Algorithmica (1989) 4: 33-60

Algorithmica


# Discrete Simulation of NC Machining[1]

Robert L. (Scot) Drysdale, III,[2] Robert B. Jerard,[3] Barry Schaudt,[2] and Ken Hauck[2]

**Abstract.** We describe a method for simulating and verifying the correctness of Numerical Control (NC) programs. NC programs contain the sequence of cutting tool movements which machine raw stock into a finished object. Our method is based on a discrete approximation of the object by a set of points. A vector is passed through each of the points and machining is simulated by finding the intersections of tool movements with these vectors. We present a point-selection method and an analysis that shows that the error introduced by the approximation can be made as small as desired. The run time is inversely proportional to the allowable error and the size of the cutting tool, and directly proportional to the distance that the cutting tool moves.

**Key Words.** Numerical control, CAD/CAM, NC verification, NC simulation.

**1. Introduction.** The machining of sculptured surfaces with numerical control (NC) is a common practice in many industries. Sculptured surfaces arise in the design of car bodies, ship hulls, aircraft, and other applications where smooth surfaces are needed to reduce resistance from air or water or simply for esthetic reasons. Coons patches [Co], Bézier curves [B], B-splines [GR], and other methods have been invented to represent such surfaces. Most commercial computer-aided geometric design systems have a sculptured surface capability. Once a surface has been designed, it is usually necessary to create a numerically controlled milling program to cut the surface on a milling machine. Either a human programmer or a fully or semiautomated process creates the NC program. A number of automated systems are described in the literature [Ar], [ACd], [DM], [SJW], [ZB]. However, NC programs created by humans and automated systems have errors where they cut too deeply or leave too much material. Therefore an important practical problem is finding the answer to the question, "Given a mathematically defined surface and a file of NC tool movements, does the shape that the tool cuts match the mathematical shape to within a given tolerance?" Fridshal [FCDZ] claims that "Current methods of verifying NC part programs result in one of the highest nonrecurring cost factors in producing NC machined parts within the aerospace industry."

[1] This research was supported in part by the National Science Foundation under Contract No. DMC 8512621 and by the Ford Motor Company.
[2] Department of Mathematics and Computer Science, Dartmouth College, Hanover, NH 03755, USA.
[3] Department of Mechanical Engineering, University of New Hampshire, Durham, NH 032824, USA.

Received July 1, 1987; revised January 29, 1988. Communicated by Chee-Keng Yap.

One approach to solving this problem is to start with a solid model representation of the block of material and to simulate the movements of the cutting tool. As each cut is made, the solid model representation is updated. Unfortunately, if there are thousands of tool movements, this becomes computationally infeasible. Furthermore, there is still the difficult problem of determining if the cut surface model matches the mathematically defined surface. Even checking to see if a particular tool trajectory intersects a particular patch of a sculptured surface is far from trivial.

We have developed a new approach for simulating NC machining of sculptured surfaces [JHD], [JDH], [DJ]. The method replaces the exact representation of a surface by a carefully chosen sample of points on that surface. We have developed algorithms and data structures for rapidly simulating the actions of the cutting tool on this discrete set of points. We then use these results to determine if any part of the surface is out of tolerance. Unlike some previous approaches which can only detect gouging, this method can find both gouging and areas where excess material remains. A version of the system is being used by the Ford Motor Company.

This idea of simulating machining at a discrete set of points is not new [AEF], [Ch], [OG], [O], [V], [WW]. However, a problem with the approach is that the cut surface is only known at the selected points, and it may be possible for all of the chosen points to be cut correctly while substantial cutting errors exist at points not chosen. None of these other papers address the question of how large an error might be missed. They simply choose enough points (usually all points corresponding to pixels on a graphics screen) so that it seems unlikely that large errors will be missed. With our system, the user selects an allowable error and the program chooses enough points to guarantee that level of accuracy. For the common case of a ball-end cutting tool, the number of sample points and running time increase only linearly with the desired accuracy. The running time of the simulation is also proportional to the total distance that the tool moves and inversely proportional to the ball radius. Similar results are true for a toroidal-end cutting tool.

One aspect of this work is engineering considerations—the system, its implementation, the graphical user interface, etc. These questions are explored in detail in two conference papers [JHD], [JDH] and in a paper in press [JDHSM]. Photographs of cut objects color-coded to show areas within tolerance, gouged areas, and areas with excess material are included in these papers. A second aspect of this work is algorithmic analysis and computational geometry—error bounds on the simulation and analysis of running times for the algorithms. This second area is the topic of this paper. It is an expanded version of a conference paper [DJ], and it contains some ideas and results not reported in that paper.

This paper first outlines our basic approach and describes our system. It then analyzes sources of simulation error and derives error bounds. It uses these bounds to motivate a method of selecting points so that simulation error can be bounded. Even though we know the cut surface only at a finite set of points, we can guarantee tolerances at every point on the surface. The paper ends by discussing alternate approaches and future work.

*1.1. Machining Strategy.* In sculptured surface machining objects are often cut on a 3-axis mill with a ball-end cutting tool. Those not familiar with NC machining are referred to [GZ], [FP], [M], and [G]. The geometry of the part is often defined by blending parametric curves such as Bézier and B-spline forms [RA]. The blending can be accomplished in many different ways and most standard computer graphics references give explanations [FV]. The parameter space is defined by $u$ and $v$ coordinates which vary between zero and one. Each choice of $u$ and $v$ parameters determines a point in 3-space. The resulting surface is called a "patch." A complex sculptured surface is often comprised of multiple patches. The surface is defined precisely at all points and slopes and outer normal vectors can be calculated.

Once an object is defined by such surfaces, the goal is to machine it. Machine tool paths that will correctly and efficiently mill the surface must be generated. This is often a semiautomated process done by parametric indexing over each patch [FP]. What is important for this paper is the fact that these "programs" of tool path movements often have errors. Either excess material is left or the tool gouges the surface too deeply. A common difficulty is that a tool movement intended to cut one patch accidentally gouges a different patch. Another is using a tool too large for the local curvature. Work has been done on detecting these sorts of problems directly (for example, Kuttner uses a method of "truncated surfaces" to avoid gouging nearby patches [KMS] and Forrest computes local curvature to detect an oversized tool [F]), but problems still arise in practice.

The current practice in industry is to debug an NC program by milling a test block and then manually examining it for gouges or excess material. The test block is usually wood or dense foam rather than metal because it is cheaper than metal, can be cut faster, and serious "bugs" are less catastrophic to the machine. Too often this process misses errors, and weeks can be spent cutting an expensive, hardened block of steel into a stamping die of the wrong shape. A computer simulation that could detect and identify errors would be very useful.

*1.2. Related Work.* Several commercial CAD systems (for example, Computervision and CATIA) contain the capability to do limited verification of NC programs for sculptured surfaces. The system verifies that no interference has occurred between the cutter and the desired surface. Duncan and Mair [DM] report on one such approach, where a polygonal approximation of the surface is used to both plan cutting tool paths and to verify that no interference occurs. These verification techniques are undoubtedly valuable but they do not meet our definition of "simulation." Simulation implies that a model of the current state of the machined object is always present. The conventional verification techniques can find gouges but they are incapable of finding areas where excess material remains. With a model of the machined object always present during the simulation it is just as easy to find areas of unremoved material as it is to find gouges. The model could also be used to calculate the material removal rate and from that determine optimum feed rates. Tool deflection and chatter effects can be simulated. If the tool is determined to be outside the remaining solid it can move at top speed.

There have been relatively few systems which do simulation. Voelcker and Hunt did an exploratory study of the feasibility of using the PADL constructive solid geometry (CSG) modeling system for simulation of NC programs [VH], [HV]. Fridshal at General Dynamics has modified the TIPS solid modeling package to do simulation [FCDZ].

The problem with using the solid modeling approach for sculptured surfaces is that it is slow. The cost of simulation using the CSG approach is reported to be proportional to the fourth power of the number of tool movements [HV]. A typical program for sculptured surface machining could contain ten thousand movements, making the computation intractable.

An alternate approach for simulating 3-axis machining was invented by Anderson [An]. He was concerned with planning NC programs that were free of "fouling." Fouling occurs when the cutter assembly collides with the partially cut object. To detect this, he divides the base of the object into squares, and keeps track of the cut height above each square. He calls this structure a three-dimensional histogram, and represents it as a two-dimensional array of heights. Each square starts with the value of the height of the stock. Each tool movement updates the heights of the squares it passes over if it cuts lower than the currently stored height. The regular square grid makes it easy to determine quickly which squares lie under the tool path and which can be ignored. Note that there are a range of heights above each square, but he chooses the highest value so that the actual part is always within his representation and no fouling will be missed. He notes that, "Logically the smallest area worth considering has a side equal to the smallest increment that the cutter assembly can make. However, this would require an excessive amount of computer storage. ... Thus the cell size has to be a compromise between accuracy and the practical requirements of computation. In practice a size of the order of 0.1 to 0.2 of the cutter diameter has proved a reasonably good value."

An alternate approach is the "point-vector" technique of Chappel [Ch]. The surfaces of the part are approximated by selecting a set of points lying on the surface. Direction vectors are created normal to the surface at each point. A vector extends until it reaches the boundary of the original stock or intersects with another surface of the part. To simulate the cutting caused by a tool movement the intersection of each vector with that tool movement's envelope is found. The length of a vector is reduced if it intersects the envelope. An analogy can be made to mowing a field of grass. Each vector in the simulation corresponds to a blade of grass "growing" from the desired object. As the simulation progresses the blades are "mowed down." The length of the final vectors correspond to the amount of excess material (if above the surface) or the depth of the gouge (if below the surface) at that point. Chappel's paper gives a detailed algorithm for computing the intersection between a vector and a randomly oriented cylinder that represents the cutting tool. However, he does not present methods to select the points.

Oliver and Goodman at Michigan State University developed a system that uses a graphical image to select the points [OG], [O]. Their approach starts with a computer graphics image of the desired surface. The user can choose the area

of interest and the view. The image space is then used as the basis for the simulation. Each pixel on the screen is projected back onto the object surface, and this set of points becomes the approximation to the object. Then their simulation proceeds like Chappel's. The large number of vectors generated would lead to a very slow system if every vector had to be separately intersected with every tool envelope. They avoid this problem by using heuristics for quickly eliminating many of the vectors from consideration.

Wang [WW] developed a different image-space approach that uses an extended $Z$ buffer. The standard $Z$ buffer algorithm is used for elimination of hidden surfaces in computer graphics [FV]. A vector is drawn at each pixel that is normal to the plane of the screen. For each pixel the $Z$ height of the front of the surface is saved. (In the lawn-mowing analogy, each blade of grass is vertical and its $Z$ value corresponds to its height.) Wang's extended $Z$ buffer is unconventional in that both the front and back $Z$ values of the volume are stored. If the vector enters and leaves a volume several times, then a list of front and back $Z$ values are saved. This allows his method to handle 5-axis machining.

Wang is able to take the swept volume of a tool movement and replace it by a polyhedral approximation. A scan-line plane intersected with this polyhedron determines the $Z$ depth of each pixel associated with the graphic image of the swept volume. Scan-line techniques allow him to compute $Z$ values for all pixels on a scan line fairly quickly. The workpiece $Z$ buffer is then modified by comparing it with the swept volume $Z$ buffer. Each tool movement changes the graphic image of the workpiece to show the cutting action. Upon completion of the simulation the $Z$ buffer of the workpiece and that of the desired mathematically defined surface can be easily compared to find unacceptable differences. Wang reports average calculation times of about 1 second of VAX 11/780 time for each swept volume tool movement, although the times must certainly be a function of the complexity of the swept volume.

Van Hook [V] also developed an extended $Z$ buffer. His method differs from Wang's in that instead of intersecting scan lines with swept-volume envelopes he precomputes a pixel image of the cutting tool and performs Boolean subtractions of the cutter from the workpiece as he steps along a tool path. This limits his method to 3-axis cutting, where the orientation of the cutting tool does not change. (Otherwise each orientation has a different pixel image, so he cannot precompute the image to save time.) Atherton [AEF] has extended Van Hook's approach to handle 5-axis machining, although his paper does not explain the details of his method.

These point-vector techniques have several advantages over the solid-modeling approach. Their run time grows linearly with the number of tool movements, so they can handle sculptured surfaces with tens of thousands of tool movements. This promises to make them useful in practical systems. They also provide an easy way to measure exactly the differences between the cut surface and the desired surface. They can determine if any of the selected points is cut out of tolerance. Solid-modeling methods can represent the final object, but comparing it with the desired object is computationally expensive.

One drawback is that approximating the surface by a set of points introduces errors. Some of the methods used for computing intersections between vectors and tool paths introduce further errors. This is not a problem as long as the size of the error introduced is small enough for a given application. Unfortunately, none of these methods give a direct method for determining or controlling the size of this error. The point selection is either unspecified or derived from an image-space view of the object. The viewer can change the amount of error introduced by changing the viewpoint, but cannot easily determine how much error is introduced by a given view.

**2. Our Approach.** Our approach is also a point-vector approach. Its point-selection methods differ from the approaches described above in two important ways. First, our method is object-based rather than image-based. All the approaches mentioned above that specify their point-selection method choose their surface points by projecting pixels from the graphics screen down onto the object. We choose our surface points based on object curvature and on interpoint spacing. Mapping our object into image space for viewing is a final step designed for viewer convenience. It need not be done at all, because points that are out of tolerance are written to an error file. Second, this approach gives us control over the amount of error introduced by the simulation. We can choose coarse error bounds to perform rapid simulations that test for gross errors or we can choose very tight error bounds to perform extremely accurate (but slower) simulations to be certain that an NC program is correct.

A major emphasis of our work is analyzing how point spacing, tool size, and local curvature affect the amount of error introduced by the simulation. The techniques described in this paper could be used to determine the maximum simulation error introduced by the point-selection schemes used for one of the other systems, if that were desired. However, our point-selection techniques are motivated by this analysis. They pick more points in areas where they are needed and fewer points in areas where they are not needed. This is a major reason that the system runs as quickly as it does.

For efficiency reasons we use a simple $Z$ buffer approach. It is relatively simple to find the intersection of the normals from the $x$–$y$ plane at the selected points with the path of the ball-end cutter, because all vectors intersect the bottom (as opposed to the sides) of the tool envelope. The $z$ value of the cut surface at each point is stored (*Zcut*). At the start of the simulation all the *Zcut* values are set equal to the top of the raw stock. The simulation processes all the cutter location (CL) file commands and modifies a *Zcut* value whenever a cutting-tool movement would remove additional material above that point. Whenever a cut modifies the *Zcut* value at a point, the number of that cut in the CL file is saved also. When the simulation is finished the *Zcut* values can be compared with the $z$ value of the desired surface. Discrepancies can be found and the number of the cut which last changed the *Zcut* value can be written to an error file.

It is tempting to simply choose points on a regular square grid, an approach similar to Anderson's [An]. That makes it easy to design an algorithm which

only checks the points which lie under the shadow of the tool path. Such localization is a key to efficiency in this approach.

Despite its appealing simplicity, a regular $x$-$y$ grid has a number of flaws. The first is the number of points required. Areas with high curvature require close spacing to represent the surface accurately. Flatter areas allow wider point spacing. The uniform-spacing method requires that the spacing needed for the few small areas of high curvature be replicated everywhere, greatly increasing the space and time requirements for the algorithm.

A second problem is that points are given in $x$-$y$ coordinates rather than parametric coordinates. The $x$-$y$ normals must be projected onto the surface patch to find the correct value of $z$ for comparison with *Zcut*. Projecting $x$-$y$ normals onto the surface is nontrivial and normally requires a trial-and-error algorithm to go from cartesian to parametric space. This can be time consuming. This problem is the same one faced by ray tracing rendering of curved surfaces [T].

We can solve the latter problem if we select points in parametric space rather than cartesian space. It is easy to transform from parametric space to cartesian space, but hard to invert the function. The first problem can be solved by using surface-curvature and tool-size information to calculate an irregular spacing between parameter values that greatly decreases the number of required points. Closer spacing is used in areas of high curvature. Therefore, a solution that solves both problems is to select points in irregularly spaced parametric coordinates rather than regularly spaced $x$-$y$ coordinates. Techniques for computing appropriate spacing and choosing points will be discussed in a later section.

This solution creates a new problem. How can we compute efficiently which points lie under the cutting tool? The regular grid made it easy to compute the points from the tool location. If we only have a list of points in parametric coordinates we might have to check every point for every tool path, which is clearly unacceptable. Localization is required for efficiency.

This problem can be solved by sorting the points into "buckets" based on their $x$-$y$ coordinates. Instead of our regular grid of sample points in $x$-$y$ space, we will have a regular grid of rectangles in $x$-$y$ space. These rectangles can be thought of as "buckets" containing sample points (see Figure 1). Where curvature is high and sample points are dense, each rectangle will contain many sample points. Where the surface is flat, each rectangle will contain few (possibly even zero) points. The points falling in a given rectangle will be stored in a linked list corresponding to that rectangle. A given tool path passes over a certain set of rectangles and all points lying within these rectangles must be examined. These rectangles are easy to compute, because their boundaries are a regular $x$-$y$ grid (see Figure 2). If their dimensions are comparable to the tool size the number of points not lying under the tool path but contained within the rectangles will be proportional to the number of points examined. They will be the points in rectangles that overlap the boundary of the cutting path which do not lie beneath the cutting path. We can roughly approximate the ratio of the unneeded points to needed points by the area of the parts of intersected rectangles lying outside the cutting path to the area of the cutting path.

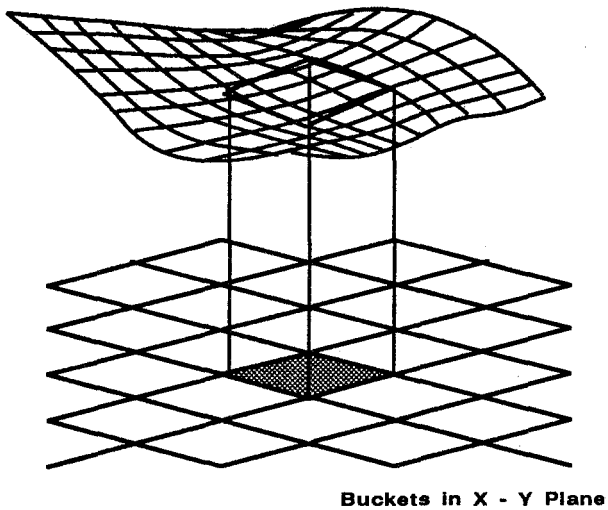


**Fig. 1.** Points calculated in parameter space are sorted into buckets with regular $x$–$y$ spacing.

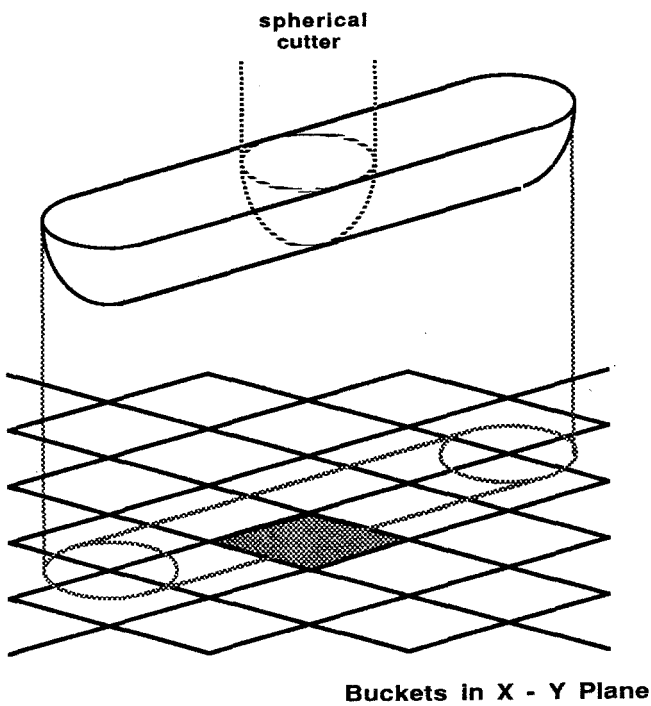


**Fig. 2.** Projection of the shadow of the cutting tool onto the buckets. Only points in those buckets are examined.

The calculation time is directly proportional to the number of intersection calculations, a number that can be estimated by multiplying the projected area of the tool path on the $x$–$y$ plane times the point density. Therefore, run time grows linearly with the number of tool movements in the CL file, provided that the average tool movement length and point density remain relatively constant.

**3. Bounding Simulation Error.** In order to simulate machining efficiently we must be able to choose points with proper spacing. The object of the simulation is to detect places where the surface is undercut by more than $T_i$ or remaining material is higher than $T_o$ above the desired surface, where $T_i$ and $T_o$ are user-defined tolerances into and out of the surface. Deviations outside the tolerance range are "cutting errors." Unfortunately, the discretization of the surface makes it possible that cutting errors may be undetected. Our simulation may determine that the surface is within tolerance at every sample point, but there may be gouging or excess material at points where we did not sample. We call the distance that the cut surface can differ from the true surface without differences detectable at the sample points "simulation error." This section shows how sample-point spacing influences the simulation error.

Our technique effectively replaces the entire surface by a sample of points whose cut values are computed. This approximation introduces simulation errors in three ways. First, our only representation of the true surface is an interpolation based on the sample points. Effectively we have replaced the surface by a polyhedral approximation obtained by triangulating the points. This approximation is not exact, so the true surface differs from the polyhedral approximation. These differences are a source of error. Errors of this type are reduced by choosing more sample points in areas of high curvature. The test for deciding if more points must be chosen in a given area is "is the distance between the interpolated surface and the true surface small in this region?"

The second source of simulation error comes from the fact that we know the cut surface height only at the sample points. It may seem that this gives little information about other points on the surface, but this is not the case as long as the sample-point spacing is small compared with the radius of the ball-end mill. In this case the ball locally appears almost flat. The amount that the tool can protrude into the polyhedral surface without a cutting error being detected is a function of the tool radius and distance between points. If the sample points are chosen correctly the height of excess material is a function of the same two factors.

A third type of simulation error comes from the fact that the differences between the *Zcut* and $z$ of the surface is a vertical distance. The actual cutting error is the distance from the cut surface to the closest point on the desired surface. If the surface is nearly vertical, the difference between the closest point and the $z$ distance can be quite large. The true shortest distance from a point to the desired surface will always be less than or equal to the distance in the $z$ direction. Therefore this type of simulation error will not cause any undetected cutting errors. Unfortunately, it can cause our program to warn that points are out of tolerance when in fact they are not. We are currently looking at postsimulation

analysis and other approaches to try to get a truer estimate of the distance from the cut point to the desired surface than simply measuring the $z$ distance.

We desire to select points in a manner that allows us to bound the size of the first two types of simulation error. The next sections quantify bounds on each type of error.

We note that other errors are introduced because our method does not exactly model the real-world cutting process. Tool movements in the CL files that our system is designed to use are described as parabolas. Some NC tools approximate these parabolas by line segments while others attempt to follow the parabolas exactly. We approximate the parabolas by line segments. We do not compute the effects of deflections caused by forces acting on the cutting tool and workpiece. We do not simulate tool wear or tool runout. These limitations are independent of our method of representing a surface by discrete points.

*3.1. Distance Between Surface and Polyhedral Approximation.* The first type of error, $e_s$, is the distance between the true surface and the polyhedron obtained by triangulating the points in our sample. It is the maximum distance from any point on the true surface to its nearest neighbor on the polyhedron. Each triangle has a region of points which are closer to it than to any other triangle, and the error for that triangle is the maximum distance from the triangle to any point in its region. Actually measuring this error requires analytical knowledge of the particular surface definition technique and/or numerical search techniques. It may be computationally easier to get some conservative bound on the error than to try to compute the exact error. (For example, a Bézier surface patch always lies within the convex hull of its control points.) The methods depend strongly on the exact method of representing the surface, and are beyond the scope of this paper. See, for instance, [BFK] for a survey of surface representations and subdivision algorithms. If a "very high probability" of errors being detected is sufficient, this type of error could be estimated by measuring the distance between the true surface and a number of points on the triangle. This trades off guaranteed error bounds for implementation ease and efficiency.

This is one of only two places that our algorithm needs information about the representation of the surface. It also needs to know how to map a $(u, v)$ point in parameter space to an $(x, y, z)$ point in object space. Thus our method reduces questions about the distances between cut surfaces and mathematical surfaces to questions about distances between triangles and mathematical surfaces and intersections between vertical rays and tool-movement envelopes.

*3.2. Tool Protrusion into the Polyhedral Approximation.* We want to find out how far below the surface of the polyhedral approximation the cutting tool can protrude without an error being detected at the vertices of the polyhedron. We call this the protrusion error $e_p$. We first derive a formula that will prove useful.

LEMMA 1. *Let $S$ be a sphere with radius $r$ and $P$ be a plane cutting through $S$. Then the maximum distance between $P$ and spherical cap cut off by $P$ is*

$$h = r - \sqrt{r^2 - s^2},$$

*where $s$ is the radius of the spherical cap.*

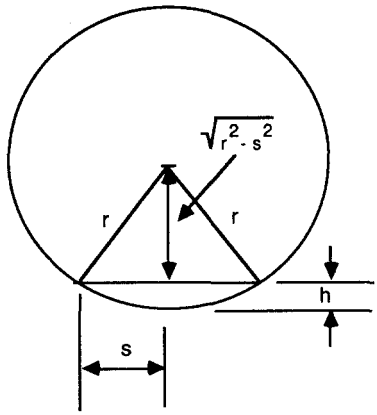


**Fig. 3.** Side view of sphere shows that the tool protrudes distance $h$ for spherical cap of radius $s$.

PROOF. Look at Figure 3 and apply the Pythagorean theorem. □

We use this fact to prove the following theorem:

THEOREM 1. *Let TP be a triangulated polyhedron, no edge of which is longer than d. Let S be a sphere of radius $r > d/\sqrt{3}$. The maximum distance that S can protrude beneath the surface of TP without containing any vertices of TP is*

$$e_p = r - \sqrt{r^2 - d^2/3}$$

*and this bound can be achieved for an equilateral triangle.*

PROOF. We want to show that any point on $S$ that lies interior to $TP$ lies within distance $e_p$ of one of $TP$'s triangular faces. We do this by considering the ways that $S$ can protrude beneath the surface of $TP$. Note that in general $S$ may protrude through several different triangles of $TP$. Our proof examines each protrusion through a triangle and establishes that the protrusion cannot lead to a point of $S$ that is further than $e_p$ from the surface of $TP$.

There are two cases to consider. The first is when $c$, the center of the sphere, lies directly above a triangle through which it protrudes (in the sense that the closest point on the triangle to $c$ is in the interior of the triangle). The second case is when the closest point in the triangle to $c$ lies on an edge between two triangles (so the ball is protruding through the edge rather than the face). Figures 4 and 5 show the two cases.

*Case 1.* We want to find the deepest protrusion possible. This will be a "proof by gravity." We first rotate the polyhedron and sphere so that the triangle we are protruding through is horizontal and see what the sphere does if we let it fall. If the sphere touches no vertices, it will protrude further if it falls until it touches one. If it touches only one vertex, it will protrude deeper if it is allowed to fall further, pivoting at the vertex. This fall will continue until the ball touches a

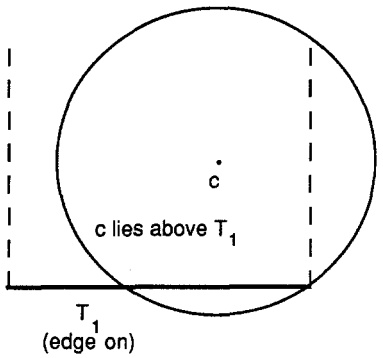


**Fig. 4.** Side view of ball and triangle $T_1$. Center of the ball lies above $T_1$.

second vertex or its center moves from above the triangle. If it touches a second vertex first, the ball will then pivot about the line between the vertices until it touches the third vertex or the center moves from above the triangle.

If the center moves from above the triangle in either case, we will have found a deeper Case 2 protrusion, so the Case 2 bound will work for this case also. This happens on obtuse triangles.

For acute triangles, the ball will be supported by the three vertices of the triangle and will have a spherical cap cut off by the plane of the triangle. From Lemma 1 we see that the protrusion will be greatest when the spherical cap has the largest possible radius. This is equivalent to finding the acute triangle (maximum edge length $d$) that has the largest circumcircle. This occurs when the triangle is equilateral, and the radius of the circle will be $s = d/\sqrt{3}$. Plugging this value for $s$ into the formula from Lemma 1 gives the formula for $e_p$.

*Case 2.* If the sphere is not touching two vertices, it will protrude deeper if it is moved so that it touches both endpoints $v_1$ and $v_2$ of the nearest edge to $c$ (see

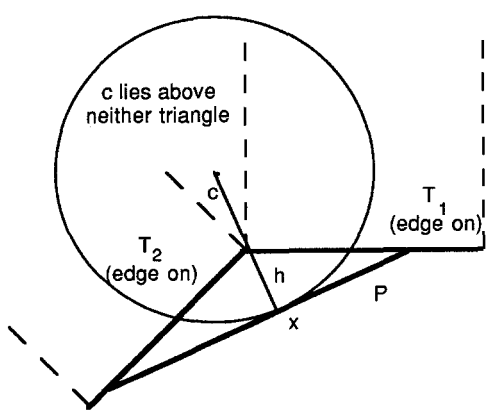


**Fig. 5.** Side view of ball and triangles $T_1$ and $T_2$. Center of the ball lies above neither triangle.

Figure 5). Let $x$ be the point of intersection between the sphere and the radius that is perpendicular to $v_1v_2$. We construct a plane $P$ tangent to the sphere at $x$. All points of the sphere lie to one side of $P$. Therefore the part of the sphere that protrudes through the triangles $T_1$ and $T_2$ does so within the triangular prism bounded by the plane $P$ and the planes containing triangles $T_1$ and $T_2$. But no point in that prism is further than $h$ from either the plane containing $T_1$ or the plane containing $T_2$, where $h$ is the distance from $x$ to edge $v_1v_2$. This distance can be computed from Lemma 1 by observing that the spherical cap cut off by a plane parallel to $P$ through $v_1$ and $v_2$ has $v_1v_2$ as its diameter. The length of $v_1v_2$ is at most $d$. Plugging $d/2$ in the formula for this case gives us a bound of

$$e_p = r - \sqrt{r^2 - d^2/4},$$

which is smaller than the bound from Case 1. □

In practice it is likely that the user will want to decide the allowable simulation error $e \leq e_p + e_s$ and the tool radius $r$, and then compute a value of $d$ which will guarantee that simulation error is less than $e$. We therefore want to solve for $d$ in terms of the user-defined values:

$$d = \sqrt{6re_p - 3e_p^2}. \tag{1}$$

When $e_p$ is small with respect to $r$ (which is usually the case), $d$ can be (conservatively) approximated as

$$d \sim \sqrt{6re_p}.$$

This means that the spacing between points grows as the square root of both the radius of the cutting tool and the simulation error allowed. The number of points is proportional to the inverse of the square of the distance between points. Therefore the number of points (and thus cutting time) grows linearly in both the inverse of the radius and the inverse of the desired accuracy.

To give some feel for what this means, assume a cutting tool with a radius of 1.0 and an allowable simulation error of 0.01. Then the spacing $d$ between points can be as large as 0.245. If the allowable simulation error is reduced by a factor of 100 to 0.0001, then the spacing between points is reduced by a factor of 10 to 0.0245.

*3.3. Computing Remaining Material.* Another type of error is the height of the material remaining above the plane of a triangle, which we call $e_r$. It is hard to get an absolute bound on $e_r$ by looking only at the height of the sample points. The most difficult case is a steep tower with a flat top. The angle between the walls and the top is nearly 90°. The cutter could approach the tower from the side and cut the entire edge of the flat top to the correct height by cutting with the side of the ball without cutting any material above the center of the flat top. All the edges would be exact, but the center would extend to the top of the stock.

If the flat top were a single triangle in our approximation (or even a group of triangles with all vertices on the edge), the error above the center of the triangle could be unbounded.

This situation can only occur when the angle between a triangle and some of the neighboring triangles is large. When a ball-end cutter cuts with its side its lowest point is far below the surface of the triangle. It avoids gouging neighboring triangles only if they drop off steeply. One approach that we considered was examining the angles between a triangle and each of its neighbors. We still consider this to be a promising approach and are trying to find a way to get an appropriate bound using this approach. We hope to get a bound that will work for the vast majority of triangles, and then to use other techniques for those few triangles where all neighboring triangles form a large angle with the given triangle.

The problem is simply that the cut height at the vertices gives too little information about what is happening over the center of a triangle. One way to get more information is to add a sample point in the interior of the triangle at the center of the largest inscribed circle (in-center).

To see how this helps, consider Figure 6. *RST* is a triangle and *P* is in its interior. Our goal is to cut *R*, *S*, *T*, and *P* to their correct height while leaving as much excess material as possible somewhere above *RST*. In other words, we want to find the placement of a sphere touching *P* and not containing any of *R*, *S*, or *T* in its interior that maximizes the vertical distance from a point of *RST* to the sphere.

Our first observation is that the maximum height will always occur at a vertex of *RST*. We call the circular projection of the sphere onto the plane of the triangle the "sphere's shadow." If one of the vertices does not lie within the sphere's shadow, then the height above that vertex is infinite so will be a maximum. If all three vertices lie within the sphere's shadow, then all points in the triangle lie with the shadow by convexity of the circle and the triangle. The height of the sphere above any line segment lying within its shadow is a unimodal function, with a single minimum. Therefore local maxima of this height function can only occur at endpoints of segments, never in its interior. Any point in the triangle except a vertex is contained in the interior of some segment lying in the triangle,

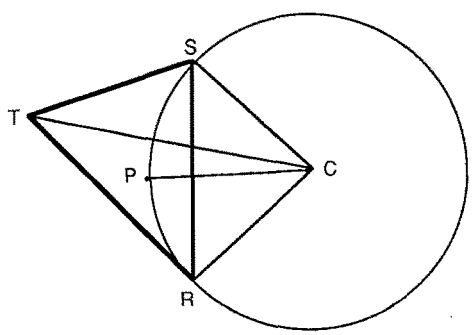


**Fig. 6.** Top view of ball and triangle with extra point in its interior.

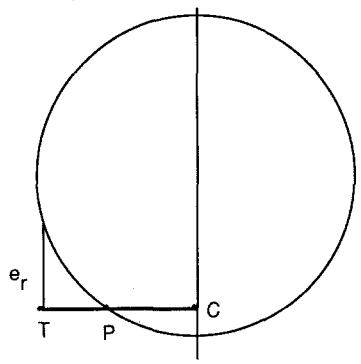


**Fig. 7.** Side view of ball and triangle showing the height of remaining material above $T$.

so cannot be a local maximum. This implies that the maximum height must occur at a vertex.

We therefore can assume without loss of generality that we want to cut $P$ to the correct height while leaving as much material as possible above $T$. To do this we start with the ball touching $P$, and then tip it away from $T$ until it touches $R$ and $S$. $C$ is the center of the circle passing through $R$, $S$, and $P$. We can compute the height of the ball above $T$ if we can compute the two distances $|CP|$ (the radius of the circle) and $|CT|$.

Figure 7 shows the side view. The plane of the triangle cuts off a spherical cap of the ball. The size of the circle $|CP|$ determines the height of the spherical cap as was shown in Lemma 1. As we move out from this circle toward $T$, the ball rises above the plane. Its height above the plane when it gets to $T$ is the difference in heights between spherical caps with radii $|CT|$ and $|CP|$. Applying Lemma 1, we get

$$e_r = \sqrt{r^2 - |CP|^2} - \sqrt{r^2 - |CT|^2}. \tag{2}$$

For a given triangle and a given interior point it is straightforward to compute these two lengths. However, our goal here is to get an upper bound on this height over all triangles.

We can increase $e_r$ in two ways. The first is to increase the difference between $|CT|$ and $|CP|$. The second is to increase $|CP|$, the size of the circle, so that we are out on a steeper part of the sphere. We increase $|CP|$ when we increase $|RS|$, when we move $P$ nearer to the perpendicular bisector of $RS$ while keeping it the same distance from $RS$, or when we reduce the distance from $P$ to $RS$.

The first three of these methods can only increase $e_r$ by a bounded amount. The difference between $|CT|$ and $|CP|$ can be at most $d$, the maximum edge length in the polyhedron. $|RS|$ can be at most $d$. $P$ can be moved so it sits on the perpendicular bisector to $RS$. However, we can put $P$ arbitrarily close to $RS$, thus making the circle arbitrarily large. If the radius of the circle becomes larger than $r$, the radius of the ball, there will be no bound on the height of the material above $T$. (The ball can cut $P$ from the side without touching $R$ or $S$.)

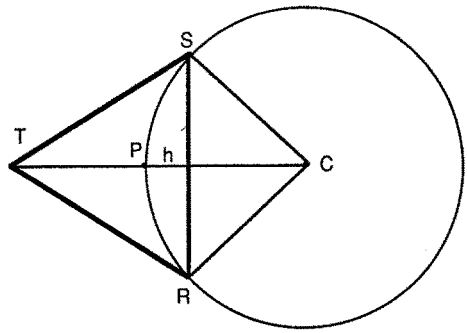


**Fig. 8.** Worst-case triangle for remaining material above $T$.

This implies that we should choose $P$ so it is far from any edge. The in-center is by definition the point farthest from its nearest edge. However, by making skinny triangles we can still force $P$ arbitrarily close to $RS$. Therefore any bound we derive will have to take into account the "fatness" of the triangle. The "fatness" parameter that we use is

$$s = \frac{\text{(distance of } P \text{ from nearest edge)}}{\text{(length of longest edge)}}.$$

Figure 8 shows the worst possible triangle of fatness $s$, obtained by maximizing all factors simultaneously. $|RS| = d$, $T$ is distance $d$ away from $|RS|$, and $P$ is distance $h = sd$ from $RS$ and is on its perpendicular bisector. In fact, this triangle cannot occur in our polyhedron, because edges $RT$ and $RS$ are longer than $d$. (There is a tradeoff between maximizing $|RS|$ and maximizing the distance of $T$ from $RS$.) However, a bound for this triangle is certainly worse than a bound for any acceptable triangle of fatness $s$.

Some geometry and trigonometry give us the following edge lengths:

$$|CQ| = (d^2 - 4h^2)/8h,$$
$$|CP| = (d^2 + 4h^2)/8h = (1 + 4s^2)d/(8s) = k_1 d,$$
$$|CT| = (d^2 - 4h^2)/8h + d = (1 - 4s^2 + 8s)d/(8s) = k_2 d.$$

The latter two lines substitute $sd$ for $h$ and then implicitly define the constants $k_1$ and $k_2$. Plugging into the formula for $e_r$ derived above gives

$$e_r = \sqrt{r^2 - k_1^2 d^2} - \sqrt{r^2 - k_2^2 d^2}.$$

Solving for $d$ in this formula gives

$$d = \sqrt{\frac{-e_r^2(k_1^2 + k_2^2) + 2e_r\sqrt{k_1^2 k_2^2 e_r^2 + (k_1^2 - k_2^2)^2 r^2}}{(k_1^2 - k_2^2)^2}}.$$

For $e'$ small relative to $r$, this simplifies to

$$d \sim \sqrt{(2e_r r)/|k_1^2 - k_2^2|} = \sqrt{Ce_r r} \tag{3}$$

for

$$C = 8s/(1+4s-4s^2).$$

We noted that this triangle cannot ever occur. The worst acceptable triangle appears to be similar to the one in Figure 8, but with the sides $RT$ and $ST$ reduced to length $d$. The same calculations for this triangle give

$$C = 16s/(4s+\sqrt{3}-4\sqrt{3}s^2).$$

($C = 2$ for an equilateral triangle, and gets smaller as $s$ decreases.) This growth rate has the same form as the gouging case, but the constant is worse.

*3.4. Combining the Errors To Obtain a Bound on Simulation Error.* We have analyzed three types of errors: the separation distance $e_s$, the protrusion distance $e_p$, and remaining excess material height $e_r$. However, our goal is to find the overall simulation error introduced by a given set of points. We break this analysis into two parts. The first is $e_i$, the amount that a cut point can lie inside the true surface even if all selected points are cut exactly. The second is $e_o$, the amount that a cut point can lie outside the true surface even if all selected points are cut exactly.

The maximum distance that the ball can protrude inside the surface is $e_s + e_p$, which occurs when the tool protrudes inside a triangle of the polyhedron and the surface lies outside the triangle, as shown in Figure 9(a). However, we can sometimes do better than this. If the surface lies completely inside the triangle of the polyhedron, then the protrusion distance and the separation distance can cancel, as shown in Figures 9(b) and (c). They certainly cannot combine. Therefore when we can prove that we are in this case, the appropriate formula is $\max(e_s, e_p)$. Therefore the error introduced by a given triangle is

$$e_i \leq \begin{cases} \max(e_s, e_p) & \text{if the surface lies entirely inside the triangle,} \\ e_s + e_p & \text{otherwise.} \end{cases} \tag{4}$$

The bounds for $e_o$ are symmetric:

$$e_o \leq \begin{cases} \max(e_s, e_r) & \text{if the surface lies entirely outside the triangle,} \\ e_s + e_r & \text{otherwise.} \end{cases} \tag{5}$$

**4. Method for Selecting Points.** The user chooses bounds $b_i$ and $b_o$ as the maximum acceptable values of the errors $e_i$ and $e_o$. We want to select our points in a way that guarantees that $e_i \leq b_i$ and $e_o \leq b_o$ for all triangles. Our general

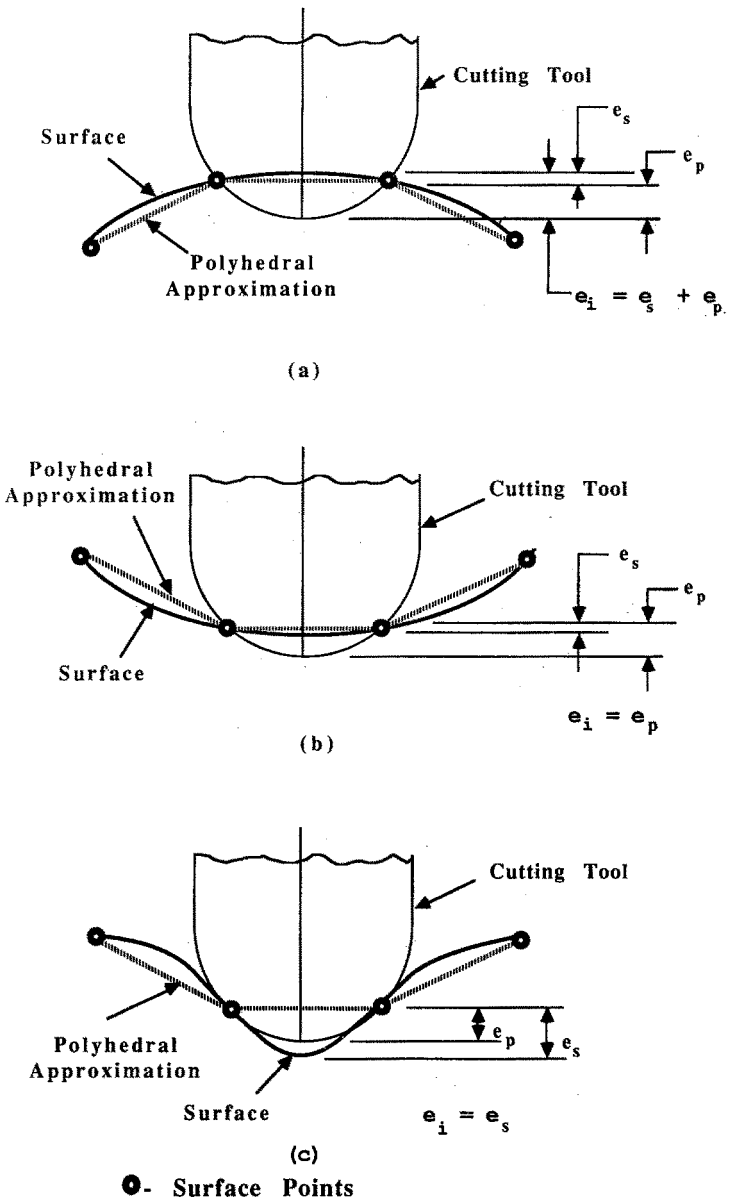


**Fig. 9.** Ways that $e_p$ and $e_s$ can interact to get $e_i$.

approach is to generate a triangulated polyhedron, all of whose vertices are on the surface. We then subdivide triangles into smaller triangles (again with all vertices on the surface) until the triangulation is good enough to guarantee the bounds using formulas (4) and (5). The vertices of the triangulation become our set of points. The triangles themselves play no part in the simulation (although

they are used to display the surface graphically). We first describe a method that guarantees that $e_i \leq b_i$ and $e_o \leq b_o$. We then describe a method that uses many fewer points for the special case where we only guarantee that $e_i \leq b_i$.

*4.1. Guarantee Both Protrusion and Excess Material Bounds.* The analyses of both protrusion and excess material express the error in terms of the longest side of the triangle. In both cases the closer our triangles are to equilateral, the fewer points we will need to achieve given bounds on protrusion and excess material. (This is especially true for the excess material bounds, because of the "fatness" parameter.) Therefore we want our initial triangulation to consist of nearly equilateral triangles, and ideally each triangle will just barely pass the appropriate error test. For those that do not, we want a subdivision method which breaks nearly equilateral triangles into smaller triangles that are still nearly equilateral.

How long should the edges of the nearly equilateral triangles in the initial triangulated polyhedron be? Formulas (1) and (3) can be used to get an estimate on an edge length $d$ that will guarantee that both $e_p \leq b_i$ and $e_r \leq b_o$. (Formula (1) is easy to use, but formula (3) requires an estimate of the fatness parameter. However, if we are optimistic and simply use $C = 2$, no real harm is done. The program will later subdivide triangles that are not small enough.) By choosing an edge length a bit shorter than this $d$, most of the triangles in relatively flat areas will need no further subdivision (because $e_s$ will be small).

The triangulation is actually generated in parameter space and then mapped into object space to avoid having to invert the mapping from parameter space to object space. A unit square in parameter space is mapped onto a bounded surface patch in object space. We initially triangulate each patch so that the triangles in object space will all be the same size and nearly equilateral. As illustrated in Figure 10, we start by dividing each patch's parameter space into strips. Each strip is initially triangulated so that the object-space triangles are as close to equilateral as possible and have sides less than 95% of the $d$ calculated from formulas (1) and (3). To avoid creating excess points each strip is initially triangulated so that the vertices of triangles on adjacent strips coincide.

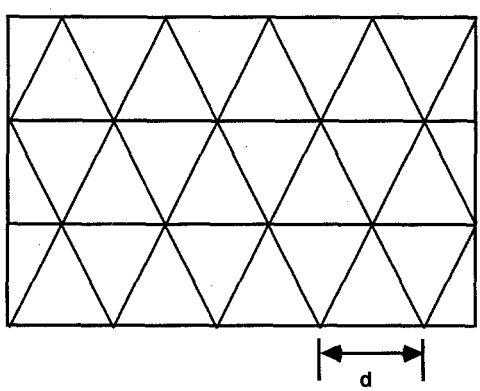


**Fig. 10.** Surface is triangulated by dividing the patch into strips of equal-sized triangles.

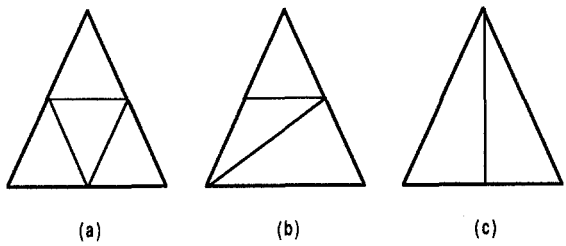


**Fig. 11.** Subdivision of triangles by subdividing three sides (a), two sides (b), and one side (c).

To perform the subdivision described in the preceding paragraph we estimate the width and length of the patch in cartesian space (by measuring the distances between the corners of the patch) and use this information to decide how many strips are needed and how many triangles can fit within each strip. We then divide the patch into that many equal-sized strips, and divide each strip into the right number of equal-sized triangles.

For each triangle we apply a testing routine to see if it meets the error bound. This routine first tests to see if the triangle satisfies error bounds in formulas (4) and (5). If it does, then the routine adds a point at the in-center of the triangle (needed for the excess material bound to work) and returns.

If the triangle does not meet its error bound, then it must be subdivided. Do this by barycentrically subdividing it (connect the midpoints of the three sides in parameter space as shown in Figure 11(a)). The barycentric subdivision divides one triangle into four similar triangles of half the size, so it meets the requirement that nearly equilateral triangles be divided into smaller triangles that are still nearly equilateral. Then recursively apply the testing routine to each of the four subtriangles to see if it meets the error bound (and subdivide it if it does not). When the procedure finishes its recursive calls, all triangles meet the error bounds.

This method of doing all subdivisions in parameter space is computationally easy to perform, but it depends strongly on the fact that equal-sized triangles in parameter space usually map into equal-sized triangles in cartesian space. Automobile body parts appear to meet this requirement, so it works well for our specific application. If the surfaces were not as well behaved an iterative method might be required to obtain the nearly equilateral triangles in the initial subdivision and to make the barycentric subdivision divide a triangle into four approximately equal-sized subtriangles.

*4.2. Guaranteeing Only Protrusion Bounds.* Guaranteeing excess material bounds using our current analysis requires many more points than guaranteeing protrusion bounds. The $d$ value computed by formula (1) is at least $\sqrt{3}$ larger than the $d$ value computed by formula (3), if $b_i = b_o$. Reducing triangle size by this amount approximately triples the number of triangles. Because most points are shared by six triangles while only three points appear in each triangle, the number of triangles is about double the number of points. This means that adding a point to the center of each triangle approximately triples the number of points.

Therefore we can get by with fewer points if we only guarantee $e_i$ and have no provable bound on $e_o$.

There is the potential for further savings. When the object is a cylinder or a similar shape with high curvature in one direction and low curvature in the other direction, triangles that are long in the direction of low curvature but narrow in the direction of high curvature can approximate the surface very closely. Fewer of these long, narrow triangles are required to closely approximate the surface than of the nearly equilateral triangles produced by barycentric subdivision.

Because the bound on $e_r$ depends on a fatness parameter, breaking up a fat triangle into long, skinny triangles with the same maximum edge length as the original can cause $e_r$ to increase. Therefore we had to keep the triangles nearly equilateral. However, splitting a fat triangle into long, skinny triangles with the same maximum edge length as the original does not increase $e_p$. This observation leads to the following modifications of the subdivision algorithm:

(1) Generate the original triangulation as described in Section 4.1, but use only formula (1) to determine $d$.
(2) For the recursive error bounds test routine, test the triangle to see if it passes the error bounds test in (4). If it passes, then the routine returns.

Otherwise we want to determine if the problem is with $e_p$ or with $e_s$. If $e_p$ is larger than $b_i$ or only slightly smaller, then $e_p$ is the problem and we barycentrically subdivide the triangle as before. Otherwise we are trying to reduce $e_s$, and dividing all three sides may not be necessary.

Generate the (parametric) midpoint of each edge and find the distance from that midpoint to the surface. If that distance plus $e_p$ is smaller than $b_i$ then that edge is not a problem. Subdivide only those edges that fail this test. The subdivisions for three, two, and one edges are shown in Figure 11. (If all three edges pass the test while the triangle as a whole fails, then it is not clear which edge is the problem. Barycentrically subdivide in this case.) Recursively apply this error bounds test to each subtriangle.

The amount that nonbarycentric subdivision reduces the number of points required to simulate milling depends on the shape of the object. For a trunk lid example discussed in the next section, it reduced the number of points by a factor of two.

This analysis shows that it would be extremely useful to find a method for bounding excess material that did not add extra points in the in-center of triangles and did not require fat triangles.

**5. Current Cutting-Simulation System.** Our current cutting simulation implements the system described in Section 4.2. We decided that the benefit of a guaranteed bound on excess material was not worth the extra run time. If that judgment changed, it would not be difficult to modify the program to follow the algorithm in Section 4.1.

We do not analytically calculate $e_s$. Ford's surface representation scheme is both very complex and proprietary. Therefore we did not consider it worth the

**Table 1.** Cutting simulation system performance for four test cases. Results show the number of points and the CPU minutes on a SUN III/160 workstation for four different user-specified accuracies.

| | | Tool movements | | | Maximum simulation error in mm (in.) | | | |
|---|---|---|---|---|---|---|---|---|
| | Patches | Parabolic | Linear | | 2.5 (0.1) | 0.75 (0.029) | 0.33 (0.013) | 0.25 (0.01) |
| Trunk | 20 | 3,500 | 18,855 | Points | 2,301 | 8,315 | 16,667 | 23,824 |
| | | | | Minutes | 4.1 | 11.0 | 19.2 | 28.1 |
| Bumper | 263 | 8,100 | 45,411 | Points | 7,172 | 17,363 | 34,815 | 46,871 |
| | | | | Minutes | 12.0 | 21.4 | 39.6 | 49.7 |
| Handle | 100 | 4,500 | 27,883 | Points | 1,889 | 4,518 | 9,091 | 11,177 |
| | | | | Minutes | 4.7 | 7.5 | 12.6 | 15.0 |
| Hood | 24 | 700 | 5,765 | Points | 956 | 3,550 | 7,421 | 10,026 |
| | | | | Minutes | 3.5 | 9.9 | 17.2 | 23.2 |

effort to derive a guaranteed bound. We chose to implement a heuristic based on a sampling of points. This technique has proven adequate for the well-behaved surfaces used in automotive styling but may not be generally applicable. We test the perpendicular distance from the triangle to the surface at the midpoint of each edge and at a point in the center of the triangle (where the midpoints and center were computed in parameter space). The maximum of these distances is our estimate for $e_s$. If all the points lie above or below the triangle, we decide that the surface lies above or below the triangle for the purpose of the error bounds test. Despite the fact that these tests do not guarantee that the surface is really close to the triangle they appear to be a good heuristic. In fact, for our sample surfaces we very rarely found a case where including the point in the center of the triangle caused a different result for an error bounds test than just using the midpoints of the edges. We therefore slightly reduced our run time by eliminating the center test.

We tested the system on actual NC program files supplied by the Ford Motor Company. Table 1 shows results for four simulations done with different accuracy requirements. The four files cut the dies for a trunk lid, a bumper, a door handle, and a hood. The maximum possible simulation errors shown are for worst-case gouging errors using a ball-end tool. The radii of the tools were determined by the tool sizes requested in the files: 50 mm for the trunk, 10 mm for the bumper, 4 mm for the handle, and 38 mm for the hood. The time to generate the surface points was generally 10–20% of the total time required for the simulation. The table shows the number of surface patches in the design file, the number of parabolic tool movements in the file, and the number of linear tool movements derived from the parabolic movements. The linearization of parabolic tool movements was accurate to within 0.0254 mm (0.001 inches).

**6. Further Modifications: Deal with Triangles Rather than Points.** The approach above keeps track of only points. We generated triangles in the process of

subdividing the surface, but we do not use them (although we will keep triangles in order to graphically display the surface). This section explores ways of using the triangle information to reduce the number of points chosen and speed up the algorithm. We are planning to implement these methods to see if they are faster than the ones described above.

*6.1. An Alternate Way To Find Excess Material.* The approach described above for detecting excess material has its drawbacks. First, the $d$ value needed to detect this type of error is smaller than the $d$ value needed to detect gouging for the same allowed error. Second, the number of center points added to triangles is large. This may prove to be an expensive way to test for a type of error that is probably less frequent than gouging. (It can only come from miscomputing or missing a tool path, not from interference from other patches.) Points from adjacent triangles should help, but we are still trying to find how to use them effectively.

Therefore we propose a different approach. For each triangle in the approximation we will find either a small number of cutting paths that remove all excess material above that triangle or else a point above which excess material remains. Either way we will know for certain if the triangle has excess material remaining above it.

Our method depends on the fact that after the simulation we know both the *Zcut* height at each sample point and the cutting-tool movement that cut that sample point most deeply. If any vertex of a triangle has a *Zcut* height which is too high, that point is out of tolerance. If not, we will test the interior of the triangle. The three tool movements that cut the three vertices of the triangle most deeply probably remove all excess material above the triangle. We test this by direct computation. (The test involves finding the intersection of the three tool movements with a translated copy of the triangle lifted above the original triangle by a distance equal to the allowed error. Often a single movement most deeply cuts more than one vertex of the triangle. In this case fewer than three intersections need to be done.)

If the three cuts eliminate all excess material, the test is done. If they do not, that triangle may still be in tolerance. A cut which passed over the center of the triangle but was not the deepest cut at any vertex may have eliminated the apparent excess material. To eliminate these "false positives," we generate a new point in the area that appears to have excess material remaining. The new points generated by all out-of-tolerance triangles become a new sample of points and the cutting simulation is run on just these points. If any point is out of tolerance, so is its triangle. If not, a new cut has been discovered, and it can be added to the set of movements that removed material from the triangle. If this new cut eliminates the remaining excess material, we are done. Otherwise the process is repeated. Since we repeat only when we discover a cut not in our set of movements that removed material from the triangle, the process terminates when we run out of tool paths over the triangle.

Note that during the simulation of tool movements we are dealing only with points. It is only when that part is done that we must deal with the more complex

intersections of cutting paths with triangles. Furthermore, intersecting with triangles will be much faster than intersecting with the exact representation of the surface.

*6.2. An Alternate Way To Detect Gouging.* Large triangles can accurately approximate flat surfaces. However, to detect gouging we must make the triangles fairly small. It would be convenient if the larger triangles could be used.

The method suggested above for excess material detection is not adequate for detecting gouging. This is because of a basic asymmetry between detecting excess material and detecting gouging. If excess material is removed by a cut, the material is gone. Subsequent cuts cannot restore the excess material. Therefore when we find a set of cuts that eliminates all excess material, we are done. On the other hand, when a surface is ungouged, any subsequent cut may gouge it. Therefore all cuts above a surface must be tested to guarantee that there is no gouging.

However, a modification of the excess material approach will work and may help. In the recursive subdivision step, split a triangle only if it is too far from the surface (i.e., test for $e_s$ and ignore $e_p$). Instead of remembering only the points, remember the triangles also. When the tool passes over a triangle, "cut" the triangle by finding the deepest penetration of the tool below its surface. By remembering this information for each triangle, we can keep track of gouges even for huge triangles. (We need only save the depth of the gouge and the tool movement number, not the location where the gouge occurred.) The code for intersecting the tool and the triangle is more complex than for computing the $z$ height of a point, so each individual computation would be more expensive. However, if the triangles are large enough the time savings may be worth the effort. (In theory it would be better still to intersect the tool with the actual mathematical surface, but this is a much more difficult computation than intersecting the tool with a triangle.)

## 7. Other Modifications

*7.1. Improved Bucketing Strategy.* The method of "bucketing" for determining which points to examine has proven to be quite efficient, but even larger savings are possible. A ball-end mill making final cuts will usually only be cutting with a small portion of its surface. Therefore it will pass over a number of points without actually cutting them. To avoid examining buckets where no *Zcut* values will change it is possible to store for each bucket the maximum *Zcut* value associated with any point in the bucket. Given the borders of the bucket and the cutting tool path, we can compute the minimum $z$ height of any part of the tool over the bucket. If this minimum $z$ height is higher than the maximum $z$ value in the bucket, no points in the bucket need be examined. Otherwise the points in the bucket should be cut and a new maximum computed. In areas where the buckets are small and the point density is high, this could save a lot of computation. In areas where the point density is low, it might be faster to test the points than compute the minimum $z$ height over the bucket. The choice could be made depending on the number of points in the bucket.

*7.2. Simulating Flat-end and Torodial-end Cutting Tools.* The error analysis is quite dependent on the shape of the ball-end mill. With a flat-end mill (a cylinder), no upper bound on the amount of excess material left is possible if we examine only the heights of points in the sample. This is because the mill has a sharp edge and vertical sides. No matter how close together the sample points are on a perfectly flat surface, it is possible to cut into each corner of some triangle from the outside, leaving an arbitrarily high spike in the middle. (We can guarantee that no such spike is thicker than $d$, the maximum point separation.) However, our alternate method would work as well for flat-end cutters as for ball-end ones.

The amount of undercutting can at least be bounded, but the bound is not as good as for a ball-end mill. The problem is again the sharp edge of the mill. If the mill approaches a long, skinny triangle that is slanted at a 45° angle, it can cut depth $d/2$ from the surface of the triangle without gouging any vertices (see Figure 12). Therefore in this case $e = d/2$ and $d = 2e$. This implies that the number of points grows quadratically as the inverse of the desired error, which is much worse than linear growth.

A toroidal-end (also called fillet-end) cutting tool is a cylinder with a rounded edge. Two parameters are needed to specify such a cutter shape, the cylindrical radius $R$ and the edge radius $r$. The amount of gouging and excess material is a function of both radii, but the worst case is when $R = r$. In this case the mill is simply a ball-end mill! From Theorem 1, we saw that the error can be at most

$$e_p = r - \sqrt{r^2 - d^2/3}.$$

Increasing $R$ reduces the curvature in one direction, which reduces the amount that the tool can protrude through a triangle. A similar argument shows that increasing the radius of $R$ decreases the bound on undetected excess material. This means that the bounds derived above for the ball-end mill apply directly to the toroidal-end mill, as long as we use the edge radius $r$ in those formulas.

In fact, this estimate is fairly tight. As the ratio $R/r$ approaches infinity the rounded edge looks locally like the side of a cylinder. The argument in Case 2

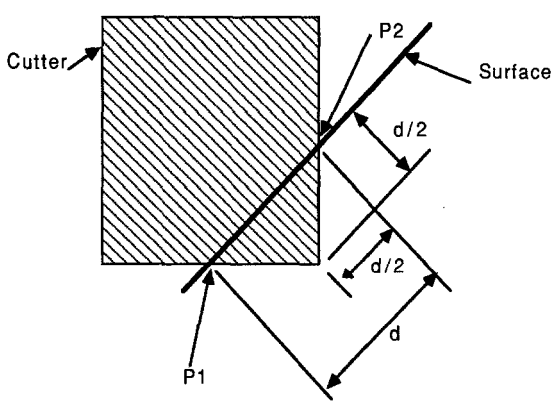


**Fig. 12.** Maximum error due to protrusion of a cylindrical cutter through a triangle.

of the proof of Theorem 1 shows that the side of a cylinder can cause a gouge as deep as

$$e_p = r - \sqrt{r^2 - d^2/4}.$$

This means that changing the value of $R$ can at most decrease the $d^2/3$ in the formula for $e_p$ to $d^2/4$.

*7.3. Parallel Algorithms.* Our technique is still quite CPU intensive. However, the general approach lends itself very nicely to parallel processing. This is because the sample points can be processed independently in arbitrary order. In particular, if each point had its own processor an entire cutting tool movement could be performed in constant time. If material removal rate is not being computed, the movements in the CL file can also be processed in arbitrary order. Parallelism can be used in generating the set of points as well as in running the simulation. (Each of the triangles to be tested for subdivision could be assigned a different processor.) Bucketing was added for efficiency and would become unnecessary if every point had its own processor.

With some additional effort the true normal could be used for each point instead of the $z$ distance. The tool-movement-envelope calculation would be only slightly more complicated than the current method if we are dealing with 3-axis machining. The standard $Z$ buffer would no longer be adequate because a vector could enter and leave the object several times. However, Wang's extended $Z$ buffer would solve this problem [WW].

**8. Future Work.** Our first areas of work are the obvious ones of exploring more efficient simulation techniques, better point-selection techniques, and better bounds. We want to find where our current approach can be improved and make it better. In particular, we want to find a better method for bounding the amount of excess material that uses adjacent triangles rather than additional points. We also want to explore ways of reducing the amount that we overestimate errors due to the fact that we measure distances in the $z$ direction rather than normal to the surface.

Next, we would like to extend our method to handle 5-axis machining. A major drawback to our approach is that it can only handle surfaces which can be oriented so that the surface is a function in $x$ and $y$. That is, there can only be a single $z$ height for any $(x, y)$ coordinate. For 3-axis machining this is not usually a problem, because this is normally the only type of surface that a 3-axis mill can cut. However, 5-axis machines can mill ledges, horizontal holes, and other shapes which cause the cut object to have multiple $z$ values for a single $(x, y)$ value.

There are three basic difficulties with extending our method so that it can handle 5-axis machines. The first is that a simple $Z$ buffer is no longer sufficient, because a vector can enter and leave the object several times. Fortunately, Wang's extended $Z$ buffer is designed to handle this problem [WW].

The second difficulty is that a 5-axis milling tool sweeps out a much more complex envelope than a 3-axis mill, because its orientation can change as it

moves. We need to find the intersection of a vector with this more complex envelope. Wang's method of approximating the volume by a polyhedron is one approach, but we would like to consider other approaches as well.

The third problem is localization. Our bucketing strategy depends on all vectors being parallel. We can continue to use this strategy if we use Wang's extended $Z$ buffer. However, our simulation would be more accurate if we were able to use normal vectors instead of having all vectors point in a single direction. In this case our bucketing strategy then breaks down completely. (Note that for a parallel 5-axis simulation this is not a problem, because we no longer need to localize.)

One approach that we are considering is to use "approximate normals." Instead of having one extended $Z$ buffer with a single $Z$ direction, we could have a number of buffers. Each would choose a different direction as its $Z$ direction. For example, we could use three extended $Z$ buffers, one with its vectors parallel to the $X$ axis, one with its vectors parallel to the $Y$ axis, and a third with its vectors parallel to the $Z$ axis. A point would be placed into a bucket in only one of the buffers. The one chosen would be the one whose $Z$ direction was closest to the normal direction for the point. By increasing the number of extended $Z$ buffers, we could guarantee that the angle between the true normal and the "approximate normal" used by its extended $Z$ buffer was small.

**Acknowledgments.** John Magewick and others at Ford Motor Company provided valuable suggestions and supplied examples of Ford NC programs for us to use as test data. A referee for this paper made a number of useful suggestions and pointed out previous work in the area.

## References

[ACd] G. T. Armstrong, G. C. Carey, and A. dePennington, Numerical Code Generation from a Geometric Modeling System, in *Solid Modeling by Computers: From Theory to Applications*, M. S. Pickett and J. W. Boyse, eds., Plenum, New York, 1984.

[AEF] P. Atherton, C. Earl, and C. Fred, A Graphical Simulation System for Dynamic 5-Axis NC Verification, Autofact Show of the Society of Manufacturing Engineers, Detroit, November 1987, 2-1-2-12.

[An] R. O. Anderson, Detecting and Eliminating Collisions in NC Machining, *Computer-Aided Design*, **10**(4) (1978), 231-237.

[Ar] G. T. Armstrong, A Study of Automatic Generation of Non-Invasive Machine Paths from Geometric Models, Ph.D. Dissertation, University of Leeds, October 1982.

[B] P. Bézier, *Numerical Control—Mathematics and Applications*, Wiley, London, 1972.

[BFK] W. Böhm, G. Farin, and J. Kahmann, A Survey of Curve and Surface Methods in CAGD, *Computer-Aided Geometric Design*, **1** (1984), 1-60.

[Ch] I. T. Chappel, The Use of Vectors to Simulate Material Removed by Numerically Controlled Milling, *Computer-Aided Design*, **15**(3) (1983), 156-158.

[Co] S. A. Coons, Surfaces for Computer-Aided Design of Space Forms, Tech. Rep. MAC-TR-41, MIT, Cambridge, MA, June 1967.

[DJ] R. L. Drysdale and R. B. Jerard, Discrete Simulation of NC Machining, *Proceedings of the Third Annual ACM Symposium on Computational Geometry*, June 1987, 126-135.

[DM] J. P. Duncan and S. G. Mair, *Sculptured Surfaces in Engineering and Medicine*, Cambridge University Press, Cambridge, 1983.

[F] A. R. Forrest, On the Rendering of Surfaces, *ACM SIGGRAPH*, **13** (1979), 253–259.

[FCDZ] R. Fridshal, K. P. Cheng, D. Duncan, and W. Zucker, Numerical Control Part Program Verification System, *Proceedings of Conference on CAD/CAM Technology in Mechanical Engineering*, MIT, March 1982, MIT Press, Cambridge, MA, 236–254.

[FP] I. D. Faux and M. J. Pratt, *Computational Geometry for Design and Manufacture*, Ellis Horwood, Chichester, 1979.

[FV] J. D. Foley and A. VanDam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.

[G] D. D. Grossman, Opportunities for Research on Numerical Control Machining, *Communications of the ACM*, **29**(6) (1986), 515–522.

[GR] W. J. Gordon and R. F. Riesenfeld, B-Spline Curves and Surfaces, in *Computer-Aided Geometric Design*, R. Barnhill and R. Riesenfeld, eds., Academic Press, New York, 1974, 95–126.

[GZ] M. P. Groover and E. W. Zimmers, *CAD/CAM: Computer-Aided Design and Manufacturing*, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[HV] W. A. Hunt and H. B. Voelcker, An Exploratory Study of Automatic Verification of Programs for Numerically Controlled Machine Tools, Production Automation Project Tech. Memo. No. 34, University of Rochester, January 1982.

[JDH] R. B. Jerard, R. L. Drysdale, and K. Hauck, The Use of Computer Graphics as a Tool for Detecting Errors in Numerical Control Machining of Sculptured Surfaces, *Proceedings of NCGA's Computer Graphics '87*, March 1987, 290–299.

[JDHSM] R. B. Jerard, R. L. Drysdale, K. Hauck, B. Schaudt, and J. Magewick, Methods for Detecting Errors in Numerically Controlled Machining of Sculptured Surfaces, *IEEE Computer Graphics and Applications*, to appear.

[JHD] R. B. Jerard, K. Hauck, and R. L. Drysdale, Simulation of Numerical Control Machining of Sculptured Surfaces, Paper no. 86057, 15th International Symposium on Automotive Technology and Automation (ISATA), Flims, Switzerland, October 6–10, 1986.

[KMS] B. C. Kuttner, D. S. Majcher, and P. B. Snedecor, Systematic Processing: An Approach to Fully Automatic NC Tool Path Generation, *Proceedings of Autofact '85 Conference*, SME, November 1985, 18-7-18-22.

[M] A. E. Middleditch, Survey of Numerical Controller Technology, Production Automation Project TR-1-D, University of Rochester, August 1973.

[O] J. H. Oliver, Graphical Verification of Numerically Controlled Milling Programs for Sculptured Surface Parts, Ph.D. Thesis, Michigan State University, 1986.

[OG] J. H. Oliver and E. D. Goodman, Color Graphic Verification of NC Milling Programs for Sculptured Surfaces, *Proceedings of the 10th Annual Automotive Computer Graphics Conference and Exposition*, Engineering Society of Detroit, December 1985.

[RA] D. F. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics*, McGraw-Hill, New York, 1976.

[SJW] S. M. Staley, R. B. Jerard, and P. R. White, Computer-Aided Design of Curved Surfaces with Automatic Model Generation, *Trans. ASME J. Mech. Design*, **104**(4) (1982), 817–824.

[T] D. L. Toth, On Ray Tracing Parametric Surfaces, *ACM SIGGRAPH*, **19**(3) (1985), 171–179.

[V] T. Van Hook, Real-Time Shaded NC Milling Display, *ACM SIGGRAPH*, **20**(4) (1986), 15–20.

[VH] H. B. Voelcker and W. A. Hunt, The Role of Solid Modeling in Machining—Process Modeling and NC Verification, SAE Technical Paper 810195, 1981.

[WW] W. P. Wang and K. K. Wang, Geometric Modeling for Swept Volume of Moving Solids, *IEEE Computer Graphics and Applications*, **6**(12) (1986), 8–17.

[ZB] D. Zhang and A. Bowyer, CSG Set-Theoretic Solid Modeling and NC Machining of Blend Surfaces, *Proceedings of the 2nd Annual ACM Conference on Computational Geometry*, June 1986, 236–245.