Approximate Envelope Reconstruction for Moving Solids

U. Schwanecke and L. Kobbelt

Abstract. We present a new approach to approximatively construct the envelope of a moving solid. It is based on dynamically updating an octree that approximates the envelope surface. Our approach guarantees a prescribed error-bound, and is scalable in the sense that it allows to calculate coarse approximations very fast while better approximations can be obtained by investing more computation time and memory. Furthermore the algorithm is robust due to the fact that no badly conditioned surface-surface intersections has to be computed.

§1. Introduction

A solid object undergoing a motion in general creates a volume. The resulting volume is called a swept volume. Swept volumes play an important role in NC (numerical controlled) machining, robotics and motion planing, e.g. in order to avoid collisions of manipulators. Different approaches to construct the swept volume of a moving solid were developed during the last decades.

One method for representing and analyzing swept volumes is the envelope method (cf. [13]). The main drawback of this method is that there are no really efficient algorithms, due to the essential limitation of efficiently solving the nonlinear envelope equations, and due to the fact that some envelope surfaces tend to resist accurate calculation by both analytical and numerical means.

To overcome the deficiencies of envelope theory, the Sweep Differential Equation (SDE) and Sweep-Envelope Differential Equation (SEDE) method, respectively, were developed (cf. [11]). It subsumes the method of envelopes and is inherently global. The success of the SEDE/SDE method for the numerical computation of swept volumes is largely due to the fact that only a finite set of points (grazing points) need to be calculated at each time step.

All of these analytic methods become extremely complex when the topology of the swept volume gets more complicated. Moreover, none of the mentioned methods is able to satisfactory handle self intersections of the swept

ISBN 0-8265-xxxx-x.

volume boundary that occur for even fairly simple sweeps. This trimming problem for swept volumes is discussed in detail in [2]. The authors give an overview of trimming methods, and develop new trimming strategies for local and global trimming of swept volumes. All of these trimming algorithms consist of computing the candidate set of the boundary of the volume, and in a second step of a test to determine the trim set.

If computational time does not matter, high quality renderings of quite general swept volumes can be obtained by using the Raycasting Engine [17]. For generalized cylinders, i.e. objects defined by sweeping a two-dimensional contour along a three-dimensional trajectory, the computation of the intersection points with a ray can be reduced to the problem of intersecting two two-dimensional curves [6]. A more efficient way of displaying generalized cylinders is the surface scanning algorithm described in [5], which draws contours generated by plane intersections close enough to cover every pixel to which the surface is projected, while avoiding drawing to many superfluous contours. All of these methods only visualize the volume from one specific viewpoint, and cannot address self intersections in an efficient manner.

For the verification of NC machining, several boundary representation and CSG (constructive solid geometry) -based methods have been developed to reconstruct generated objects by moving one object in space. An accurate boundary representation to represent and manipulate solids based on the extension of the octree data structure has been suggested in [7]. This approach can maintain the resulting model, but the processing time per cut, given as a Boolean operation, increases rapidly with part complexity due to the fact that all of the object components have to be cut to each other. Because a part of average complexity requires several thousands and more cuts, the resulting algorithm is too slow for practical use. To overcome this problem, a faster algorithm generating cross sections of the swept profile using extended quadtrees has been given in [15]. However, this approach does not reconstruct the whole object, but only visualizes cross sections, and therefore cannot give a complete impression of the object. In order to construct a three-dimensional model, one has to connect the contours in a post-processing step (c.f. [19]).

In this article we present an algorithm for reconstructing a polygonal approximation of the envelope of a swept solid that automatically generates the topologically correct solution within a prescribed error tolerance. The resulting envelope surface is a guaranteed manifold. In our approach we do not have to pay special attention to the trimming of the swept envelope. As a consequence, the presented algorithm is highly eligible for NC milling simulation for example, where typically a lot of self intersections occur due to the fact that the milling tool usually passes the same surface region many times.

Instead of calculating an algebraic expression of the swept solid, we determine a subdivision of the space the swept volume lives in and use linear sweeps between discrete time steps to approximate the target surface. With the help of an octree based data structure, a polygonization of this surface can be determined in a very efficient manner. Dynamically updating the octree in each time step, we incrementally construct the whole piecewise linear approximation of the swept volume.

§2. Functional Representation of Solids

A solid object S can be represented as a closed subset of $\mathbb{R}^3$ with a defining function f. Thereby f is a real continuous implicit function of point coordinates, particularly a generalization of a distance field, with

$$f(\mathbf{p}) \begin{cases} > 0 & \text{for points } \mathbf{p} \in \mathbb{R}^3 \text{ inside the solid,} \\ = 0 & \text{for points } \mathbf{p} \in \mathbb{R}^3 \text{ on the solids boundary,} \\ < 0 & \text{for points } \mathbf{p} \in \mathbb{R}^3 \text{ outside the solid.} \end{cases} \tag{1}$$

Defining functions of complex solids can be created from a finite set of solid primitives or given defining functions with set-theoretic operations by a CSG-like scheme. With the help of the theory of R–functions the exact analytical definitions of set theoretic operations can be expressed (cf. [20,23]). For example, if solid S_1 is defined as $f_1 \geq 0$ and solid S_2 as $f_2 \geq 0$, then

$$\begin{array}{llllll} \text{Intersection} & : & S_3 = S_1 \cap S_2 & : & f_3 = f_1 \wedge f_2 = f_1 + f_2 + \sqrt{f_1^2 + f_2^2} \\ \text{Union} & : & S_3 = S_1 \cup S_2 & : & f_3 = f_1 \vee f_2 = f_1 + f_2 - \sqrt{f_1^2 + f_2^2} \\ \text{Complement} & : & S_3 = S_1 & : & f_3 = -f_1 \\ \text{Subtraction} & : & S_3 = S_1 \backslash S_2 & : & f_3 = f_1 \backslash f_2 = f_1 - f_2 - \sqrt{f_1^2 + f_2^2} \end{array} \tag{2}$$

These R-functions have C^1 discontinuities only in points where both arguments equal zero, i.e. at intersections of the belonging solid's surfaces.

Using the concept of defining functions of solids, one can make these solids move by introducing a fourth variable t representing the time. Thus, the general defining function of a moving solid is

$$f(x, y, z, t) \geq 0, \tag{3}$$

where $f(x, y, z, t_1)$ and $f(x, y, z, t_2)$ only differ by a rigid transformation.

Since the elimination of the parameter t in (3) is quite difficult in general, one uses a piecewise constant approximation of the moving solid given by

$$S := S(t_1) \cup S(t_1 + dt) \cup S(t_1 + 2dt) \cup \ldots \cup S(t_2), \tag{4}$$

where $\cup$ is the set-theoretic union defined in (2). For $dt \to 0$ we get a more and more accurate approximation of the swept solid.

In order to achieve a piecewise linear approximation instead of the piecewise constant given by (4), one can define the sweep as union of two solid samples in initial and final positions and an envelope swept by the moving solid, i.e. the swept object can be described by

$$f(x, y, z) = f(x, y, z, t_1) \vee f(x, y, z, t_2) \vee E(x, y, z), \tag{5}$$

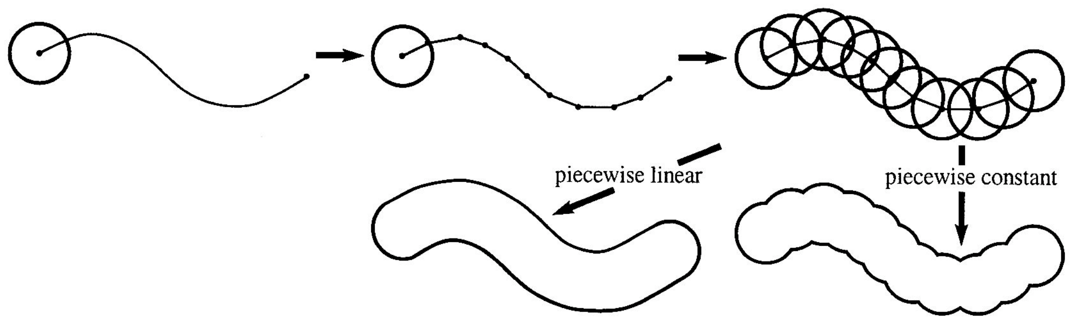

Fig. 1. Piecewise constant or linear approximation respectively.

where E is the envelope, i.e., the object swept during the motion of the generator (cf. Fig. 1).

The envelope surface is the surface which is tangential to all members of the family of surfaces. Consider the two members of the family at times t and $t + dt$. The first one has equation $f(x, y, z, t) = 0$, while the second one has equation $f(x, y, z, t + dt) = 0$. Due to the fact that the envelope must meet both of these surfaces, and thus satisfy both equations, it can be seen by expanding the latter in terms of dt, and letting dt tend to 0, that the envelope surface must simultaneously satisfy

$$f(x, y, z, t) = 0 \quad \text{and} \quad \frac{\partial f(x, y, z, t)}{\partial t} = 0.$$

The implicit form of the envelope surface can be obtained by eliminating t from these two equations. If the surfaces are described by polynomials, the elimination can be performed automatically using methods of *Computer Algebra* such as resultants or Gröbner bases (cf. [9]). Although these methods are restricted to rational functions, many other forms can be converted to them. As an example, we can represent $\sin(s)$ and $\cos(s)$ as the rational functions $2t/(1 + t^2)$ and $(1 - t^2)(1 + t^2)$, with help of the substitution $\tan(s/2) = t$.

All of the proposed methods are computationally expensive, needing exponential or even doubly exponential time in the number of time-steps t_i. Moreover, the resulting implicit functions in general have very high degree which implies hundreds of coefficients.

Moving a rigid solid along the trajectory $\mathbf{g}(\mathbf{t}) : [t_1, t_2] \longrightarrow \mathbb{R}^3$ with the orientation $\mathbf{p}(t) : [t_1, t_2] \longrightarrow \mathbb{R}^{3\times3}$ the structure of the defining function f is

$$f(x, y, z, t) = s(\mathbf{p}(t)(x, y, z)^T + \mathbf{g}(t))$$

In our actual implementation we only consider spheres, thus we can assume $\mathbf{p}(t)$ to be the identity. The error we make by approximating the trajectory $\mathbf{g}(t)$ by the polygon $\mathbf{l}(t)$ being the piecewise linear interpolation of the points $\mathbf{g}(t_1), \mathbf{g}(t_1 + dt), \ldots, \mathbf{g}(t_2)$ can be estimated by

$$\max_{u\in[t_1,t_2]} \|\mathbf{g}(t) - \mathbf{l}(t)\|_2 \leq \frac{1}{8}\, l^2_{\max} \cdot \max_{u\in[t_1,t_2]} \|\mathbf{g}''(t)\|_2, \tag{6}$$

where $l_{\max} = \max \|\mathbf{g}(t_1 + idt) - \mathbf{g}(t_1 + (i+1)dt)\|_2$ is the length of the longest polygon edge. If the moving solid is a sphere, than the right hand side of (6) is also an upper bound for the error made by approximating the moving sphere's envelope by a collection of cylinders. This is obvious if we take into account the fact that the tangent to the trajectory $\mathbf{g}(t)$ is parallel to $\mathbf{l}(t)$ at the parameter value $t_{\max}$ where the maximum error occurs. Thus, the envelope constructed by our approach converges quadratically to the desired envelope. Calculating only a piecewise constant approximation instead of a linear one, we increase the error (6) by

$$E(r) = r - \frac{\sqrt{4r^2 - l_{\max}^2}}{2}, \tag{7}$$

where r is the radius of the sphere, resulting in linear instead of quadratic convergence.

§3. Polygonization

One simple way to visualize a solid is to raytrace it directly from its defining implicit function. This only visualizes the object from one specific rendering viewpoint, and needs very high computational resources as we have still mentioned.

For applications beyond mere visualization, it is useful to approximate the surface with an *explicit representation* (indirect visualization), such as a set of triangles or polygons. In fact, any continuous manifold may be approximated by a triangulation (cf. [25]). This process, called polygonization in general involves sampling the surface. One standard approach for implicitly defined surfaces is to compute the intersection of a three-dimensional grid with the surface in order to determine the location and connectivity of surface points.

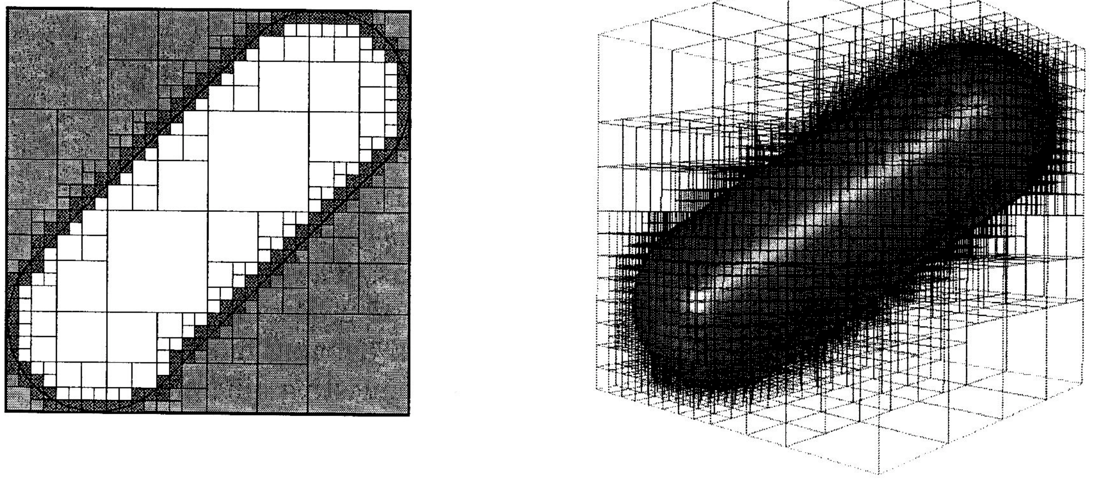

Fig. 2. Quadtree and octree-representation of an implicit function respectively.

Using the divide-and conquer strategy of binary *subdivision* is one of the most efficient ways to generate this three-dimensional grid. It leads to quadtree and octree data-structures, respectively: A quadtree (octree) is derived by successively subdividing a plane (space) in both (three) dimensions to form quadrants (octants). An comprehensive survey of quadtrees, octrees and other forms of subdivision such as KD-trees and bintrees can be found in [21,22]. When an octree is used to represent an implicit function, each cell may be inside, outside, or partially inside and outside (also called white, gray and black respectively) the solid described by the implicit function (cf. Fig. 2).

When building up the octree, we only have to subdivide gray cells (adaptive refinement). To test if a given cell in the octree has to be subdivided, we check the sign of the implicit function at the corners. Different signs at both ends of an edge indicate that the surface must have an intersection with that edge.

Unfortunately, there are configurations where parts of the surface lie in the interior of the cell, yet the implicit function has the same sign at all corners. Such situations are quite hard to detect in general, since many special cases have to be checked [7]. However, in the case of adaptive refinement for isosurface extraction, it is sufficient to find a *conservative* splitting criterion. For correct reconstruction, we have to guarantee that a cell is subdivided if some part of the surface lies in the interior, but we can tolerate wrong decisions where the cell if subdivided, even if the surface does not intersect the cell.

In the case of a moving solid, we can derive such a conservative criterion by computing a bounding box or a bounding sphere that encloses the solid. For such basic primitives, the intersection test with a given cell can be implemented very efficiently (cf. [1]). In oder to reduce the number of erroneous decisions, we can additionally compute an empty box or empty sphere in the interior of the moving solid. For the practically relevant geometries, i.e., the standard shapes of milling tools, the approximation of the moving solid by an outer bounding box and an inner empty box is sufficiently tight.

Another criterion to subdivide given octants in order to achieve an approximation of an object is the variation of the target's distance field over the parent cell (cf. [10]).

Once the octree representation of the solid is given, one can construct a polygonal approximation of the solid's surface by examining the different configurations of the corners of the partially full voxels (border voxels). Therefore, we use the distance values $f(\mathbf{p})$ evaluated at the voxel corners and stored in our hash table. Each of the 8 voxel corners can have a negative or nonnegative value, resulting in 256 different corner configurations. These cases can be handled by a table method (*Marching Cubes*) proposed in [14,18], or an algorithmic method described in [3]. Because our implicit defining functions are continuous, they always intersect an edge of a voxel connecting differently signed vertices. A coarse approximation of the surface samples are the midpoints of these edges. In order to get a better approximation of the desired object, we either approximate the exact intersection using bisection or Newton iteration, or linear interpolate with respect to the values assigned to the

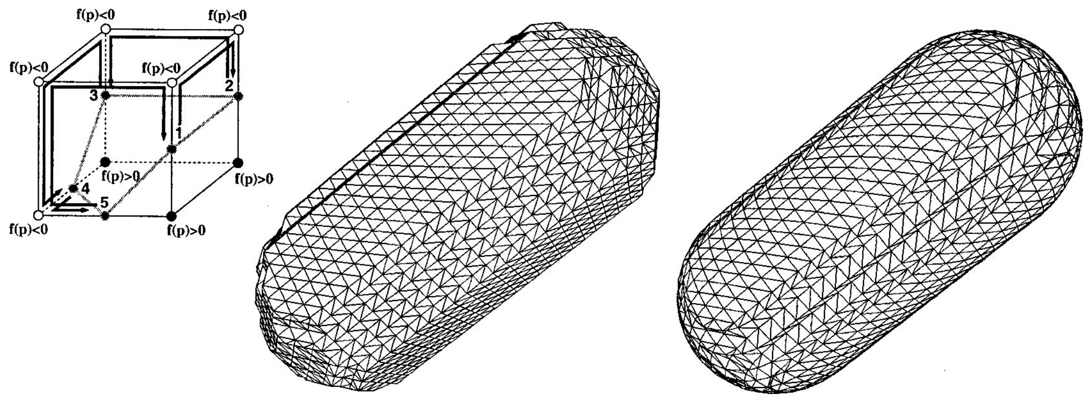

Fig. 3. Left: The algorithmic polygonization method: The surface vertices are ordered by walking from one surface vertex to the next, around a face of a given voxel in clockwise order. When arriving at a new vertex, the face across the vertex's edge from the current face becomes the new current face. Middle and right: Surface obtained by connecting midpoints or linear interpolation respectively.

corners of the voxels. Figure 3 illustrates the algorithmic approach, and shows two resulting triangular meshes, with and without linear interpolation.

§4. Dynamic Octree Manipulation

In this section we describe our new approach to approximately reconstruct the envelope of a moving solid. In particular our goal is to construct a polygonal approximation of the target surface given by (4) and (5).

We denote the volume given by the linear interpolation of $S(t_1 + idt)$ and $S(t_1 + (i+1)dt)$ by L_i. Because it is computationally too expensive to determine the implicit function given by (4) and (5), we just consider the volume L_{i+1} with respect to that given by $L_0 \cup \ldots \cup L_i$. In this incremental reconstruction process we choose a bounding box (= root cell of the octree) that is large enough to contain the whole swept volume given by (4), and start computing the octree approximating L_0.

In order to construct the octree approximating that part of the envelope that is generated by $L_0 \cup L_1$, we just traverse the initial octree of L_0 and check in what manner it is changed by L_1. Thereby, we do not have to worry about branches that are marked inside because they can never change status. We only have to update leaves that are marked as border or outside leaves. This makes our approach quite fast.

If a branch of the octree is marked as outside, we have to construct a new suboctree corresponding to L_1. If a branch is marked as border, we have to check whether this branch is inside L_1 or not. If it is not, we go on with the traversal of the initial octree with respect to L_1. Thereby all suboctrees of the initial octree now being inside L_1 can be removed to reduce computation time and memory (cf. Fig. 4).

In order to get the correct results, we also have to update the values of the defining functions evaluated at the voxel corners. Due to the fact that a

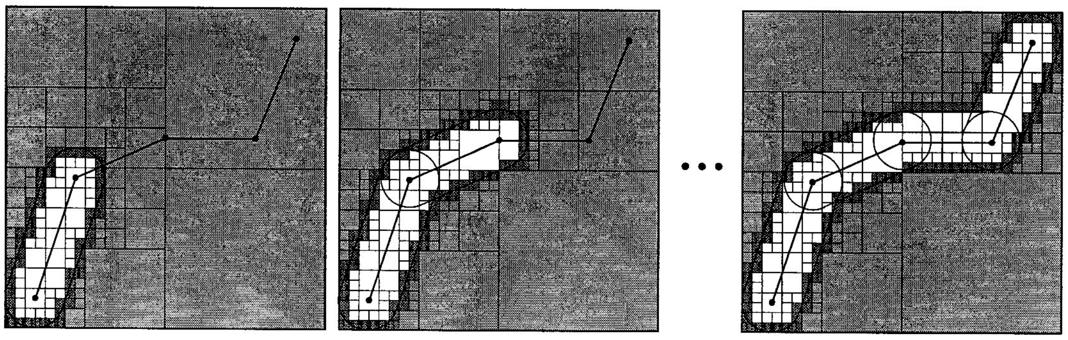

Fig. 4. Dynamically updating a quadtree in order to approximate the envelope of a circle moving along a polygon.

region once marked as inside alway stays inside, we have to compute the new values and store the higher one of an old and a new value in our hash table.

If we repeat this procedure until we reach L_{t_2-dt}, we have built an octree representing the whole moving solid given by (4) and (5) up to an error tolerance determined by the size of the voxels or the maximum depth of the octree respectively. Thereby the time step dt has to be chosen sufficiently small, so that the error made by our discretization (4) is smaller than the size of the voxels. Figure 4 illustrates the dynamically growing octree, quadtree respectively. Notice that voxels of regions that have been subdivided to the maximum refinement level can be combined and deleted during this process.

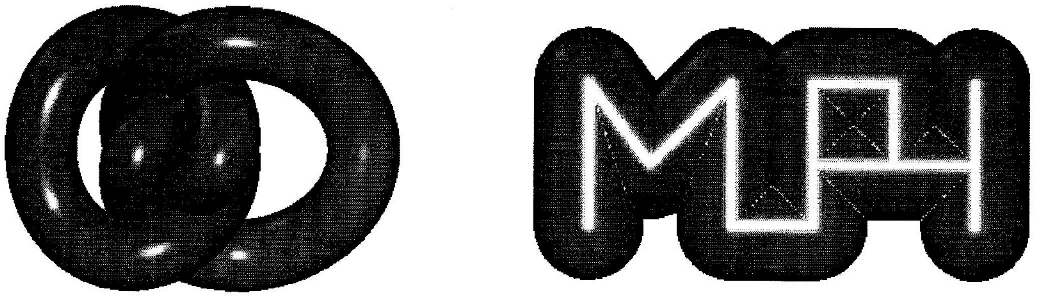

Fig. 5. Some example sweeps. Notice the alias effect along intersections.

Figure 5 shows some resulting sweeps. On the left hand side a sphere of radius $r = 2$ is moving along the knot curve defined by

$$k(t) = \begin{pmatrix} 10\cos(t) + \cos(3t) + \cos(2t) + \cos(4t) \\ 6\sin(t) + 10\sin(3t) \\ 4\sin(3t)\sin(\frac{5}{2}t) + 4\sin(4t) - 2\sin(6t) \end{pmatrix}, \quad t \in [0, 2\pi].$$

With $dt = \frac{2\pi}{1000}$ and edge-length of the smallest voxels equals 0.02 (=error tolerance). The resulting polygonal approximation consists of approximately 376000 triangles, and it took about 50*sec* to compute it. On the right hand side a sphere of radius $r = 5$ is moving along 13 points building a polygon describing the letters MPI. The edge-length of the smallest voxels has also been set to 0.02 resulting in a triangular mesh with about 500000 triangles. It took about 20*sec* to calculate this mesh which is faster than the first example because fewer time steps had to be computed.

§5. Post Processing

The approach described in the previous section guarantees a prescribed error-bound. The error made by our reconstruction process is at most the length of the edges of the voxels. But due to the discretization we made, and due to the fact that we only choose one possible triangulation of the calculated points, our approach, like every discrete approach, produces alias effects appearing at sharp features (cf. Fig. 5).

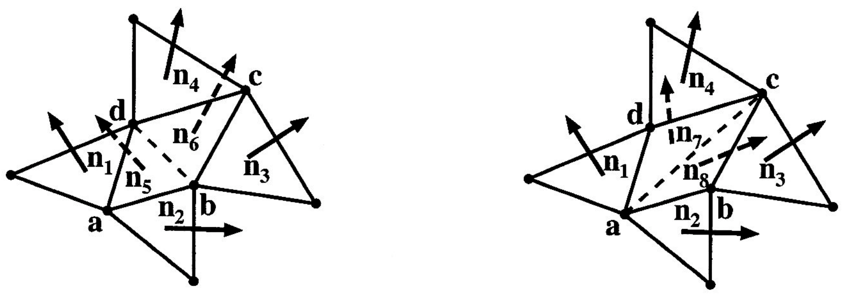

Fig. 6. Flipping an edge according to the deviation of normals.

In order to reduce these effects one can improve the triangulation by flipping edges of the mesh representing the solids envelope. Therefore, one has to define a global curvature functional as a criterion for the mesh quality. Such a functional may be the deviation of the normals of adjacent triangles or the discrete Gaussian curvature. Several methods to calculate the discrete Gaussian curvature of a polygonal net exist. A fast and numerically stable method to estimate not only the Gaussian curvature but also the tensor of curvature of a polygonal net can be found in [24]. A comparison of the errors of several approximations to the surface normal and the Gaussian curvature can be found in [16].

In our implementation we chose a functional based on the deviation of the normals of adjacent triangles which is illustrated in Fig. 6. We examine all triangles of our net and flip an edge **bd** shared by two adjacent triangles into the edge **ac** if $\min(A\setminus\min(A)) > \min(B\setminus\min(B))$ with $A = \{\mathbf{n}_1^T\mathbf{n}_5, \mathbf{n}_4^T\mathbf{n}_5, \mathbf{n}_3^T\mathbf{n}_6, \mathbf{n}_2^T\mathbf{n}_6\}$ and $B = \{\mathbf{n}_1^T\mathbf{n}_7, \mathbf{n}_4^T\mathbf{n}_7, \mathbf{n}_3^T\mathbf{n}_8, \mathbf{n}_2^T\mathbf{n}_8\}$.

Figure 7 shows a surface before and after processing the edge-flipping step. It can be seen that the quality of the mesh has been improved drastically. Unfortunately, as one can also see, the described edge–flipping process does not remove all of the occurring alias effects but at least most of them. This is due to the fact that our greedy approach can become stuck at local minima of the functional measuring the mesh quality. To overcome this problem, one has to choose more sophisticated techniques, e.g., considering the mesh quality

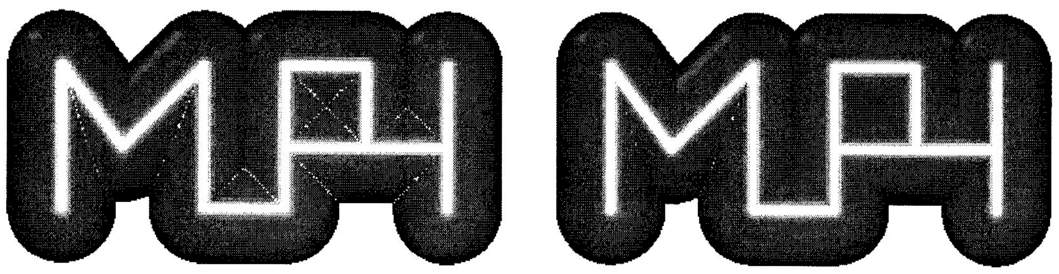
Fig. 7. A Swept surface before and after flipping the edges.

not only after performing one edge-flip, but two ore even more, which will be future work.

In Figure 8 an example from an industrial application is shown. The pictured surface resulted from moving a ball cutter of diameter $r = 8mm$ along a path given by 3770 points. The edge-length of the voxels approximating the surface is $0.1mm$. The resulting surface consists of about 1.2 million triangles, and it took about 10 minutes to calculate.

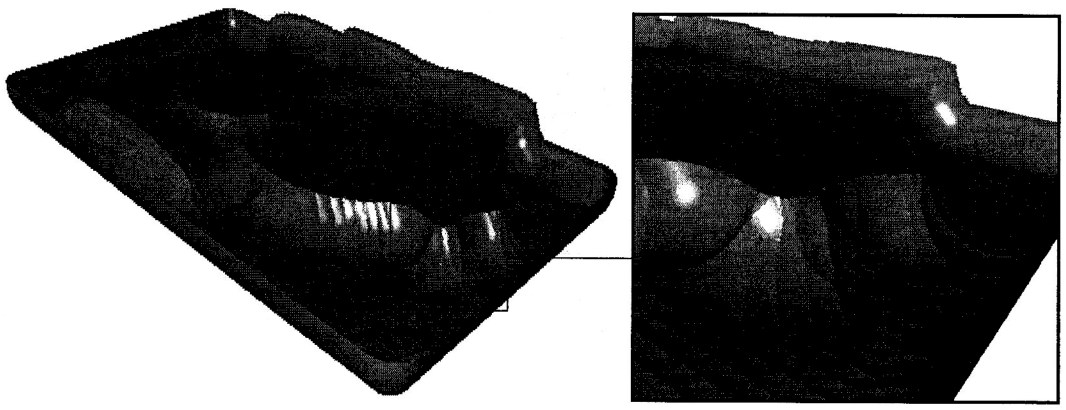
Fig. 8. A NC machining simulation example.

§6. Conclusion and Future Work

We presented a technique to construct a polygonal approximation of the envelope of a moving solid which is based on dynamically updating an octree approximating the envelope surface. Since we use linear sweeps between discrete time steps, our approach is very fast. It can reconstruct the envelope of a moving solid up to a prescribed error bound, while it is scalable in the sense that tolerating a larger approximation error speed up the calculation so that rough approximations even could be obtained in real time. Because no surface-surface intersections have to be computed explicitly, the algorithm is robust and highly eligible for applications like NC milling simulation, where typically a lot of these intersections turn up.

Finally, we showed that flipping the edges of a mesh can improve the mesh-quality drastically. Up to now we only have implemented a simple greedy edge-flipping procedure. Using a more sophisticated optimization technique could improve the mesh-quality even more.

Our actual implementation only allows moving spheres, because there exist very fast and simple algorithms to check whether a sphere or ellipsoid intersects a given box or not. Our algorithm could be generalized to arbitrary solids, as long as we can evaluate the distance function in an efficient manner (e.g. for polygonal meshes cf.[12]). However, already the envelope swept by an arbitrary solid moving along a straight line can become quite complex which has been shown in Section 2. At this point, the piecewise constant approximation might lead to the more efficient reconstruction algorithm.

Acknowledgments. This research was partly supported by Siemens AG München.

References

1. Arvo, J., A simple method for box-sphere intersection testing, in *Graphics Gems*, A. S. Glassner (ed.), Academic Press, New York, 1990, 335–339.
2. Blackmore, D., R. Samulyak and M. C. Leu, Trimming swept volumes, Computer-Aided Design **31** (1999), 215-223.
3. Bloomenthal, J., Polygonization of implicit surfaces, Computer Aided Geom. Design **5** (1988), 341–355.
4. Bloomenthal, J., C. Bajaj, J. Blinn, M. Cani-Gascuel, B. Wyvill and G. Wyvill, *Introduction to Implicit Surfaces*, Morgan–Kaufmann, San Francisco, 1997.
5. Bronsvoort, W., A surface scanning algorithm for displaying generalized cylinders, The Visual Computer **8** (3) (1992), 162–170.
6. Bronsvoort, W. and F. Klok, Ray Tracing generalized cylinders, ACM Trans. on Graphics **4** (4) (1985), 291–303.
7. Brunet, P. and I. Navazo, Solid representation and operation using extended octrees, ACM Trans. on Graphics **9** (2) (1990), 170–197.
8. Carmo do, M., *Differential Geometry of Curves and Surfaces*, Prentice Hall, 1976.
9. Davenport, J. H., Y. Siret and F. Tournier, *Computer Algebra: Systems and Algorithms for Algebraic Computation*, Academic Press, 1993.
10. Frisken, S. F., R. N. Perry, A. P. Rockwood and T. R. Jones, Adaptively sampled distance fields: a general representation of shape for computer graphics, Proc. Siggraph '00, ACM, 2000.
11. Gouping, W., S. Jiaguang and H. Xuanji, The sweep-envelope differential equation algorithm for general deformed swept volumes, Comput. Aided Geom. Design **17** (2000), 399-418.
12. Guéziec, A., "Meshsweeper": Fast closest point on a polygonal mesh and applications, IEEE Trans. on Visualization and Computer Graphics, to appear.

13. HU, Z. and Z. Ling, Swept volumes generated by the natural quadric surfaces, Computer and Graphics **20** (2) (1996), 263–274.
14. Lorensen, W. E. and H. E. Cline, Marching cubes: a high resolution 3D surface construction algorithm, Computer Graphics **21** (3) (1987), 163–169.
15. Liu, C., D. M. Esterling, J. Fontdecaba and E. Mosel, Dimensional verification of NC Machining profiles using extended quadtrees, Computer-Aided Design **28** (11) (1996), 845–852.
16. Meek, D. S. and D. J. Walton, On surface normal and Gaussian curvature approximations given data sampled from a smooth surface, Comput. Aided Geom. Design **17** (2000), 521–543.
17. Menon, J. P. and D. M. Robinson, Advanced NC verification via massively parallel raycasting, ASME DE Manufacturing Review **6** (1993), 141–154.
18. Montani, C., R. Scateni and R. Scopigno, A modified look-up table for implicit disambiguation of marching cubes, The Visual Computer (**10**) (1994), 353–355.
19. Oliva, J-M., M. Perrin and S. Coquillart, 3D Reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram, EUROGRAPHICS 96 (1996), 397–408.
20. Rvachev, V. L., *Methods of Logic Algebra in Mathematical Physics*, Naukova Dumka Publishers, Kiev, 1974.
21. Samet, H., *Applications of Spatial Data Structures*, Addison-Wes., 1990.
22. Samet, H., *Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1990.
23. Shapiro, V., Real functions for representation of rigid solid, Comput. Aided Geom. Design **11** (1994), 153–175.
24. Taubin, G., Estimating the tensor of curvature of a surface from a polyhedral approximation, Proc. ICCV '95, 1995, 902–907.
25. Whitney, H., Elementary structure of real algebraic varieties, Annals of Mathematics **66** (1957), 545–556.

Ulrich Schwanecke, Leif Kobbelt
Max-Planck-Institute for Computer Sciences
Stuhlsatzenhausenweg 85
66123 Saarbrücken, GERMANY
{schwanecke,kobbelt}@mpi-sb.mpg.de