Constant Scallop Height Tool Path Generation

Jin J. Chou and Elaine Cohen

UUCS-89-011

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

August 27, 1990

Constant Scallop Height Tool Path Generation*

Jin J. Chou and Elaine Cohen
Computer Science Department
University of Utah
Salt Lake City, UT 84112
Technical Report Number: UUCS-89-011

April 1989

Abstract

An approach for the automatic generation of constant scallop height tool paths is presented. An example is shown generated from a B-spline model, although it can be used with many types of sculptured surfaces. The approach utilizes surface subdivision techniques and a new algorithm for tool path generation. The new algorithm is based on computer graphics shading algorithms and on methods from graph theory. A tool path with a constant scallop height renders minimum waste tool moves and hence results in much better machine time. Since neither numerical methods nor high order derivatives are required by the algorithm, it provides an efficient and robust method for tool path computation. Besides, the new algorithm is capable of producing tool paths whose milling directions are based on local surface geometry.

1 Introduction

The common practice, by far, for free-form surface milling has been done by directing the tool to follow a number of paths which lie on an approximation to the surface. Critical to this process is deciding how many tool paths on a surface should be generated, where they should be, how many points one must specify along a particular tool path, and where these points should be placed . Except for a few methods [15], tool paths for a surface are either chosen to be parallel lines in either the parametric or geometric domain. That is, isoparametric curves are specified or the intersection curves of the surface and a family of parallel planes are used. [14,10,2]. The parametric spacing and the distance between parallel planes are usually either specified explicitly by users or placed at equally-spaced positions in the parametric or spatial domain.

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

Even if the specification of tolerances is allowed, most NC systems compute scallop height at only several discrete places along adjoining paths to decide the spacing of the tool paths. Tool positions along a tool path are chosen in the same ways as for tool paths, except that in some systems scallop sizes are measured between two consecutive points for the placement of every tool position.

Systems which do estimate the scallop sizes between consecutive tool positions on non-planar surfaces usually employ some numerical computation and/or local geometry approximation [14,15]. Local approximation to the geometry is necessary in order to simplify the computation. Most of these systems produce satisfactory tool positions and paths on smooth, simple surfaces, but unsatisfactory results for complex surfaces with rapid changes in derivatives or unsmooth shapes. They are usually defined over one single "regular" surface and ignore derivative discontinuities embedded in the surfaces.

Inefficiency is a major problem with all the methods which produce tool paths along either isoparametric curves or along curves formed by the intersection of the surface with parallel planes. Uniform spacing between the planes or between the values defining the isoparametric curves does not guarantee uniform scallop height, so it is intrinsically difficult to maintain constant maximum scallop height using these methods on high curvature surfaces and those with parametrizations that correspond to nonuniform geometry. Either strategy generates more tool paths than necessary because maintaining a certain scallop height in one region for the surface will force a much smaller scallop height in other regions of the surface. And in regions with small scallop heights, the density of the tool paths is wasted. Since machine time is roughly proportional to the total length of the tool paths, extra tool paths will increase the machine time, which is undesirable.

Mortensen [15] worked on the generation of constant scallop height tool paths for sculptured surfaces. He remarked that, even though specification of scallop height tolerances is allowed in most of the commercial NC machining systems, few of them can guarantee that on sculptured surfaces the tolerances will be satisfied by the generated tool paths, and some of them produce tool paths with very uneven scallop height between paths. Tool paths are not necessarily along isoparametric curves or on parallel planes in his approach. The tool positions are generated one at a time on each path. The allowable distance to step forward along a tool path and the allowable side stepping distance perpendicular to the adjacent tool path are computed according to the tolerances. A tool position is chosen based on these distances. The computations are done by applying methods in differential geometry and by approximating the surface region around the current tool position by a low order patch. His method works well on smooth surfaces but has trouble with surfaces with rapidly changing derivatives. Since each tool position must be computed using methods from differential geometry, we suspect that the method is relatively slow.

In traditional tool path computations, like the APT approach, numerical methods (e.g., Newton's iteration) and high order (greater than the second order) derivatives are required. These computations can produce satisfactory tool positions and tool paths on smooth, simple surfaces but on complex surfaces with high orders or rapidly changing derivatives, or on unsmooth shapes, the results are less satisfactory, or, are not produced. In this paper, a new approach is taken toward constant scallop height tool path generation which uses neither nu-

merical methods nor high order derivatives in the computations. The approach is capable of handling surfaces of arbitrary degrees, and its robustness does not depend on the degree of the surfaces. The generated tool paths are neither along isoparametric curves nor on parallel planes, but instead, based on local geometry, the algorithms automatically generates optimal tool paths. Although ball end tools are assumed in the description of the approach, tools with other shapes may be used with slight modifications. Also, the approach is capable of generating tool paths for both 3- and 5-axis milling.

Given a ball end tool and a surface, the tool path generation method has three steps.

1. *Compute the appropriate offset surface approximation to the given surface. If the offset has self-intersecting loops, they should be trimmed off.*
2. *Obtain a low order approximation to the offset, usually planar or bilinear.*
3. *Generate the path for the center of the tool from the approximation by the tool path generation algorithm presented here.*
4. *Tip positions of the tool are computed from the central points of the tool. These tip positions form a cutter location file and can be fed to the postprocessor of the NC machine.*

When this strategy is used with B-spline surfaces, methods for computing approximations to their offsets as presented in [8,9,16,1] can be used. An in-depth discussion of the issues in offset computation for tool path generation is addressed in [4]. The methods which follow assume that an appropriate offset surface is already obtained. In the example in Section 4, we use the offset operator in [1] to obtain a B-spline surface which approximates the offset of the original surface.

B-splines have the convex hull property, so the surface always lies within the convex hull of the control points so it is easy to bound the surface. Adaptive subdivision implemented with the Oslo Algorithm [7] can be applied to the offset approximation to obtain a first order approximation to the entire surface. The algorithm tends to produce large polygons cover large flat spots on the surface and produce small, shape following polygons for regions with dramatic change in geometry.

Introductory information about the tool path generation algorithm is given in Section 2, while the algorithm is detailed in Section 3. An application of the approach to a reasonably complex model is described in Section 4. Finally, remaining issues are discussed in Section 5.

2 Introduction to the Algorithm

2.1 The Graphics Analogy

Two main steps are followed when rendering images of objects bounded by tensor product B-spline surfaces. First, the above adaptive subdivision process is applied to the B-spline surfaces to produce a first order approximation. With a single user specified control parameter, this robust process can automatically produce an approximation as close to the original as necessary. Second, some process is used to fill each pixel in the polygons of the approximation, usually a scan line (or plane sweep) process. Embedded in this second stage is shading model which represents smooth changes in shade within the polygon by normal or shade interpolation.

Most of the time these rendering techniques accurately convey the shape of the object, so adaptive subdivision and normal interpolation increase the fidelity of the image. The time efficiency of the shading techniques is gained by the scan line process. The algorithm proceeds by first sorting the polygons; then the scan lines are visited in a screen space vertical order. Within each scan line, the pixels are visited and filled horizontally in sequence. Geometric coherence gained by the sorting and plane sweep makes the shading process extremely efficient. Our algorithm is a direct analogy of the shading process that we have described. And indeed, in a simplified view, we can regard surface milling as a form of surface shading, not by electronic beams but by a mechanical pen (the machine tool).

As it is discussed above, polygon shading produces a high quality image. The process of shading is quite automatic without extensive human interaction; with a single resolution parameter the subdivision process produces a desired degree one approximation to the surfaces. It is difficult to generate a constant scallop height tool path for a free-form surface, but to generate such a tool path for a planar polygon is trivial and can be handled efficiently. Also since NC machines in multiaxis mode perform a kind of normal interpolation automatically, we get this corresponding part of the *shading process* free in multiaxis machining.

Since the polygonization and the normal interpolation processes in image rendering are directly applicable to tool path generation, the remaining problem in the employment of the shading approach to tool path generation is to move the machine tool through the polygons in the approximation. Working on the finite discrete screen space is adequate for the graphic shading process, which makes the algorithms for shading simpler. And the buffering of the screen makes the order of the shading of the pixels unimportant, even though algorithms, like plane sweep, may impose an order. In contrast, object space coordinates must be used in NC machining, and we must consider how to connect the tool positions into a continuous path, as tool paths formed by ordering the connections of a set of tool positions differently may produce scallops of different sizes.

In addition to the reasons stated above, the plane sweep process utilized in the graphics rendering procedure is not directly applicable to tool path generation. For example, a plane sweep process for tool path generation may pass a plane perpendicular to the z axis in the machine space from the maximum to the minimum z value of the part. And on each incremental position of the sweep plane, the intersection of the plane and the approximation of the surfaces is found. Then such intersections are taken as the tool paths. This approach will result in an inefficient set of tool paths. For a given scallop size, polygons with different incline angles to the sweep plane require different increments in positioning the sweep plane. To insure scallop heights everywhere are kept less than the tolerance, only the smallest increment can be used. But then, tool paths are placed closer than necessary on polygons in which a larger increment would be allowable. And that results in extra tool paths. Also in order to gain maximum efficiency in NC machining, the optimal milling directions of the polygons may be different from polygon to polygon, and minimization should be performed on the total length of waste cuts and the total number of tool retractions. It is not known how to achieve these goals using a simple plane sweep approach.

2.2 Milling and Graph Theory

Since the plane sweep scheme is not directly applicable to tool path generation without creating wasted paths, we have designed a different strategy. We formulate the problem as finding a systematic way to visit each polygon of the approximation once and perform the necessary milling action the whole polygon at that time. We then create a planar graph G whose vertices and edges correspond to the vertices and edges of polygons of the loop trimmed approximation, and find the graph of the geometric dual [17] of the polygonization, , denoted Q, and use graph theory to solve the problem. Note that vertices in Q and the polygons in the polygonization are in one-to-one correspondence, and each edge of Q represents a pair of adjacent polygons.

The problem of planning the motion of the tool to move through each of the polygons exactly once can be solved by finding a path which visits all the vertices of Q exactly once. This path finding problem is the well known Hamiltonian path problem of graph theory. There are no known systematic ways to find such a path, and it has been proved that there are some graphs for which no Hamiltonian paths exist. On the other hand, we can choose to relax the restriction of moving through each vertex of Q exactly once in NC machining. That is, we can let the tool visit vertices of Q more than one time. However, we tag each of the visited polygons so that when we visit the polygon at a later time, no milling action is performed, or we retract the tool to move it between polygons to achieve efficiency. Then the problem can be formulated as a version of the Traveling Salesman Problem.

However another consideration must be taken into account. When high resolution is required, the number of polygons for a complex surface will increase dramatically, and at the same time the size of each polygon will decrease accordingly. Frequently, the size of the polygons will be so small that it will not be necessary to clean the interior of the polygon. So, since the tool will move along most of the edges in G twice, once for the boundary of each bordering polygon, moving through the small polygons would be wasteful. In the extreme case, all of the polygons would be so small that none would require the tool to move through their interiors so moving the tool through all the edges of G would be enough. To gain maximum efficiency, the path would be required to pass through each of the edges of G exactly once. This is also a very well known problem in graph theory, the Euler path problem. The Euler theorem tells us that an undirected graph has an Euler path if and only if the graph is connected and has exactly 0 or 2 vertices of odd degree. In general the graph G from a polygonization is not an Eulerian graph, hence an Euler path does not exist. Once again, to solve the problem for NC machining the restriction of passing through each edge no more than once can be relaxed. Tool retractions and traversal moves can be used to gain efficiency when an edge is passed through the second time.

Once we relax the restriction of moving through each of the edges exactly once, we would like to require the total cost of the tool moves to be minimized in order to get optimal results. The length of each tool move with milling action can be taken as its cost with different costs assigned to retractions and traversal moves. The minimum cost problem is again a known problem in graph theory called the Chinese Postman Problem (CPP) and was first formulated in 1960's [13]. A solution can be found in [11], and requires a perfect matching of a graph.

Given an undirected graph $G = (E, V)$, where E is the set of edges and V is the set of vertices, a *matching* is a subset of edges, M, of E, such that no two edges of M are adjacent. A *perfect matching* is a matching, M, whose set of end vertices is V. Two steps are involved in the solution. First, the perfect matching with minimum cost on the complete graph formed by all the vertices of odd degrees from the graph at hand is obtained. Second, the graph at hand is augmented with the edges from the perfect matching. Then, a solution to the CPP is an Euler path on the augmented graph. However, even though an exact solution can be found, the computational cost of such a method is very high ($O(n^4)$), and the algorithm is very complex. There is a high computational cost and complexity in finding the perfect matching with minimum cost.

In [11,12], methods are suggested to solve the problem in real time by using a perfect match which may not be minimal cost. The approximate methods lower the time required to solve the first step by removing the requirement of *a minimum cost* perfect matching. This approach also has two steps. First, the edges of the complete graph are sorted by cost. Second, edges in order of increasing costs are chosen, excluding edges adjacent to those previously selected. These chosen edges are taken as the edges in the perfect matching. The main computational cost of such methods occurs on the sorting step, which can be done in $O(n^2 \log n)$ time and so is faster. Also, algorithmically, sorting is less complex than the steps required in finding the minimum cost perfect matching. Of course such an algorithm is "greedy" and may not find a minimal total cost even though it is locally minimal. Note that the difference in the costs of the resulting paths between an approximate method and the exact method lies on the costs of the edges in the perfect matching.

Most practical adaptive polygonizations of a surface will be a mix of polygons with both large and small sizes. In this case, one way to solve the related graph theoretic problem is to move the tool through the edges of G and through the faces of G only when necessary for scallop height control. This formulation will enjoy efficiency of both traversing through edges and traversing through polygons if the polygonization is composed of polygons with both large and small sizes. The tool path generation algorithm presented here assumes that the average case consists of such a mixture.

3 The NC Shading Algorithm

In this section, the details of the tool path generation algorithm are given. In order to avoid possible confusion, the following terms are defined and used throughout this section. The ter *graph* will be used for both graph and multigraph, although a graph is only a subset of a multigraph in graph theory.

Edge: An edge is an arc in a graph with its usual meaning in graph theory.

Side: A side is a maximally-connected collinear portion of the boundary of a polygon. A side can pass through several vertices of a polygon, if these vertices are connected by collinear line segments from the boundary.

Path: A path is an ordered sequence of edges so that neighbors in the sequence have a vertex in common, the same as its usual meaning in graph theory.

Tool Path: A tool path is a sequence of tool positions through which the tool moves.

As described in the introduction, a polygonal approximation to the offset is required by the algorithm. We assume that this polygonal approximation is represented by the graph G. In the general case, the approximation is a mixture of polygons of various sizes, with polygons so large that their interiors must be moved through by the tool for scallop height control. We denote the set of these large polygons as $\mathcal{D}$. In the tool path generation algorithm, tool paths are generated to move through the edges of G and the interiors of the polygons in $\mathcal{D}$. Roughly, the algorithm produces the tool paths by traversing through the graph. It generates the tool paths moving through the interiors of the polygons in $\mathcal{D}$ by associating a special edge with each of the polygons. And then, later on, tool paths for the interiors are generated when we move through these special edges in the graph traversal.

For a polygon $P \in \mathcal{D}$, the algorithm generates a zigzag tool path for the interior of P. There are two kinds of segments in the zigzag tool path. The first kind of segment is inside the polygon and parallel to a fixed side of the polygon, a plane sweep (scanline) approach. The other kind is on the boundary of the polygon and is used to connect the segments of the first kind.

For a convex polygon, the selection of the side to which the segments are parallel, the plane sweeping direction, can be used to optimize the tool path to some extent. The goal is to minimize the number of segments and to maximize the possible length of the segments in the tool path. The maximum spacing between the parallel segments is determined for a given tolerance and tool radius. Then, for a convex polygon, the number of segments in the tool path is determined by the maximum perpendicular distance from the vertices of the polygon to the selected side. So in order to minimize the number of segments and to maximize the possible length of the segments, the side should be chosen so that the fore-mentioned maximum perpendicular distance is minimal. For a concave polygon, there is the added constraint of avoiding tool retractions within the polygon. This can be achieved for four-sided polygons if one of the sides which are not connected to the concave vertex is selected. For other types of polygons the optimal selection of the side can be done accordingly. The zigzag tool path will end at a vertex of P when we start from the selected side and move across the polygon through the zigzag tool path. We denote this ending vertex as O.

The algorithm is composed of four major steps, where $\mathcal{D}$ is the set of large polygons:

1. *Process each polygon in the set $\mathcal{D}$ and associate a special edge with each polygon.*
2. *Derive a second graph $\overline{G}$ from the graph G based on the information obtained in step 1.*
3. *Solve the CPP on graph $\overline{G}$ and obtain an Euler path.*
4. *Generate the tool path based on the Euler path obtained in step 3.*

Details of each step follow.

3.1 Step 1

Given a polygon P in $\mathcal{D}$, the following steps are performed.

A. Find the side, denoted as S, rendering the optimal tool path for the interior of P. And at the same time, obtain the vertex O at the end of the tool path.

B. Construct the special edge, e, associated with P.

   a. If S corresponds to more than one edge in G, a new edge is created incident to the two vertices at the ends of S. At the same time, we mark all the edges in G which form S.

   b. If S corresponds to only one edge in G and the edge is not marked, the edge is taken as e. At the same time, we mark the edge.

   c. In the remaining case in which S corresponds to only one edge and the edge is marked, we make a copy of the edge and take the copy as e.

C. Select the "return path" and mark all its edges. A return path is a sequence of edges which corresponds to a part of the boundary of P and connects the vertex O and an end vertex of the edge e. Two paths are possible. The one which has smaller Euclidean length is taken as the return path. The end vertex of e which is an end vertex of the return path is denoted as B.

E. Tag the return path and the vertices B and O to the special edge e for later use in step 4.

The set of the special edges constructed in this step is denoted as $\mathcal{F}$.

3.2 Step 2

In this step we construct a second graph $\overline{G}$. The edge set of $\overline{G}$ consists of the special edges in $\mathcal{F}$ and those edges in the graph G which have not been marked in step 1. The vertices of $\overline{G}$ are the vertices of G, which are not isolated vertices with respect to the edges of $\overline{G}$.

3.3 Step 3

Now, the tool path generation problem has been transformed to a CPP on the graph $\overline{G}$, that is to traverse through each edge of the graph $\overline{G}$ at least once and at the same time to minimize the total cost of the complete path. This can be solved in two steps: finding the perfect matching and then finding an Euler path. The perfect matching is found on a complete graph G_k, formed by all the vertices of $\overline{G}$ with odd degrees. If tool retractions and traversal moves of the tool are used, the edges corresponding to these motions between the vertices of G_k can be taken as the edges of G_k for the perfect matching, and the costs of such edges are relatively small.

The time required to move the tool can be used as the cost measurement. Using a simplified model, the time to move the tool through an edge with milling action is proportional to the Euclidean length of the edge. The time for tool traversal moves is much less than that for moves with milling action. A retraction move of the tool is more time consuming than a traversal move, since each retraction is followed by plunging back into the material to resume the machining, and which must be done with a slow feedrate if the material at the place of plunging was not removed previously. But a retraction move takes less time than a move with milling action, especially when the move with milling action is long. The computational cost of finding a problem of perfect matching with minimum cost is high, and algorithms to do so are complex. However, the difference in the costs of the resulting paths between an approximation and the exact method lies in only the costs of the edges in their respective perfect matchings. Since the costs of the edges in G_k are relatively small, it is unnecessary to require the perfect matching on G_k to be of minimum cost, so a suboptimal method is used to find a perfect matching on G_k. The method in [11,12], outlined in Section 2, could be used to find a perfect matching. But sorting the edges of G_k sometimes requires too much time for the process to be interactive, since for a complex surface the number of vertices of G is on the order of thousands and the number of vertices of G_k is on the order of hundreds. Hence a different approach is needed.

Since traversal moves without retractions can be used to move the tool along a path if all the edges in the path have been machined previously, if an edge of G_k is created so that it corresponds to a path in $\overline{G}$ and if we can make sure the path in $\overline{G}$ is machined before the edge in G_k is moved through, we can use traversal moves for the tool to pass through the edge without tool retractions. The advantage of using traversal moves instead of tool retractions for tool motion is that the tool motion with traversal moves is less time consuming.

From the above observation, we would like to create the edges of G_k so that each of them corresponds to a path in $\overline{G}$, and we want to make sure the path to be machined before the corresponding edge in G_k is moved through. This is a quite difficult task, in general, especially to make sure the path to be machined before the edge is used for traversal moves. In order to keep the algorithm simple, we create an edge of G_k for traversal moves only if the two vertices of G_k to which the edge is incident are neighbors in $\overline{G}$. Our algorithm for finding a perfect matching is to

A. Pair as many vertices in G_k as possible by the criteria that they be neighbors in $\overline{G}$ and that each of them is paired with at most one other vertex in G_k. In the pairing process, if a vertex v is a neighbor to more than one unpaired vertex, the vertex which has the minimum distance to v is selected.

B. Create a new edge between each pair of vertices obtained in step 3.3. The set of edges so obtained is denoted as T, and will be a subset of the perfect matching of G_k.

C. Form another complete graph $\overline{G}_k$ whose vertices are the vertices of G_k that remain unpaired after step 3.3.

D. Apply the method in [11], described in Section 2, to find a perfect matching on $\overline{G}_k$. The set of edges obtained in this step is denoted as $\mathcal{U}$. It is obvious that the set of edges in $\mathcal{T}$ and $\mathcal{U}$ is a perfect matching of G_k.

E. Add the edges in $\mathcal{T}$ and in $\mathcal{U}$ to graph $\overline{G}$ and call this new graph $\tilde{G}$.

F. Find an Euler path on $\tilde{G}$. For an edge e in $\mathcal{T}$, there is an edge $\bar{e}$ in the graph $\overline{G}$ which has the same connectivity as e. When generating the Euler path, we should tag the edge, either e or $\bar{e}$, which is passed through last by the path. The set of edges so tagged are denoted as $\mathcal{H}$.

In the matching algorithm, step 3.3 can be done by going through each of the vertices in G_k once and is a very fast process. The method in step 3.3 requires sorting the edges of the graph $\overline{G}_k$. By the nature of the graph $\overline{G}$, most of its vertices with odd degrees have neighbors of odd degree. Hence step A will pair most of the vertices and leave many fewer vertices in graph $\overline{G}_k$ than in G_k. This will make the sorting in step D much faster than what would be if the sorting were applied on G_k directly. Algorithms for step F can be found in most books on graph theory, for example [3].

3.4 Step 4

The final step in the algorithm is the tool path generation. The basic skeleton of this step is to traverse through the Euler path found in the previous step and, at the same time, produce the sequence of tool positions for moving the tool on the polygonal approximation according to the path. We generate the tool paths while traversing the Euler path depending on the type of the edge encountered.

A. If the edge is in $\mathcal{H}$, traversal feed is used to move the tool along the edge without tool retraction.

B. If the edge is in $\mathcal{U}$, a tool path is generated to retract the tool, fast move to the point above the other end vertex of the edge, and then down feed to the point at the end vertex.

C. If the edge e is in $\mathcal{F}$, i.e., e is a special edge, the zigzag tool path for the interior of the polygon corresponding to this edge is generated. First, we retrieve the return path and the vertices O and B from the edge. This information was tagged to the edge in step 1 of the algorithm. Then, the tool path is generated in different ways depending on whether the vertex on which we encounter the edge e is the vertex B on the edge or not.

   a. If it is, we first move the tool from the vertex B through the return path to the vertex O. Then the zigzag path described previously is generated to move the tool from O toward the edge e and back to the vertex B. Finally, we move the tool through the side S corresponding to e to reach the other end vertex of the edge.

   b. If the encountered vertex is not the vertex B, we first move the tool through the side S corresponding to e to the vertex B. Then the zigzag path is generated to move the tool from vertex B to vertex O. Finally, the return path is followed back to B.

While the tool path between polygons is not an Euler path, the only edges which are covered more than once are those in $\mathcal{H}$ and those edges copied in step 1.B.c. Traversal feed is used in moving through edges in $\mathcal{H}$. With careful implementation, the edges copied in step 1.B.c can be moved through by traversal moves too. The number of tool retractions is minimized by identifying the edges in $\mathcal{H}$ and by solving the CPP. So the algorithm generates a tool path with nearly minimum machine time.

Even though a zigzag tool path is proposed for the interior of a polygon in $\mathcal{D}$, other types of tool paths can be used too. The only requirement of the tool path is that it starts from one side of the polygon and ends at a vertex. One determining criterion on choosing the tool path is that the tool path should be efficient.

4 An Example

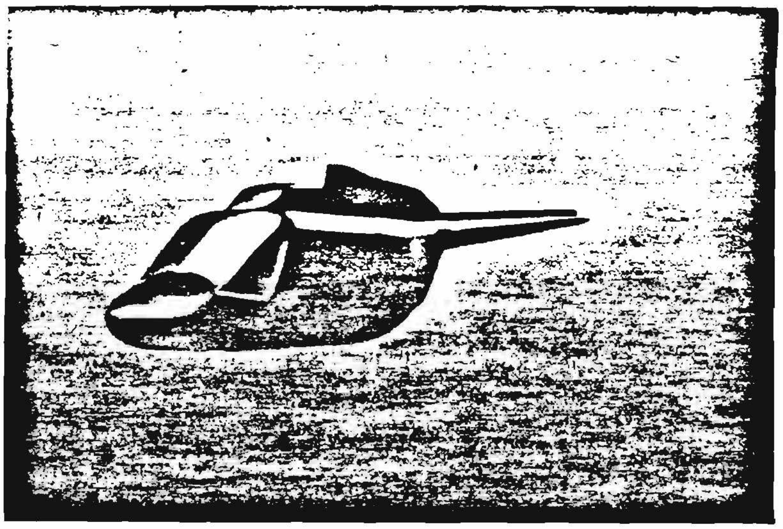
A helicopter model (see Figure 1) is chosen as an example to demonstrate the proposed approach. The model was constructed in 1983 [6]. We chose to produce a hard copy of the body portion of the helicopter model, since the other parts, such as the skids, are not suitable for manufacture by milling machines. Also since the body of the helicopter is left-right symmetric, only half of the body is made. To construct the whole body, two halves can be made and joined together. In order to simplify the fixturing problem, 3-axis milling is used, although the algorithm is capable of producing tool paths for 5-axis milling.

The main reason for choosing the helicopter model as an example is that the model is complex enough to be interesting and to expose the strength and problems in the approach. There are two surfaces in the body of the helicopter, both of which are nonuniform B-splines that are tangent continuous everywhere except at the tip of the tail. The first surface is for the cowling which starts right behind the windshield and is extended all the way to the tail. The cowling is above another surface, the main body, which makes up the front nose, windshield, side vent windows, passenger cabin and the main boom. This surface extends for the full length of the helicopter.

To simplify the offset surface-to-polygonization step and avoid the surface/loop trimming problem, which are not a topic of this research, a modified version of the original model was constructed using the sweep operator in [1], that was smoothed with a tool radius to insure that the surface does not self-intersect. After the offset approximation was obtained, the recursive subdivision process produced a polygonized approximation to the offset consisting of four-sided patches. Finally the tool path generation algorithm is used to produce the tool path. Figure 2 shows the generated tool path. The patches in the approximation are nearly flat, bilinear patches, but they are not necessarily planar, since, the tool path generation algorithm can be used on such patches.

We made one half of the model from a rectangular block of wax. The tool path shown in 2 was used for the finish-cut, but a separate roughing process was performed first. The

Figure 1: The Helicopter Model.



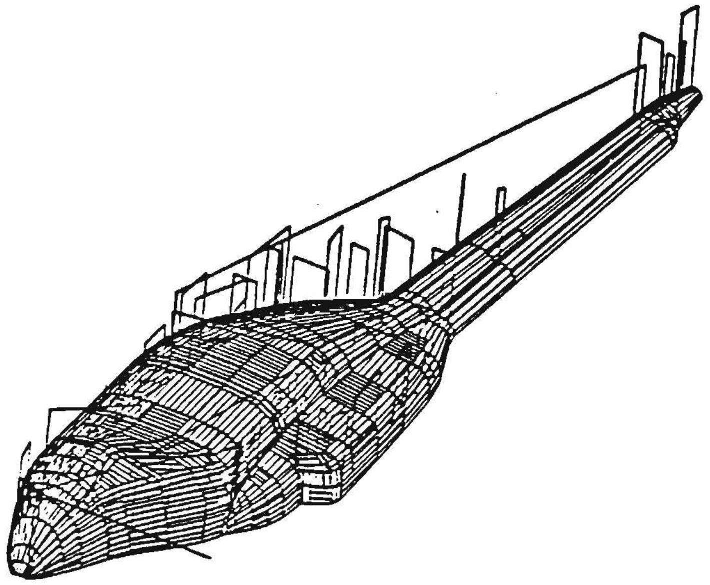


Figure 2: The Finish-Cut Tool Path for the Helicopter From the NC Shading Algorithm.

roughing process used a flatend tool to move on four planes parallel to the base of the stock at increasing depth. Each of the planes was used to intersect the helicopter model to produce a contour. On each plane, the rectangle enclosing the stock and the contour on the plane in the rectangle form a polygonal region. The polygonal region is in-between the rectangle and the contour. The tool path for removing the stock material in the polygonal region was generated using the pocketing algorithm in [4,5] which produces a a spiral-like tool path based on the Voronoi diagram. Applying the pocketing algorithm on all the four polygonal regions, one on each plane, produces the tool paths for roughing. Figure 3 shows the results of the roughing process. Figure 4 shows the half of the helicopter model after the finish-cut. A half inch flatend mill was used in roughing; a quarter inch ballend mill was used in the finish-cut.

5 Conclusion

We have drawn upon computer graphics and graph theory to creat a new method for generating efficient constant scallop height toolpaths automatically. We can see in figure 4 that the tool path generation algorithm produces efficient tool paths. The tool paths in the small patches in the subdivision provide details to the part, and at the same time those in the large patches provide efficiency. The algorithm creates many fewer tool paths at the nose and the tail than those in the middle bulge portion, even though the same parameter range is used for the surface throughout the whole length of the helicopter. We also find the tool paths have a nearly constant scallop height. This is achieved by the adaptive subdivision producing the approximation and by the optimal zigzag tool paths. With the nearly constant scallop height tool paths and the optimization done by the algorithm in solving the CPP, the algorithm produces tool paths which have minimum waste tool moves and are nearly optimal in machine time. The algorithm not only decides the number of tool paths but also selects the optimal milling directions based

Figure 3: The Wax Helicopter Model After Roughing With Pocketing.

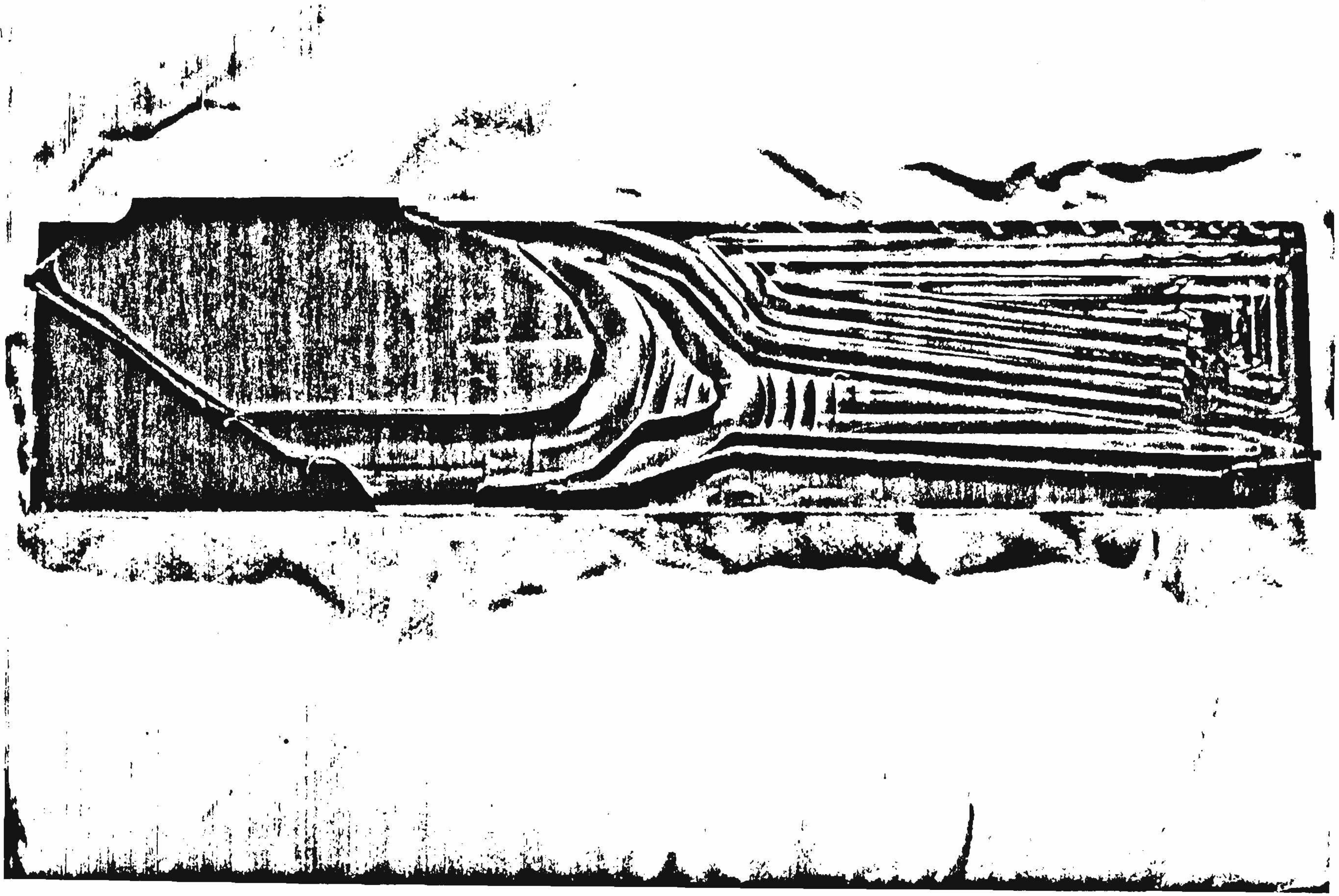
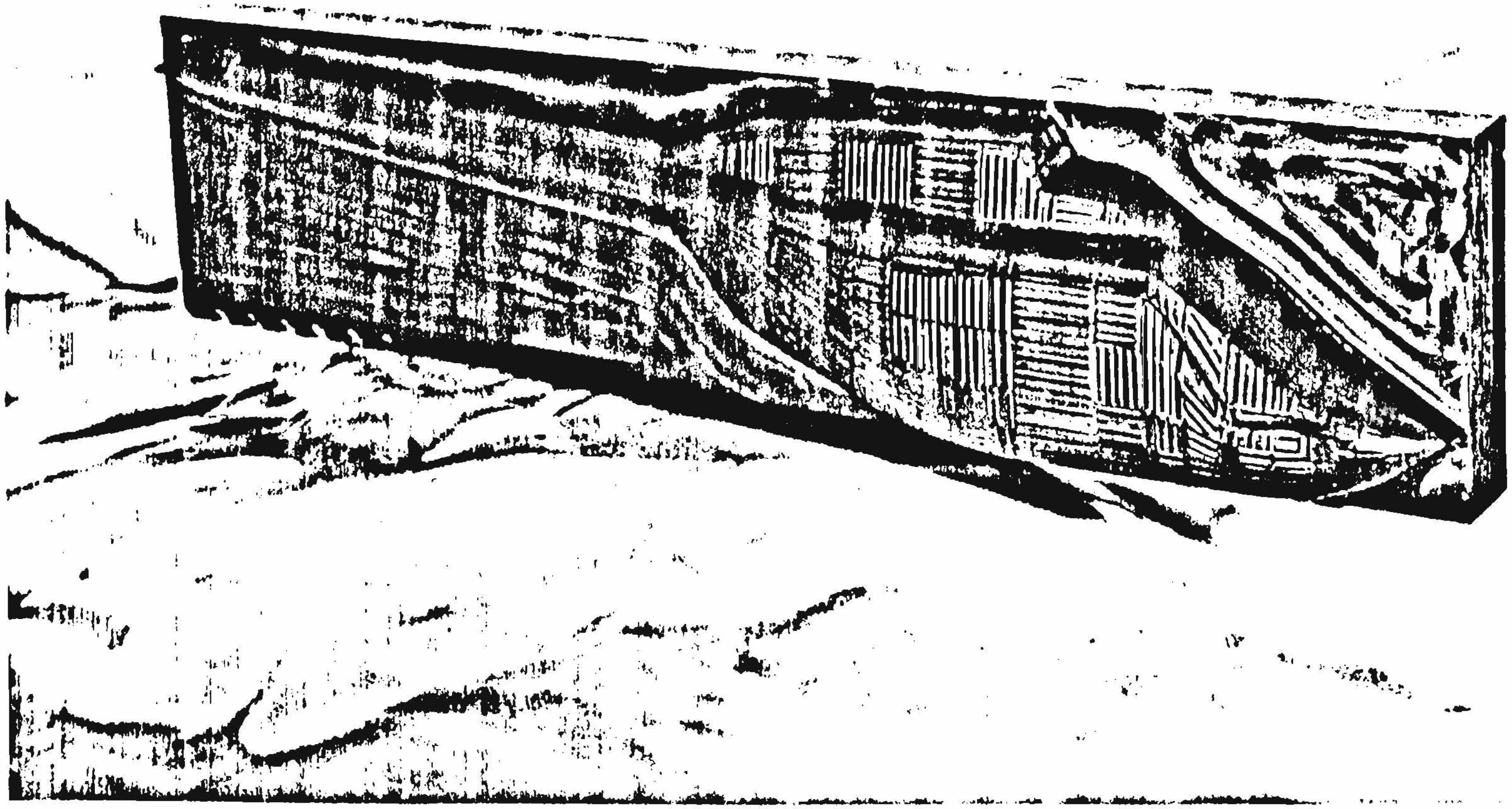


Figure 4: The Wax Helicopter Model After Finish-Cut.



on the local geometry of the surfaces. Additionally, the directions are not confined to the isoparametric directions of the surface. For example, none of the parametric directions in the vents is a good milling direction. Actually, not a single direction seems to be the best. The tool path generation algorithm automatically generates tool paths along several directions for the vent area.

Polygons are used in the description of the algorithm. There are cases in which the desirable approximations to the surfaces are composed of patches which are very close to polygons, but are not exactly planar. We can apply the algorithm to these patches as long as the tool paths for the interiors of the patches can be generated conveniently.

Although 3-axis milling is used in the example, the algorithm is capable of generating tool paths for 5-axis milling with minor modifications. B-spline surfaces are used in the discussion; however, any surface representation can be used as long as an offset surface approximation and a first order approximation to the offset can be computed. If the tool path generation algorithm are used for face milling on convex surfaces with flatend tools, the offset surface to the original surface is not required.

References

[1] Engineering Geometry Systems. *Alpha_1 Users Manual.* Salt Lake City, Utah, USA, 1988.

[2] Bobrow, J. NC machine tool path generation from CSG part representations. *Computer-aided Design 17*, 2 (March 1985), 69–76.

[3] Chachra, V., Ghare, P. M., and Moore, J. M., Eds. *Applications of Graph Theory Algorithms.* North-Holland, New York, 1979.

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

[5] Chou, J. J., and Cohen, E. */Computing Offsets and Tool Paths with Voronoi Diagrams.* University of Utah, Computer Science Tech. Report UUCS-89-017, 1989.

[6] Cohen, E. Some mathematical tools for a modeler's workbench. *IEEE CG&A 3* (Oct. 1983), 63 – 66.

[7] Cohen, E., Lyche, T., and Riesenfeld, R. Discrete B-splines and subdivision techniques in computer-aided geometric design and computer graphics. *Computer Graphics and Image Processing 14* (1980), 87–111.

[8] Farouki, R. T. The approximation of non-degenerate offset surfaces. *Computer Aided Geometric Design 3* (1986), 15 – 43.

[9] Farouki, R. T. Exact offset procedure for simple solids. *Computer Aided Geometric Design 2* (1985), 257 – 279.

[10] Gaal, B., and Varady, T. Experiences and further development of the FFS (Free-Form Shapes) CAD/CAM system. *Robotics & Computer-Integrated Manufacturing 2*, 2 (1985), 149 – 154.

[11] Gondran, M., and Minoux, M., Eds. *Graphs and Algorithms.* John Wiley & Sons, New York, 1984.

[12] Goodman, S., and Hedetniemi, S. Eulerian walks in graphs. *SIAM Journal of Computing 2* (1973), 16–17.

[13] Kwan, M. Graphic programming using odd or even points. *Chinese Mathematics 1* (1962).

[14] Loney, G. C., and Ozsoy, T. M. Interactive design & NC machining of free form surfaces. *Computer-aided Design 19*, 12 (March 1987), 85 – 90.

[15] Mortensen, F. L. *Constant Scallop Height Tool Paths for Sculptured Surfaces.* Master's thesis, Brigham Young University, 1988.

[16] Patrikalakis, N. M., and Prakash, P. V. Free-from plate modeling using offset surfaces. In *6th Int. Symposium on Offshore Mechanics and Arctic Engineering, A.S.M.E.* (1985).

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