry that intersects it. These optimizations significantly speed up the ray-tracing process by traversing rays through the subdivision and performing intersection calculations only with the geometry inside the voxels along the ray path. Uniform subdivision, as its name implies, subdivides the object space uniformly in each of the x, y, and z directions and is the easiest to compute. Linear traversal algorithms for uniform subdivision are the three-dimensional version of the 2D line rasterization algorithm (DDA, see [9]) and are typically called 3D-DDA. The Octree scheme is an adaptive scheme that exploits the scene's geometry, recursively subdividing those voxels that contain the significant scene complexity, thus resulting, in general, in fewer voxels than in Uniform subdivision. Therefore, this method requires less memory and traversal steps. The Octree approach ,however, requires slightly more complex preprocessing and traversal algorithms.

4. The *LDQ* Data Structures

In the process of NC-machining of a sculptured surface, potential interference between the cutting tool and the machined surface needs to be checked at least at each cutter location along the toolpath. That is, for each tool position and orientation we need to perform an intersection test between the tool and the machined part. If we consider the tool to be a cylinder of radius R, the task will be to identify all the polygons of the machined surface that interfere with the cylinder. A key observation is the similarity of this tool interference problem to the task in ray tracing of finding the polygons that intersect with the ray. Our NC-machining application differs in the fact that now instead of a simple ray, we have to consider a ray of a finite thickness, i.e., a cylindrical one. Thus, optimization techniques for ray-tracing algorithms could be applied to the problem in hand with minor alterations.

Due to the fact that the surface that undergoes machining changes in time, the Octree representation is not our first choice for the data structure, since it will be difficult to update. Hence, a uniform grid will better suit our purpose. Nevertheless, as we aim to exploit the advantages offered by the Octree representation, instead of having a simple uniform grid, we use a hierarchy of uniform grids for the LDQ, doubling the grid's resolution on each successive level; see Figure 3. If, for example, the initial resolution is 2 by 2 by 2, and the depth of the hierarchy is 4, then on the lowest level we will be employing a grid with the resolution of 16 by 16 by 16. This representation is identical to a fully expanded Octree, with the convenience of being able to use a simple 3D-DDA traversal algorithm at each level. We denote this specific data structure a *HiGrid*, short for a Hierarchical Grid. The following is a key observation:

> An intersection test of a polygon with a cylindrical ray of radius R is equivalent to the intersection of a simple ray with the 3D-offset of the polygon by radius R.

Hence, in the initialization stage, we place a reference to a polygon in a voxel if the polygon intersects the box that is the offset of the voxel by R; see Figure 4. Toward this end, we utilize the fast box-triangle overlap testing algorithm by Möller [3], which is based on the Separating Axis Theorem for two polyhedrons. Then, in real time, we traverse the HiGrid using a simple ray, collecting all the polygons inside the voxels visited by the ray. The result will be a set of polygons that potentially interfere with the tool. When traversing the HiGrid, we step down to the lower level and continue the traversal of the grid with finer resolution if the number of polygons in the current voxel is greater than some predefined constant. To support changes in the geometry and polygon removal, all instances of every polygon in all the voxels in the HiGrid are linked in a linear list. Hence, the HiGrid is an efficient data structure that

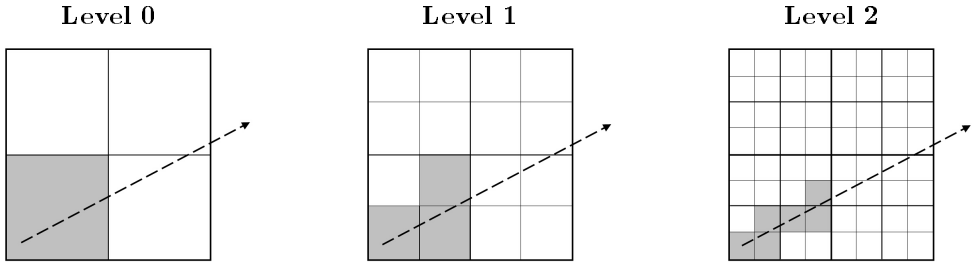


Figure 3. Hierarchical grid levels. Ray traversal at each voxel continues to the lower levels of the grid hierarchy with finer resolution, reducing, thereby, the number of polygons accumulated along the ray.

supports fast queries of Potential-Interfering-Polygons (PIP), given the tool position and orientation and a radius R of the bounding cylinder.

5. Collision Detection

In this work, we assume that the tool is given as a polyline representing a tool's profile to which we will refer as the tool's *silhouette*, and which lies in the first quadrant of the yz-plane; see Figure 5. A similar treatment will apply to a tool whose profile contains quadratic or higher order curves, which will require slightly more intricate calculations.

We denote the *canonical orientation* of the tool as an orientation with the cutter position set at the origin and oriented along the positive z-direction. Let M be the inverse transformation that brings the tool from its current machining position back to the canonical orientation. To test for PIP–tool interference, we first apply M to each triangle from the PIP. We proceed by radially projecting the mapped triangles around the z-axis onto the yz-plane. This radial projection is the trace that the triangle etches on the yz-plane (more precisely, on the half-plane $y > 0$) when rotated around the z-axis.

Consider the line segment AB, $A = (a_x, a_y, a_z)$, $B = (b_x, b_y, b_z)$, rotated around the z-axis (see Figure 6). The trace of AB in the yz-plane is given by the explicit quadratic equation that is derived by looking at the distance between a point on the segment and the z-axis:

$$d^2(z) = \left\{ \frac{1}{(b_z - a_z)} \left[(b_z - z)a_x + (z - a_z)b_x \right] \right\}^2 + \left\{ \frac{1}{(b_z - a_z)} \left[(b_z - z)a_y + (z - a_z)b_y \right] \right\}^2 , \tag{5}$$

where z changes continuously between a_z and b_z.

Now consider a segment of the tool's silhouette, ST, $S = (0, s_y, s_z)$, $T = (0, t_y, t_z)$. The squared distance from point $P \in ST$ and the z-axis for a prescribed value of z is given by:

$$\delta^2(z) = \left\{ \frac{1}{(t_z - s_z)} \left[(t_z - z)s_y + (z - s_z)t_y \right] \right\}^2 . \tag{6}$$

For the segment AB to intersect with the segment ST of the profile of the tool, the right-hand terms of (5)

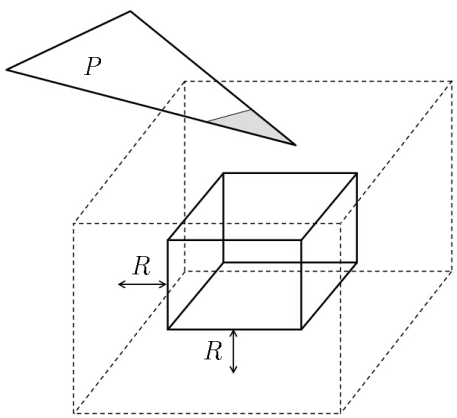


Figure 4. Voxel offset by the tool radius R. Here a reference to the polygon P will be placed in the original voxel since it intersects the offset box.

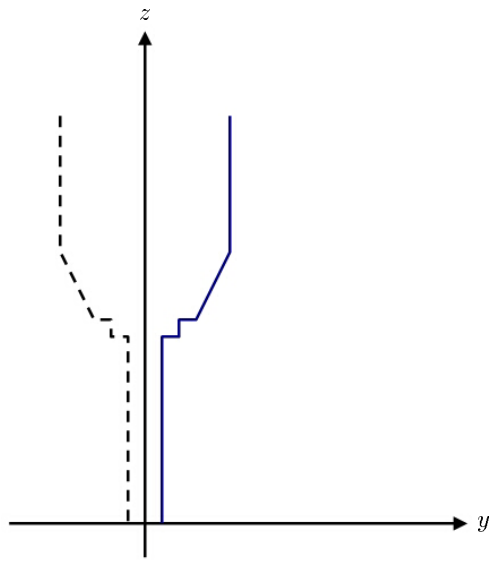


Figure 5. The tool's silhouette: a polyline representing the tool's profile.

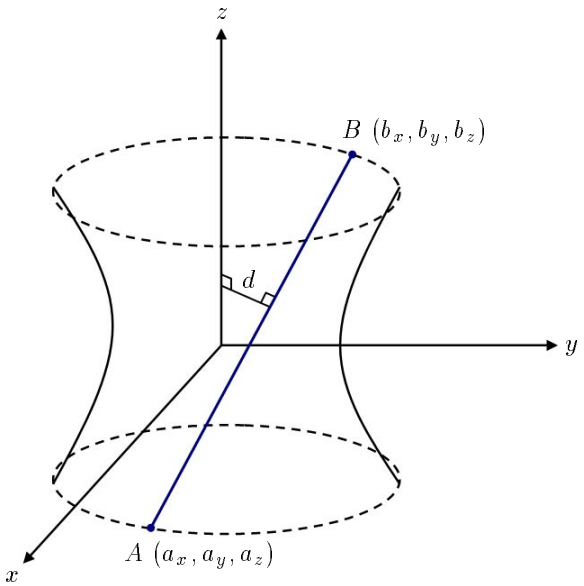


Figure 6. Segment rotated around the z-axis.

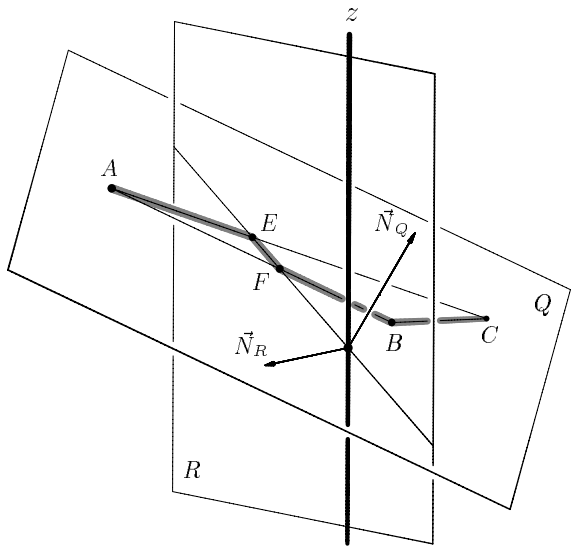


Figure 7. The relevant points of $\triangle ABC$ contributing to the trace lie on the segments AE, EF, FB, BC.

and (6) must be equal. The comparison of these two terms yields the following quadratic constraint for z:

$$\begin{aligned} z^2\left(ab^2 - e(f_x^2 + f_y^2)\right) &+ \\ z\left(2abc - 2e(f_x g_x + f_y g_y)\right) &+ \\ \left(ac^2 - e(g_x^2 + g_y^2)\right) &= 0\,, \end{aligned} \tag{7}$$

where

$$\begin{aligned} a &= \frac{1}{(t_z - s_z^2)}\,, & f_x &= (a_x - b_x)\,, \\ b &= (s_y - t_y)\,, & f_y &= (a_y - b_y)\,, \\ c &= (t_y s_z - t_z s_y)\,, & g_x &= (a_x b_z - b_x a_z)\,, \\ e &= \frac{1}{(b_z - a_z^2)}\,, & g_y &= (a_y b_z - b_y a_z)\,. \end{aligned}$$

By solving Equation (7) for each relevant tool silhouette segment ST and verifying that none of the real roots, if any, lies in a proper range, we can conclude that the segment AB either does or does not intersect the tool.

Now given a triangle $\triangle ABC$, one needs to identify the closest points of $\triangle ABC$ to the z-axis for each value of z. There are several cases to handle; here we will examine the most complicated one. Consider the example depicted in Figure 7. In this example we assume that the triangle's vertices, A, B, C, are in descending z-order. Let Q be the plane containing $\triangle ABC$ and $\vec{N}_Q$ be its normal. Then, for each value of z, the closest points of Q to the z-axis will lie on the intersection between Q and the plane R whose normal $\vec{N}_R$ is given by $\vec{N}_R = \vec{N}_Q \times \mathbf{z}$, where $\mathbf{z} = (0, 0, 1)$, and which contains the z-axis. If R intersects $\triangle ABC$, as in this example, then the closest triangle's points to the z-axis will be lying on segments AE, EF, FB and BC. Segments AE, FB and BC will give raise to hyperbolic curves, while EF will generate a linear curve. Otherwise, and depending on the order of the triangle's vertices, one or two edges of $\triangle ABC$ will be forming the closest points set to the z-axis.

Once the triangle is analyzed and the closest segments are identified and extracted, Equation (5) is applied to each segment. The resulting set of connected hyperbolic arcs and/or line segments is, practically, the lower envelope of the radial projection of $\triangle ABC$ onto the yz-plane.

We proceed and obtain the set of hyperbolic arcs and/or line segments for all the triangles in the PIP. We call this set the *Hyperbolic Segments Set* (HSS). To complete the collision detection test, for the current tool orientation, we have two possibilities.

The first option is the direct approach. In this approach, each curve from the HSS is tested for intersection with the tool's silhouette. Binary search is used on the tool's silhouette segments, along the z-axis. Because each test is performed by analytically solving the quadratic Equation (7), this approach proves to be efficient when the tool's silhouette contains a small number of segments.

The second approach takes advantage of the lower envelope method described in Section 3.1. First the lower envelope of all the curves from the HSS is constructed in $O(\lambda_4(n)\log n)$ time, assuming HSS consists of n curves. Thus, we have potentially reduced the total number of hyperbolic segments to be tested against the tool. Then, the tests for intersections between this lower envelope and the tool's silhouette are performed by a simultaneous traversal over the lower envelope and the tool's silhouette along z and a comparison between the two entities (we assume here that the tool's silhouette is weakly monotone in the z direction). The last

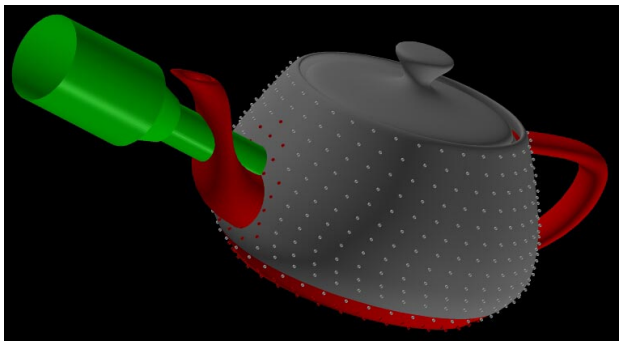


Figure 8. Machining the body of the Utah teapot, using a flat-end tool oriented along the surface normal. Only 1% of the contact points are shown. Red dots and polygons indicate collisions at the cutter locations and the interfering geometry, respectively. The tool's geometry is shown for one contact point with interference.

step is clearly linear in the complexity of the envelope and the tool, making this approach more attractive when a very complex tool's geometry is used, and the tool's silhouette consists of large number of segments. We compare these two above mentioned approaches in Section 6.

6. Results

The collision detection algorithm presented in this paper was implemented in the IRIT modeling environment [2]. The lower envelope calculations and the comparison of the envelope with the tool profile are carried out with recent extension to the CGAL arrangement package [1]. In our tests, we used the Utah teapot model and a wine glass model, both in a 5-axis machining mode. Toolpaths were generated by sampling iso-parametric curves along the model surfaces. For the teapot model, we used a toolpath consisting of $\sim$ 50000 contact points covering the teapot's body with tool orientations following the surface normals. A flat-end tool with a tool holder of a larger radius was used for machining the teapot. In the case of the wine glass model, the toolpath was sampled along an offset surface, having the orientation of the tool set so that its axis line goes through a fixed point. A total of $\sim$15000 contact points were used. The tool that machined the interior of the wine glass is a ball-end tool. Figures 8 and 9 show the output of running the algorithm on both models. The teapot and the wine glass models consist of 12600 and 2700 polygons, respectively. It took about 64.2 seconds and 5.01 seconds, respectively, to test for collisions on a Pentium IV 2.4GHz machine with 512MB of RAM, setting the initial resolution of the HiGrid data structure to 2 and the depth to 4. Red polygons designate geometry that is overlapped by the tool during the simulation process. Tables 1 and 2 show more detailed statistics for different values

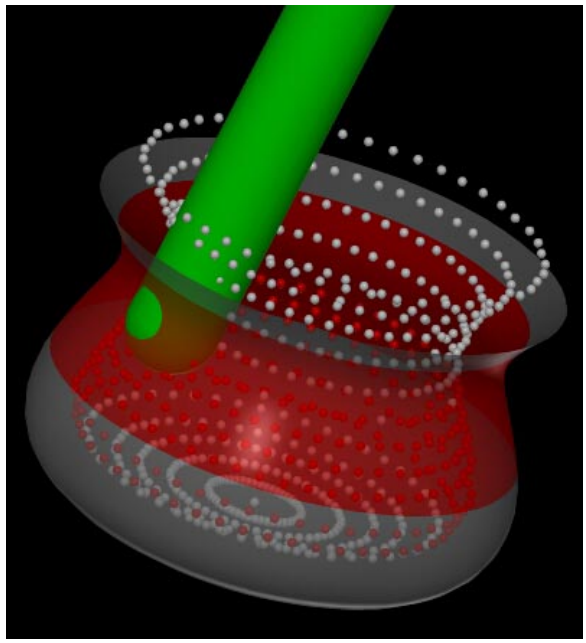


Figure 9. Machining the interior of a wine glass, using a ball-end tool through a point. Only 3% of the contact points are shown. Red dots and polygons indicate collisions at the cutter locations and the interfering geometry, respectively. The tool's geometry is shown for one contact point with interference.

of depth used for initializing the HiGrid data structure for both models. The initial resolution of the HiGrid was set to 2 in all test cases. Larger tool radius in the case of the teapot model resulted in a longer preprocessing time, as each polygon occupies more voxels in the HiGrid data structure when considering a bigger offset. Table 3 demonstrates the advantage of the lower envelope approach when using a very complex tool geometry. Here, we tested the Utah teapot model with the toolpath as in Table 1. A tool with complex geometry was chosen (see Figure 10), and tests with a tool's silhouette consisting of 50, 500, 5000 and 50000 line segments were performed.

7. Conclusions

We have presented a new approach to the problem of collision detection in multi-axis NC-machining which yields, in the tests so far conducted, promising results. Preprocessing can take up a significant amount of time and memory, depending on the tool radius, model resolution (number of polygons) and the HiGrid resolution. Since this preprocessing is performed only once and can be used for more than one toolpath verification, it considerably increases the overall performance by narrowing the calculations to a small subset of the geometry.

In the future, we intend to compute an *exit vector* from the intersection data to support *collision avoid-*

Table 1
The Utah teapot model with 12600 polygons and ∼50000 contact points along the toolpath, using a flat-end tool with a tool holder with a larger radius; see Figure 8.

HiGrid depth	2	3	4
Preprocessing time	0.188 sec.	0.609 sec.	2.594 sec.
Total query time	133.7500 sec.	81.4070 sec.	61.6100 sec.
Avg. PIP extraction time per query	0.0005 sec.	0.0003 sec.	0.0003 sec.
Avg. intersections time per query	0.0020 sec.	0.0012 sec.	0.0009 sec.
Avg. num. of polys. per query	1376.4468	801.7224	574.2602
Avg. num. of intersections per query	18.3600	18.3600	18.3600

Table 2
A wine glass model with 2700 polygons and ∼15000 contact points along the toolpath, using a ball-end tool; see Figure 9.

HiGrid depth	2	3	4
Preprocessing time	0.031 sec.	0.093 sec.	0.422 sec.
Total query time	9.5160 sec.	6.1260 sec.	4.5940 sec.
Avg. PIP extraction time per query	0.0001 sec.	0.0000 sec.	0.0000 sec.
Avg. intersections time per query	0.0005 sec.	0.0003 sec.	0.0002 sec.
Avg. num. of polys. per query	318.4795	192.1181	130.7773
Avg. num. of intersections per query	14.3005	14.3005	14.3005

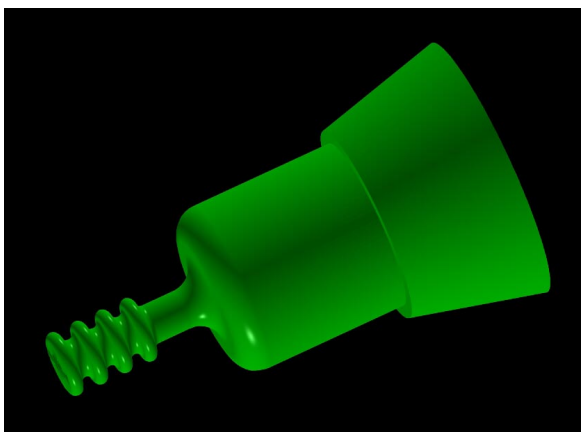


Figure 10. Complex tool geometry.

ance as well as *detection*. Further, extending the tool definition to allow its silhouette to contain quadratic or higher order curves is on our research agenda. We expect to investigate the direct use of polynomial and/or rational surfaces in the near future. As already mentioned, the computation of the lower envelope and its comparison with the tool profile are coded generically so that allowing for higher degree curves will not affect the topological part of the implementation and only local numerical procedures will have to be modified. Furthermore, as long as the tool profile consists of at most second degree curves, our software can already carry out the collision detection precisely in the same way it is done for the polygonal tool profile in the current work, being able to analytically solve degree four polynomial constraints.

So far, we have considered possible collisions at contact points only. Clearly, collisions could occur between contact points and hence missed. We plan on expanding the presented approach and support the representation of lower envelopes that change (slightly) over time between contact points, and detect such intermediate collisions as well.

8. Acknowledgement

This research was supported in part by the Technion Vice President for Research Fund - New York Metropolitan Research Fund and in part by the Israeli Ministry of Science Grant No. 01-01-01509. This work has also been supported in part by the IST Programmes of the EU as Shared-cost RTD (FET Open) Projects under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces) and No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments), by The Israel Science Foundation founded by the Israel Academy of Sciences and Human-

Table 3
The Utah teapot model with 12600 polygons and ∼50000 contact points along the toolpath, using complex tool geometry with the tool's silhouette consisting of up to 50000 line segments. Average time per query (in milliseconds); see Figure 10.

Number of segments in the tool's silhouette	50	500	5000	50000
Direct Approach	0.2	0.4	2.4	19.4
Lower Envelope Approach	4.8	4.7	4.9	4.0

ities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

REFERENCES

1. The CGAL project homepage. `http://www.cgal.org/`.
2. Irit modeling environment. G. Elber, Department of Computer Science, Technion, Haifa, Israel. `www.cs.technion.ac.il/~irit/`.
3. T. Akenine-Möller. Fast 3d triangle-box overlap testing. *Journal of Graphics Tools*, 6(1):29–33, 2001.
4. M. Balasubramaniam, S. E. Sarma, and K. Marciniak. Collision-free finishing toolpaths from visibility data. *Computer-Aided Design*, 35(4):359–374, April 2003.
5. H. Brönnimann, L. Kettner, S. Schirra, and R. Veltkamp. Applications of the generic programming paradigm in the design of CGAL. In M. Jazayeri, R. Loos, and D. Musser, editors, *Generic Programming—Proceedings of a Dagstuhl Seminar*, LNCS 1766. Springer Verlag, 2000.
6. G. Elber. Freeform surface region optimization for 3-axis and 5-axis milling. *Computer-Aided Design*, 27(6):465–470, June 1995.
7. G. Elber and E. Cohen. A unified approach to verification in 5-axis freeform milling environments. *Computer-Aided Design*, 31(13):795–804, November 1999.
8. E. Flato, D. Halperin, I. Hanniel, O. Nechushtan, and E. Ezra. The design and implementation of planar maps in CGAL. *ACM Journal of Experimental Algorithmics*, 5, 2000. Special Issue, selected papers of the Workshop on Algorithm Engineering (WAE).
9. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Fundamentals of Interactive Computer Graphics*. Addison Wesley, second edition edition, 1990.
10. A. S. Glassner. *An Introduction to Ray Tracing*. Academic Press, second printing edition, 1990.
11. D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 21, pages 389–412. CRC Press LLC, Boca Raton, FL, 1997.
12. I. Hanniel. The design and implementation of planar arrangements of curves in CGAL. M.Sc. thesis, Dept. Comput. Sci., Tel Aviv University, Tel Aviv, Israel, 2000.
13. J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.
14. C.-S. Jun, K. Cha, and Y.-S. Lee. Optimizing tool orientations for 5-axis machining by configuration-space search method. *Computer-Aided Design*, 35(6):549–566, May 2003.
15. B. Lauwers, P. Dejonghe, and J. P. Kruth. Optimal and collision free tool posture in five-axis machining through the tight integration of tool path generation and machine simulation. *Computer-Aided Design*, 35(5):421–432, April 2003.
16. H. Müller, T. Surmann, M. Stautner, F. Albersmann, and K. Weinert. Online sculpting and visualization of multi-dexel volumes. In *Proc. 8th ACM Sympos. on Solid Modeling and applications*, pages 258–261, 2003.
17. S. S. S. Ho and Y. Adachi. Real-time interference analysis between a tool and an environment. *Computer-Aided Design*, 33(13):935–947, November 2001.
18. M. Sharir and P. Agarwal. *Davenport–Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.
19. S. Verma. Simulation of numerically controlled machines. Master's thesis, Computer Science Department, The University of Utah, September 1994.
20. R. Wein. High-level filtering for arrangements of conic arcs. In *Proc. ESA 2002*, pages 884–895. Springer-Verlag, 2002.
21. L. Yuan-Shin and C. Tien-Chien. 2-phase approach to global tool interference avoidance in 5-axis machining. *Computer-Aided Design*, 27(10):715–729, October 1995.