Computing Offsets and Tool Paths With Voronoi Diagrams[1]

Jin Jacob Chou and Elaine Cohen

UUCS-89-017

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

August 29, 1990

[1]This work was supported in part by DARPA(DAAK1184K0017 and N00014-88-K-0689). All opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not reflect the views of the sponsoring agencies.

Computing Offsets and Tool Paths With Voronoi Diagrams*

Jin Jacob Chou and Elaine Cohen
Computer Science
University of Utah
Technical Report Number: UUCS 89-017

Abstract

In this paper we describe the use of Voronoi diagrams to generate offsets for planar regions bounded by circular arcs and line segments, and then use the generated offsets as tool paths for NC machining. Two methods are presented, each producing a different type of offset. One of them generates the offsets of the region; the other divides the region into *subpockets* first, then offsets of the subpockets boundaries are used for tool paths. We show that a set of m offsets can be computed in $O(c_1 n \log n + c_2 mn)$ time , where n is the number of sides of the region, using either method.

1 Introduction

Pocketing in milling can be defined as the removal of the material within a pocket, a planar region whose boundary definition is permitted to have both arcs and line segments. The number of the arcs and line segments is denoted as n. The region can be multiply connected, i.e., loops are allowed within the outermost boundary. The loops are called islands. The capability of automatic tool path generation for pocketing is important, since it is not only a frequently occurring operation for producing two-dimensional shapes, but also a primitive step used repetitively in the roughcutting of three-dimensional parts [1,2].

Traditionally, raster-cuts and spiral-cuts are used in pocketing [3,4,5]. In the raster-cut approach, a zig-zag tool path along a family of parallel lines is used. In the spiral-cut approach, a sequence of offsets to the pocket boundary is taken as the tool path. The tool is directed to shrink the uncut portions of the pocket in a step-wise fashion starting from the boundary or, in a reverse order, to expand the cut portion of the pocket starting from the innermost points. Essentially, the computation of an offset is required for every move toward or away from the boundary. If shrinking strategy is taken, in pockets with concave boundaries, the uncleaned area may break into several disconnected regions after the tool moves through a number of offsets. This implies that a single offset curve for the tool path may contain disjoint loops. On

*This work was supported in part by DARPA (DAAK1184K0017 and N00014-88-K-0689). All opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not reflect the views of the sponsoring agencies.

the other hand, disconnected regions may need to be merged at some points, if an expanding strategy is taken. This implies that disjoint loops in a single offset may merge together in the subsequent expanding offsets.

It is indicated in [1] that each offset requires $O(n^2)$ time to compute if the method which computes the intersection of consecutive boundary elements is used. In this paper, spiral-cuts are used in pocketing. It will be shown that, with Voronoi diagrams, a sequence of offsets to a pocket can be found with $O(n \log n)$ preprocessing time and $O(n)$ time for each offset, a complexity which is independent of the shape of the pocket and the existence of islands.

2 Voronoi Diagrams

Practical applications of Voronoi diagrams arise in diverse fields [6,7]. In particular, Voronoi diagrams have been used extensively in motion planning in robotics [8,9]. The *Voronoi diagram* for a set of elements can be depicted as the partition of a domain into regions, each of which contains points closer to one element than the others [6]. The regions so partitioned are called Voronoi polygons. The element to which a Voronoi polygon is closest is called the *owner element* of the Voronoi polygon. In our case, the set of elements is the boundary of the pocket: line segments, circular arcs, and end points of the line segments and circular arcs.

For a point p in a Voronoi polygon R, the distance between p and the owner element of R is called the *clearance of p*. The *graph of the Voronoi diagram*, or simply called the *Voronoi diagram*, is the set of edges and vertices of the Voronoi polygons. Every point on an edge is at a minimum distance to two elements. The minimum distance is the *clearance of the point*, and the two elements are called the *forming element of the edge*. If all points on an edge have the same clearance, the edge is called a *constant clearance* edge. The *forming elements of a vertex* are the forming elements of the edges incident to the vertex. For a set of circular arcs, line segments, and points, the edges are portions of conics [10]. An important property of the Voronoi diagram is that the numbers of edges, vertices, and Voronoi polygons are linearly proportional to the number of elements that define the Voronoi diagram.

When restricted to the interior of a polygon, the Voronoi diagram is known as the *internal skeleton*, or simply the *skeleton*, of the polygon [11]. For a simple polygon, the internal skeleton is also called the *medial axis*. The problem of constructing the skeleton of a polygon is linearly reducible to that of constructing the Voronoi diagram of the polygon [12]. The construction of the Voronoi diagram has been studied by a number of researchers [6,9,10,12,13]. It is possible to construct the Voronoi diagram in $O(n \log n)$ time. In the following we assume the skeleton of the pocket is already obtained. This can be done in $O(n \log n)$ time as mentioned above. Figure 1 shows the skeleton of a pocket with an island.

For simplicity and clarity in describing the algorithms, we restrict ourselves to pockets whose skeletons are without constant clearance edges. Later we show why this is necessary and will suggest a way to handle them in cases that they do exist. Note that a constant clearance edge occurs when the forming elements of the edge are parallel line segments or concentric arcs.

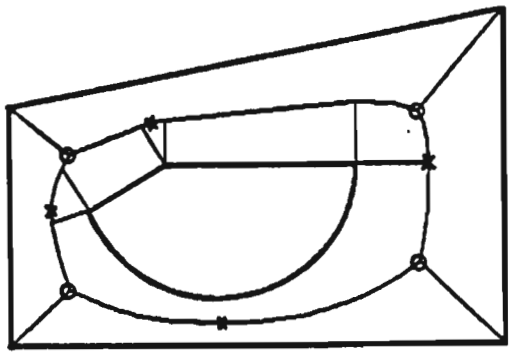

Figure 1: Skeleton of a pocket.

3 Using the Skeleton to Find Offsets

3.1 Properties and Definitions

The basic idea of the algorithm is that it is easy to find the set of points in a Voronoi polygon that are at a constant distance from the owner element of the Voronoi polygon. If such a set is denoted as W, the offset of a pocket consists of all the Ws from all the Voronoi polygons of the pocket. Note that for a given offset distance, a Voronoi polygon may not contain points with that distance away from the owner element. In this case the set W is empty, and the algorithm really does not need to examine such a Voronoi polygon. Also, the offset produced by the algorithm should consist of closed curves. To achieve these, a special walk through the Voronoi polygons is used by the algorithm. In order to facilitate the description of the algorithm, some additional terminologies and properties are introduced below.

Definition 3.1

1. *A* mountain point x *is a point in a skeleton with the maximum clearance among those points in the neighborhood of* x.
2. *A* valley point x *is a point in the skeleton with the minimum clearance among those points in the skeleton in the neighborhood of* x, *when* x *is not a point on the boundary of the pocket.*

Since edges of constant clearance are excluded, mountain and valley points in a skeleton are disjoint. Mountain points represent points in the pocket that have local maximum clearance. However, since all the points on a constant clearance edge may have the same maximum (or minimum) clearance, the above definitions for mountain and valley points are not suitable for our purposes when constant clearance edges are present. In our discussion, we assume that there are no constant clearance edges in the skeleton, although special care could be taken to incorporate those edges in our discussion. For example, when all points on a constant edge have the same maximum clearance, a middle point on the edge could be taken as the mountain point, and all the subsequent discussion would have to take this special case into consideration.

Property 3.2 *A mountain/valley point of a skeleton is either at a vertex of the skeleton, or at the apex of the conic of an edge. A valley point also can occur on a linear edge with two points or circular arcs as forming elements.*

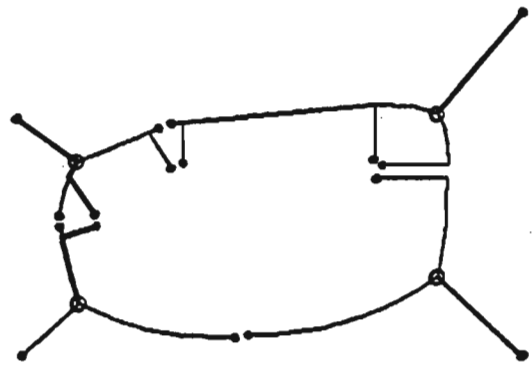

Figure 2: Trees for the skeleton in figure 1.

By property 3.2, all the mountain and valley points in a skeleton can be obtained in linear time since we only have to examine each of the edges and the vertices in the skeleton once. In Figure 1, the mountain points are circled, and the valley points are crossed. We modify the skeleton by adding a vertex into the skeleton at each of the mountain (valley) points that do not coincide with a vertex. A vertex on the skeleton which is a mountain (valley) point is called an *m-vertex* (*v-vertex*). Next, we attach the clearance value to each of the vertices in the skeleton. By adding the m-vertices and v-vertices, we have the following.

Property 3.3 *All the edges in the skeleton have monotonic clearance.*

Property 3.4 *The skeleton is a planar embedding of a forest with vertices and edges of the skeleton as the nodes of the forest. The roots of the trees in the forest are the m-vertices. The leaves are either v-vertices or vertices at the boundary of the pocket.*

Property 3.5 *The clearance of the points on a branch of a tree in the skeleton is in a decreasing order from the root to the leaf.*

Figure 2 shows the trees for the skeleton in Figure 1. The roots are circled. The leaves are dotted. Note that two branches (vertices) of the trees may correspond to one edge (vertex) in the skeleton.

To initiate computing offsets, we search for the starting point of an offset from a list of special vertices from the skeleton, called *control vertices*. Associated with each of the control vertices is a branch of the trees from the skeleton, called a *key branch*. The list of control vertices and the key branches are defined as:

1. All the mountain vertices are the control vertices. The key branch associated with a mountain vertex is the branch of the tree from the mountain vertex to the leave of the tree which has the minimum clearance. If more than one leaf of the tree has the same minimum clearance, an arbitrary one is chosen.

2. For each of the boundaries of the islands and the outmost boundary of the pocket, choose an arbitrary vertex on the boundary. Start from this vertex, and walk up the tree containing the vertex until either a vertex in a found key branch or the root of the tree is

reached. The vertex at which we stop is defined to be a control vertex and is added into the list. The edges and vertices along the path constitute the key branch of that control vertex.

In Figure 2, the heavy lines indicate a set of key branches. Note that a vertex may appear more than once in the list.

Given an edge e of the skeleton, we will need to distinguish between the two Voronoi polygons which share the edge. By property 3.3, one of the end vertices of e will have larger clearance than the other. We define the polygon Vinc(e) as the Voronoi polygon which is to the right of e, when e is traversed from the end vertex with the smaller clearance to the one with the larger clearance.

If a value d is within the range delimited by the clearance of the end vertices of an edge, we say the *edge spans over d*. A *control vertex spans over a value d*, if d falls between the maximum and minimum clearance of the points on the key branch of the vertex.

3.2 The Offset Algorithm

Now, we are ready to describe the offset algorithm. Since the offsets will be used for toolpaths, we impose additional constraints. One requirement of an offset for tool paths is that the offset should consist of closed curves. Also, to minimize the number of tool retractions, the set of offsets that centers around an m-vertex should be grouped together. We will incorporate these requirements into the offset algorithm.

Algorithm 3.6 (Offset)

1. *Generate the list of control vertices described previously, and denote it as as the m-list.*
2. *Sort the m-list by the clearance of the vertices into a nonincreasing order (largest clearance first).*
3. *For a given offset distance d, create a new list, the a-list, which contains vertices from the m-list spanning over d. The vertices in the a-list should have the same order as they had in the m-list.*
4. *For each vertex v in the a-list, according to its order, until the a-list is empty:*
   (a) *Search along the key branch of v for a starting edge which spans over d. Call it* e_s.
   (b) *Set the current edge* e_c *to be* e_s.
   (c) *If* e_c *is on a key branch, k, remove the control vertex of k from the a-list.*
   (d) *Traverse clockwise through the edges of Voronoi polygon Vinc(*e_c*) starting from the edge following* e_c *until an edge that spans over d is found.*
   (e) *Output the segment connecting* e_c *and the new edge.*
   (f) *If the new edge is not* e_s, *denote the new edge as* e_c *and return to 4c.*
   (g) *If the new edge is* e_s, *we have completed a loop in the offset and return to 4;*

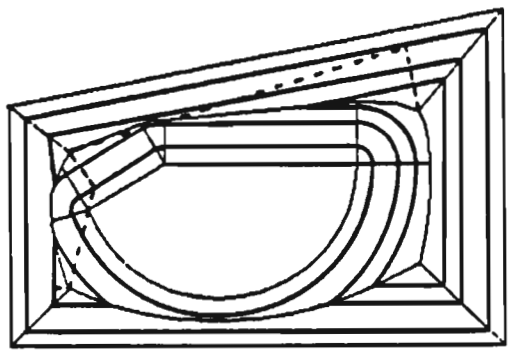

Figure 3: Offsets and toolpaths for figure 1.

Step 4c can be done efficiently if we tag the edges in a key branch with pointers to their corresponding control vertex. The output segment in 4e is a line segment if the owner element of $\mathrm{Vinc}(e_c)$ is a line segment and an arc if the owner element is a point or an arc. Potentially, the number of control vertices is proportional to n, hence step 2 requires $O(n \log n)$ time. The rest of the steps can be done in linear time. For a sequence of offsets steps 1 and 2 are executed only once and can be considered as preprocessing. Now the main result of this section can be stated:

> **Theorem 3.7 (Offset)** : *A sequence of m offsets of a pocket with n sides can be found in $c_1 n \log n + c_2 mn$ time: $O(n \log n)$ for preprocessing, and $O(n)$ time for each offset.*

In Figure 3, a set of offsets is shown in heavy solid lines.

4 Tool Path Generation With Offsets

It is necessary to know offset distances in order to generate offsets for a toolpath. Offsets with distances less than the tool radius are not allowed since the center of the tool will follow the offset and we want to generate noninvasive toolpaths. The largest offset distance needed is dependent on the maximum clearance of the pocket. Since the point with the maximum clearance in a pocket is an m-vertex, the maximum clearance is the clearance of the vertex at the head of the *m-list* in step 2 of the offset algorithm and can be found easily. The distance between two consecutive offsets cannot exceed the tool radius. Otherwise, at sharp corners, regions between the offsets may be left uncut. Other than this constraint, the distance is quite arbitrary and should be chosen to gain efficiency and to meet the mechanical constraints of the tool, the machine and the stock material.

The simple tool path generation algorithm that generates tool paths to move through the loops in an offset one by one may require a large number of tool retractions, since an offset of a pocket with h different m-vertices may have h loops. Therefore, $h-1$ retractions will be needed, in general, to move the tool between these loops in an offset. For a set of m offsets, $O(mh)$ retractions are needed. To reduce the number of retractions, we need to group the loops in the offsets together so that we can move through the sequence of loops in the same group without

retractions. This can be done by carefully arranging the output from the offset algorithm. In the following tool path generation algorithm we achieve the grouping by associating a loop list L with each control vertex.

Algorithm 4.1 (Tool Path Generation I)

1. *Follow steps 1 and 2 in Algorithm 3.6 and obtain the maximum clearance of the pocket.*
2. *Create an ordered list of nondecreasing offset distances, the d-list.*
3. *For each distance d in the d-list perform steps 3 and 4 of Algorithm 3.6, using the following modified version of step 4g:*
   (4g) If the new edge is e_s, we have completed a loop in the offset. Add this loop to the end of the loop list of v *and return to 4;*
4. *For each control vertex v with a nonempty loop list L,*
   (a) *Retract the tool and move to the m-vertex, $m(v)$, at the root of the tree which contains v.*
   (b) *For each loop l in L starting from the loop with the largest d,*
      i. *Move the tool along the side-step direction, discussed below, until it encounters l.*
      ii. *Move the tool through l.*

When we "side-step" the tool in step 4(b)i, we move the tool toward a forming element e of the m-vertex, $m(v)$. The side-step direction is chosen to be the direction from $m(v)$ to the projection of $m(v)$ onto e. The maximum number of retractions needed in the algorithm is the number of control vertices. Figure 3 shows a tool path. The dash lines are the path connecting offsets.

The maximum scallop size from a tool path generated by this algorithm can be calculated automatically. This allows us to select a tool with maximum size to satisfy a given tolerance automatically. This automatic selection is important when there are narrow passages between regions in a pocket. Sometimes a border with a certain distance q from the boundary of the pocket is required in NC machining. This can be done easily with the above algorithm, by setting the last offset value in the *d-list* to $q + r$, for a tool with radius r.

5 Subpocket Decomposition

In this section we discuss a method to decompose a pocket into simpler shapes whose offsets are easier to compute. In [4], Ferstenberg et al. tried to generate tool paths for a pocket with linear boundary by first decomposing the pocket into convex polygons. Then they generated tool paths for the convex polygons. The motivation behind their decomposition is that the topology of the tool path for a convex polygon is much simpler than that for a nonconvex one. There are two drawbacks with the approach. First, algorithms for good convex decompositions are both complex and computationally expensive. Second, a large number of convex regions will be required in the decomposition of complex pockets.

The term "subpocket" is used for the regions resulting from our decomposition method. The definition of a subpocket is given below.

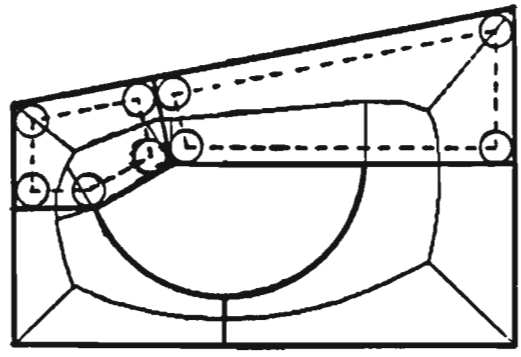

Figure 4: Subpockets for the same original pocket.

Definition 5.1 *A subpocket is a simply-connected two-dimensional region that contains only one point or one set of connected points with the maximum clearance.*

The boundary of the subpocket defined above can contain line segments and circular arcs. Note that a convex polygon is a subpocket. Also there may be a lot of components in a pocket's convex decomposition but only a few in its subpocket decomposition. We now describe the decomposition method

To develop the decomposition, we first augment the skeleton. This is done by performing the following steps for the forming elements of each v-vertex, v, in the skeleton.

1. If the projection from v to the forming element coincides with an existing vertex, add an edge between the vertex and v if such an edge does not exist.

2. If the projection does not coincide with an vertex, then a new vertex is created at the projection, and an edge is added between the new vertex and v.

The edge between a v-vertex and its projection is called a *critical edge.*

At the same time we modify the definition of the trees in the skeleton. For each of the trees in the original skeleton, we attach to each v-vertex, v, in the tree the added critical edges connected to v and the vertices at the end of the critical edges. The vertices so attached are taken as the children of v in the tree. Note that a tree may have two leaves at the same vertex. Such situations can be detected and should be tagged for later reference. We also require the branches on a tree to be arranged so that in the course of a tree traversal the descendent vertices of a vertex, h, are visited in counter-clockwise order around h.

To produce the list of ordered points defining the boundary of a subpocket, we only need to perform a tree traversal through the tree whose root is the maximum clearance point of the subpocket and report the nonredundant points at the leaf nodes. The reported points define the subpocket. The redundancy can be found easily through the tags mentioned above. It is clear that each of the region obtained from the above tree traversal has only one maximum clearance, the clearance at the root of the tree. To obtain the decomposition of a pocket it is necessary only to perform a tree traversal for each of the trees in the skeleton. Upon each traversal, a subpocket is produced. The above steps spell out the subpocket decomposition algorithm. Figure 4 shows the four subpockets (in heavy lines) from the decomposition. To

traverse through all the trees, we only need to examine all the edges and vertices in the skeleton a fixed number of times. Hence we obtain the complexity of the algorithm.

Theorem 5.2 (Subpocket Decomposition) *A subpocket decomposition of a pocket with n sides can be found in $O(n \log n)$ time.*

6 Tool Path Generation With Subpocket Decomposition

The subpocket decomposition algorithm allows us to divide a pocket into subpockets. A subpocket has the following nice properties.

Property 6.1 *An offset of a subpocket contains a single loop.*

Property 6.2 *The regions enclosed by all the offsets of a subpocket share one common point, the m-vertex in the subpocket.*

Properties 6.1 and 6.2 allow us to generate the tool path for a subpocket without retractions by starting at the m-vertex, side-stepping toward the boundary of the subpocket along a fixed direction, moving through the encountered offset loop, and repeating the process until all the offset loops are finished. The tool path thus generated is capable of covering every reachable point in the subpocket as long as the distance between two offsets is less than the tool radius But there are two drawbacks that occur when the tool paths for all subpockets are put together and used to cut the entire pocket. The first drawback is demonstrated in Figure 4. The figure shows that fillets at the corners where two subpockets meet will be left uncut. To remove these fillets requires both extra retractions and extra tool moves. The second drawback is that if a border is desirable, the tool path for the entire pocket cannot be generated by simply putting together the tool paths for each subpocket generated with border, since this will result in uncut regions between two adjacent subpockets.

We modify our tool path generation strategy slightly to overcome these drawbacks by requiring the tool to move to and along the critical edges on each offset loop when we reach the regions where two subpockets join together. Hence the loop should contain the appropriate critical edges. Given an offset distance d and a tree T, the following loop generation algorithm produces such a loop for the subpocket containing T by a traversal of T.

Algorithm 6.3 (Algorithm Loop Generation)

1. *Collect edges into a list L by performing a modified depth-first traversal of T as follows.*
   (a) *If we are at a node h whose vertex has clearance $> d$, visit the children of h.*
   (b) *If we are at a node h whose vertex has clearance $\leq d$, add the edge which joins h and its parent into L and do not traverse further down on this branch.*
2. *For each pair of consecutive edges in L, compute the point with clearance d on each of the edges and output a segment connecting these two points. The segment is either*

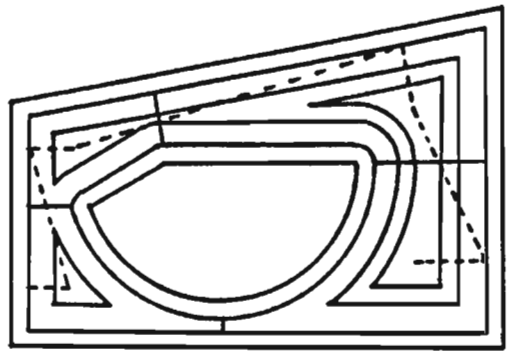

Figure 5: Loops and Tool Path.

(a) a line segment if both of the consecutive edges are critical edges or
(b) a portion of the d-offset of the common forming element of the two edges.

A segment produced in step 2b is either a circular arc, if the common forming element is a point or an arc, or a line segment, if the element is a line segment. Figure 5 shows a set of loops. Note that the loop thus constructed is neither an offset of the subpocket nor an offset of the entire pocket.

The tool path generation algorithm can be formulated easily once we can construct a loop. Given a pocket, we can generate a tool path to move through its subpockets sequentially by the following algorithm.

Algorithm 6.4 (Algorithm Tool Path Generation II)

1. *Generate the skeleton mentioned above.*
2. *Choose an ordered list of nondecreasing offset distances, the d-list.*
3. *For each m-vertex, m, in the skeleton,*
   (a) Retract the tool and move the tool to m.
   (b) For each value d in the d-list,
      i. Move the tool along the side-step direction until we are at a point with clearance value equal to d.
      ii. Generate the loop with offset distance d by the loop generation algorithm and move the tool through the loop.

The side-step direction can be selected the same as in algorithm 4.1. Algorithm 6.4 has properties and complexity analysis similar to that of Algorithm 4.1. Figure 5 shows a set of tool paths. The dash lines are the path connecting loops and subpockets.

7 Conclusion

For general free form curves, the topology of their offset curves is complex and difficult to determine. Here, with the help of Voronoi diagrams, we have demonstrated efficient ways to do

so for curves composed of line segments and arcs. The offset and the subpocket decomposition algorithms can be used not only for tool path generation but also for situations in which offsets or a subpocket decomposition is necessary. The ability of handling islands is important since they happen frequently in practical situations. All the algorithms presented handle islands uniformly.

References

1. Chou, J. J., *NC Milling Machine Tool Path Generation for Free Form Curves and Surfaces.* PhD thesis, University of Utah, 1989.

2. Gaal, B., and Varady, T., *Minicomputer based CAD/CAM system for mechanical components of free-from shapes.* In CAD '82, 1982, pp. 381 - 390.

3. Wang, K. K., and Wang, W. P., *A PADL-based numerical control machining data generation program.* In Annals of CRIP (1981), pp. 359-362.

4. Ferstenberg, R., Wang, K. K., and Muchkstadt, J., *Automatic generation of optimized 3-axis NC programs using boundary files.* In IEEE 1986 Int. Conf. on Rob. & Auto. (1986).

5. Persson, H., *NC machining of arbitrary shaped pockets.* Computer- aided Design 10, 3 (May 1978), 169 - 174.

6. Preparata, F. P., and Shamos, M. I., *Computation Geometry, An Introduction.* Springer-Verlag Inc., 1985.

7. Lee, D. T., and Preparata, F., *Computational geometry - a survey.* IEEE Trans. on Computers C-33, 12 (Dec. 1984), 112-113.

8. O'Dulaing, C., and Yap, C., *A retraction method for planning the motion of a disc.* Journal of Algorithms 6 (1985), 104-111.

9. Leven, D., and Sharir, M., *Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagram.* Journal of Discrete and Computational Geometry 1,2 (1987), 9-31.

10. Yap, C. K., *An O(nlogn) algorithm for the Voronoi diagram of a set of simple curve segments.* Journal of Discrete and Computational Geometry 1 (1987), 24-37.

11. Pfaltz, J. L., and Rosenfeld, A., *Computer representation of planar regions by their skeletons.* CACM 10, 2 (Feb. 1967), 119 -125.

12. Kirkpatrick, D. G., *Efficient computation of continuous skeletons.* In 20th Annual IEEE Symposium on Foundations of Computer Science (1979), pp. 19-27.

13. Srinivasa, V., and Nackman, L., *Voronoi diagram for multiply- connected polygonal domains I: algorithm.* IBM Journal of Research and Development 31, 3 (May 1987), 361 - 372.