REAL-TIME SHADED NC MILLING DISPLAY

TIM VAN HOOK

Trancept Systems Inc.
521F Uwharrie Court
Raleigh, NC 27606

Abstract

The real-time shaded display of a solid model being milled by a cutting tool following an NC path is attained by the image-space Boolean subtraction of solid objects. The technique is suitable for implementation in microcode in a raster graphic display processor. Update rates of 10 cutting operations per second are typical.

Problem

The integration of computer aided design (CAD) and computer aided manufacturing (CAM) into computer integrated manufacturing (CIM) places increasing emphasis on the design of tools and paths for numerically controlled (NC) milling machines.[1] An NC path consists of a list of tool positions and orientations which directs a milling machine to cut the desired part out of a block of material. In CIM, NC paths are generated from solid models designed with CAD systems. The path must be verified for gouging, clamp interference, speeds, feeds, depths, elapsed time, as well as whether the part is actually produced as designed. As path generation becomes more automated and less the responsibility of skilled human parts programmers, the need for verification increases.

Shaded computer graphics is central to the design of solid models, but NC paths are commonly displayed as line drawings. Line drawings don't completely describe the three-dimensional solid being milled (figure 1). Since the milling operation is equivalent to a Boolean (logical) subtraction of the tool solid from the block solid for each position of the tool along the path, constructive solid geometry (CSG) modelers, such as MAGI Synthavision, have been used to generate shaded images of NC milling. Ray tracing is typically used as a display technique for CSG models. A ray for each pixel intersects each solid object in the scene and generates a line segment. These line segments are bounded by the points at which the ray enters and exits the objects. Boolean operations, such as subtraction for milling, can then be simply performed on these one-dimensional line segments.[2] However, in contrast to a CSG solid model, which might consist of dozens of Boolean operations between primitive solids, a typical NC path contains many hundreds of tool positions. Execution times for that number of intersection calculations are many hours on mainframe computers, especially if a continuous animation sequence of the part being milled is generated. The display of NC milling can also be implemented by the calculation of the surface intersections of the block with each position of the tool or with the swept volume of the tool path, but the number of calculations is not less than by ray tracing.

Because of the time and expense of graphical simulation, NC paths are commonly verified on an actual milling machine, using a material softer than the final product. The expense and hazard of testing on a costly production machine are undesirable, especially during the early debugging of a path or a path generation routine.

Background

The Z buffer is the most common commercial approach to the interactive shaded display of solid models. A Z buffer is an extended frame buffer. A frame buffer consists of a matrix of memory locations which refresh a raster display. Each memory location, commonly called a pixel, contains a color number. In addition to a matrix of color numbers, a Z buffer contains a corresponding matrix of Z or depth values of the nearest or visible surface at each pixel. [3] Z buffered visible surface processing is simple. Each surface element is converted into pixels with a color and a Z value, often by stepping through each X pixel for each Y line (scan conversion). Each of these pixels is conditionally written to the Z buffer if its Z value is nearer the viewpoint than the Z value of the corresponding pixel in the Z buffer. The simplicity of Z buffered visible surface processing is suitable for bit-slice microcode or hardware implementation. Because the Z buffer is basically a large memory, it benefits greatly from the continuing increases in semiconductor memory density and decreases in price.

Several graphics techniques have extended the Z buffer to include more information than depth and color at each pixel. Lucasfilm [4] maintains a list of the subpixel fragments of surfaces at each pixel for use in anti-aliasing calculations. Soft shadows are generated from a list of potentially shadowing objects at each pixel. [5] An extended depth buffer stores both a near and far depth at each pixel with a hole count and primitive identifiers for CSG modeling in the TIPS system. [6] Atherton [7] described an object buffer matrix of depth vectors for cutaway and section views, translucency, and shadows.

In these and other applications, the Z buffer is generalized from a display memory and visible surface method, to a spatial data structure which is indexed by the X and Y pixel address, and contains graphic, spatial, and modeling information. The use of spatial data structures is more common in voxel (volume cell) representations, which are often organized in octrees. [8] The ability to access model geometry with an index of a spatial location can simplify both display and intersection calculations, and polygonal data as well as solid volume elements can be organized in octree structures. [9] Ray tracing intersection calculations can be reduced by selecting probable objects to intersect from an octree structure indexed by the position of the ray. [10]

Algorithm

Real-time shaded NC milling display is accomplished with an extended Z buffer data structure. Each pixel is a rectangular solid extending along the Z axis, and consists of a near Z depth, a far Z depth, a color, and a pointer. In keeping with the convention of the names pixel and voxel, this rectangular solid is called a dexel, or depth element. The dexels are organized in an X by Y matrix called a dexel structure, which corresponds to the raster display frame buffer. Each cell in the matrix may contain one of the following: no dexel, with a background color and near Z depth of maximum Z; one dexel, with a null pointer; or more than one dexel, with a pointer to another dexel in a linked list data structure (figure 2). Multiple dexels in a cell are maintained in near to far Z order. The dexel structure is directly displayed just like an ordinary frame buffer, because the color of a dexel in the X Y matrix is the visible surface color at that pixel location.

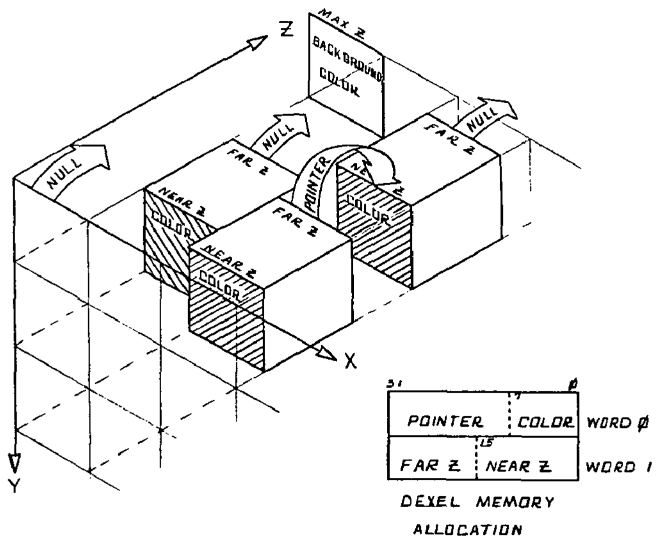


FIGURE 2

The dexel structure is created by scan conversion of surface data much like scan conversion for a conventional Z buffer, except that the Z value of the farthest surface, as well as the Z value of the nearest surface, is stored at each dexel. After scan conversion of an object, the dexel structure consists of one dimensional segments on the Z axis similar to those created by intersecting rays with objects. One dimensional Boolean operations can then be performed on these segments like those done in CSG ray tracers, but with the efficiency of the coherence of scan conversion, instead of individual ray to object intersections. Similar efficiencies have been noted in scan-line CSG algorithms. [11] For NC milling operations, cutting tools and blocks of material are often convex solids without internal concavities, so multiple dexels are created in a cell only by the cutting operation. For more general data types, scan conversion creates a linked list of surfaces in Z order at each pixel which are grouped in pairs to form dexels as a post process.

The tool is a negative solid which will be subtracted from the block or positive solid. The only difference between the tool and block dexel structures lies in the dexel color. The tool is an inverse visible surface image, in that the color of the farthest surface is stored in the dexel. The surface normals of the tool solid are inverted for shading so that they point inside the solid. When the tool is subtracted from the block, the far surface of the tool becomes the new near surface of the block, and the inversely shaded tool color is the properly shaded new block color.

The cutting procedure consists of three nested loops which step through each Y and X tool dexel, and subtract it from each block dexel in the linked list. In order to subtract an object with internal voids, a fourth loop can be added to step through the linked list of dexels in the tool. This capability would provide a general CSG subtraction operation, but it is unnecessary in NC milling display where the tool is a convex solid of revolution. The offsets in X and Y from the tool dexel to the block dexel, along with the Z offset of the tool, are three degrees of freedom in the cutting procedure. For more than three degrees of freedom, the tool dexel structure is recreated from a transformed tool surface description. Since the tool is typically small in proportion to the entire scene, and the recreation is equivalent to scan conversion, 5 or 6 axis milling can be effectively displayed.

The cutting procedure in pseudo-code (below) shows that the algorithm can be implemented in read, write, compare, and conditional branch instructions, without any geometric computation. This simplicity, and an appropriate processor with direct access to a large memory, yield real-time performance. Linked list management turns out to be a relatively small proportion of the operation, and the linked list is under 50K dexels in a typical 512 by 512 resolution display. The logical states of the operation are referenced to eight different cases (figures 3 and 4) of the intersection of two dexels.

```
procedure cut ( tool_x_min, tool_y_min, tool_x_max, tool_y_max,
                  block_x_off, block_y_off, tool_z_off)
[
structure tool_dexel [
                 tool_near_z, tool_far_z, tool_ptr, tool_clr ]
structure block_dexel [
                  block_near_z, block_far_z, block_ptr, block_clr ]

for tool_y = tool_y_min to tool_y_max [

        for tool_x = tool_x_min to tool_x_max [

                read tool_dexel (tool_x : tool_y);
CASE 1:         if tool_near_z = background then next tool_x;

                tool_near_x  +=  tool_z_off;
                tool_far_x  +=  tool_z_off;
                block_ptr = tool_x = block _x_off :
                                tool_y + block_y_off;

                while block_ptr != null [

CASE 8:                 read block_dexel(block_ptr);
                        if block_near_z = background then next tool_x;

CASE 2:                 if tool_far_z < block_far_z [
                                if tool_far_z  <  block_near_z
                                        then next tool_x;
CASE 3:                         else if tool_near_z  <  block_near_z
                                        then update block_near_z
                                                and block_clr;
CASE 5:                         else create block_dexel;
                                next tool_x;
                                ]
                        else [
CASE 7:                         if tool_near_z > block_far_z
                                        then next block_ptr;
CASE 6:                         else if tool_near_z > block_near_z
                                        then update block_far_z;
CASE 4:                         else delete block_dexel;
                                next block_ptr;
                                ]
                        ]
                ]
        ]
]
```

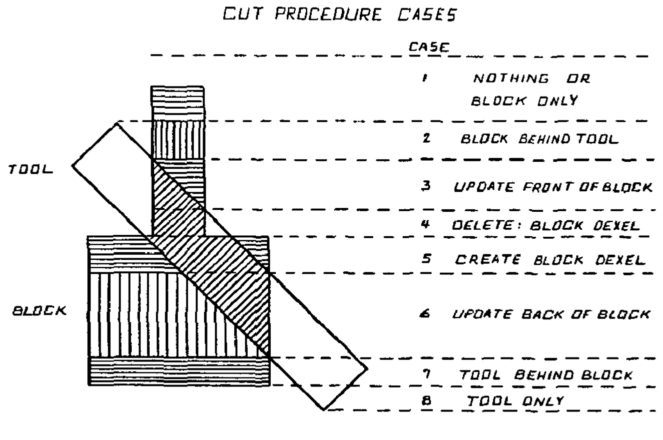


FIGURE 3

Implementation

The cutting operation is implemented on an Adage 3000 raster display system in Icross, a C-like microcode compiler for the Adage GPS bit-slice processor. The cutting function is a part of SOLID 3000, a set of Fortran-callable graphics functions for the solid model display [12]. A 100 by 100 pixel tool can be subtracted from a block at 10 updates per second, providing a continuous shaded animation of an NC path for a 3-axis machine. A new tool can be generated and subtracted at 2 updates per second for multiple cutters or for a 6-axis machine. Tools and blocks can be modeled with any SOLID 3000 data types, including bicubic surface patches and polygonal meshes.

Figure 5 shows a sequence of stages in the milling of a part. The path consists of 490 tool positions, which are linearly interpolated to 8179 steps. At each step, the cutting operation is invoked. The number of steps between each pair of tool positions depends on the distance between the positions and a programmable interpolation tolerance. A finer interpolation produces a smoother part, and a coarser interpolation more quickly follows the path. The part requires 7 cutter tools, each of which is converted into a dexel structure when it is specified in the path. The total time to mill the part at an interpolation tolerance of 3 screen units is 780 seconds for the 8179 steps, or slightly over 10 cutting operations per second.

Limitations

The view of the final part at the completion of the milling path cannot readily be redisplayed from another viewpoint. The milled part exists only as an image-resolution data structure, without any object-space surface description. Although voxel-based view transforms can be applied, the results are of the quality associated with voxel data, generally lower than that expected of solid model display. Shading for a new view by calculating new surface normals based on the dexel structure is much less accurate than the surface descriptions of the tool and its path. However, since the milling display runs in real-time, a new view can be generated by running the path from a different viewpoint, or running a path from multiple viewpoints in different image windows at the same time (figure 6).

The view of the final part at the completion of the milling is an image-resolution model that does not provide tolerancing verification or mass properties. The cutting function will not entirely replace object-space intersection calculations on general purpose computers, or test runs on the actual milling machine. However, it will limit those expensive operations to visually verified paths, and encourage interactive experiment and optimization to better use production resources. Image space verification on color raster displays is being developed by other researchers [13]. If a boundary model of the part is available to the user, it could be converted into a dexel structure and then intersected with the milled part in a matter of seconds.

Extensions

The visualization of voxel data is possible through ray-tracing techniques [14], and voxel data structures are used to optimize ray-tracing primitive intersection selection [15]. A straightforward extension of the dexel structure for milling changes the dexel color to a general attribute. This attribute can be the name of the primitive or element which created that dexel, such as a polygon, a surface patch, a quadratic, or the number of the tool position in the NC path. A ray-tracing display approach intersects each ray with the dexel structure and finds the object space element with which to perform a precise intersection. This approach retains the efficiency of the dexel structure and adds the capabilities of arbitrary viewing angles and precise surface verification.

Figures 7 and 8 are examples of ray-traced images of dexel structures with attributes as pointers to object-space surface elements. Figure 7 is a 426 surface patch scene, equivalent to approximately 9000 polygons, and figure 8 is a 13,500 polygon scene. The ray tracing display program includes reflection, refraction, shadows, and anti-aliasing entirely in a microcoded integer display processor. Each ray averages less than ten intersection calculations, instead of the equivalent thousands of potential intersections in the scene, because of the spatial locality of the dexel structure.

Acknowledgments

I'd like to thank Nick England of Trancept Systems for creating an environment at Adage which made this work possible, and Alan Coles of Northrop for the support and test data which made this work practical.

REFERENCES

[1] Pratt, Mike, "Interactive Geometric Modelling for CAD/CAM", Eurograph 84 course notes, 1984.

[2] Roth, S. D., "Ray Casting for Modeling Solids", Computer Graphics and Image Processing, Vol 18, 1982.

[3] Catmull, E., "Computer Display of Curved Surfaces", Proc. IEEE Conf. on Computer Graphics, Pattern Recognition, and Data Structures, May 1975, Los Angeles.

[4] Carpenter, Loren, "The A-buffer, an Antialiased Hidden Surface Method", Computer Graphics, Vol 18 No 3, July 1984, ACM, NYNY.

[5] Brotman, Lynne Shapiro, and Badler, Norman I., "Generating Soft Shadows with a Depth Buffer Algorithm", IEEE Computer Graphics and Applications, Vol 4 No 10, October 1984.

[6] Okino, Norio, et al., "Extended Depth-Buffer Algorithms for Hidden Surface Visualization", IEEE Computer Graphics and Applications, Vol 4 No 5, May 1984.

[7] Atherton, Peter R., "A Method of Interactive Visualization of CAD Surface Models on a Color Video Display", Computer Graphics, Vol 15 No 3, August 1981, ACM, NYNY.

[8] Doctor, Louis, and Torborg, John, "Display Techniques for Octree-Encoded Objects", IEEE Computer Graphics and Applications, Vol 1 No 3, July 1981.

[9] Carlbom, Ingrid, et al., "A Hierarchical Data Structure for Representing the Spatial Decomposition of 3D Objects", Computer Graphics Tokyo 84 Proceedings, April 1984, Japan Management Association, Tokyo Japan.

[10] Glassner, Andrew, "Space Subdivision for Fast Ray Tracing", IEEE Computer Graphics and Applications, Vol 4 No 10, October 1984.

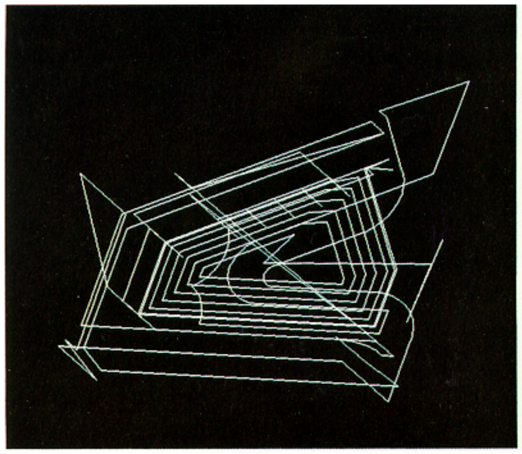
[11] Atherton, Peter R., "A Scan-line Hidden Surface Removel Procedure for Constructive Solid Geometry", Computer Graphics, Vol 17 No 3, July 1983, ACM, NYNY.

[12] Van Hook, Tim, "Advanced Techniques for Solid Modeling", Computer Graphics World, Vol 7 No 11, November 1984.

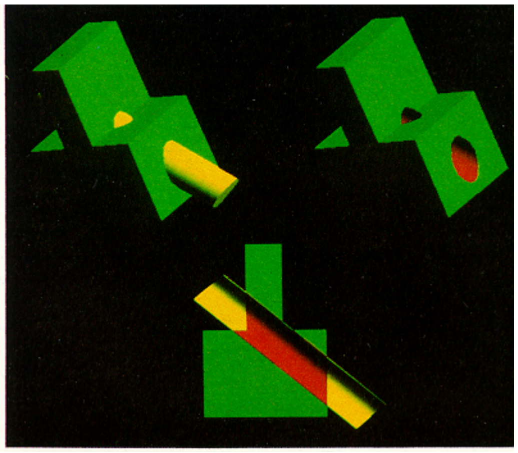
[13] Oliver, James, and Goodman, Erik, "Color Graphic Verification of NC Milling Programs for Sculptured Surfaces", Proceedings of the Automotive Computer Graphics Conference, December 1985, Detroit.

[14] Schusselberg, Daniel, Smith, Wade, and Woodward, Donald, "Three-Dimensional Display of Medical Image Volumes", NCGA Conference Proceedings, May 1986, Anaheim.

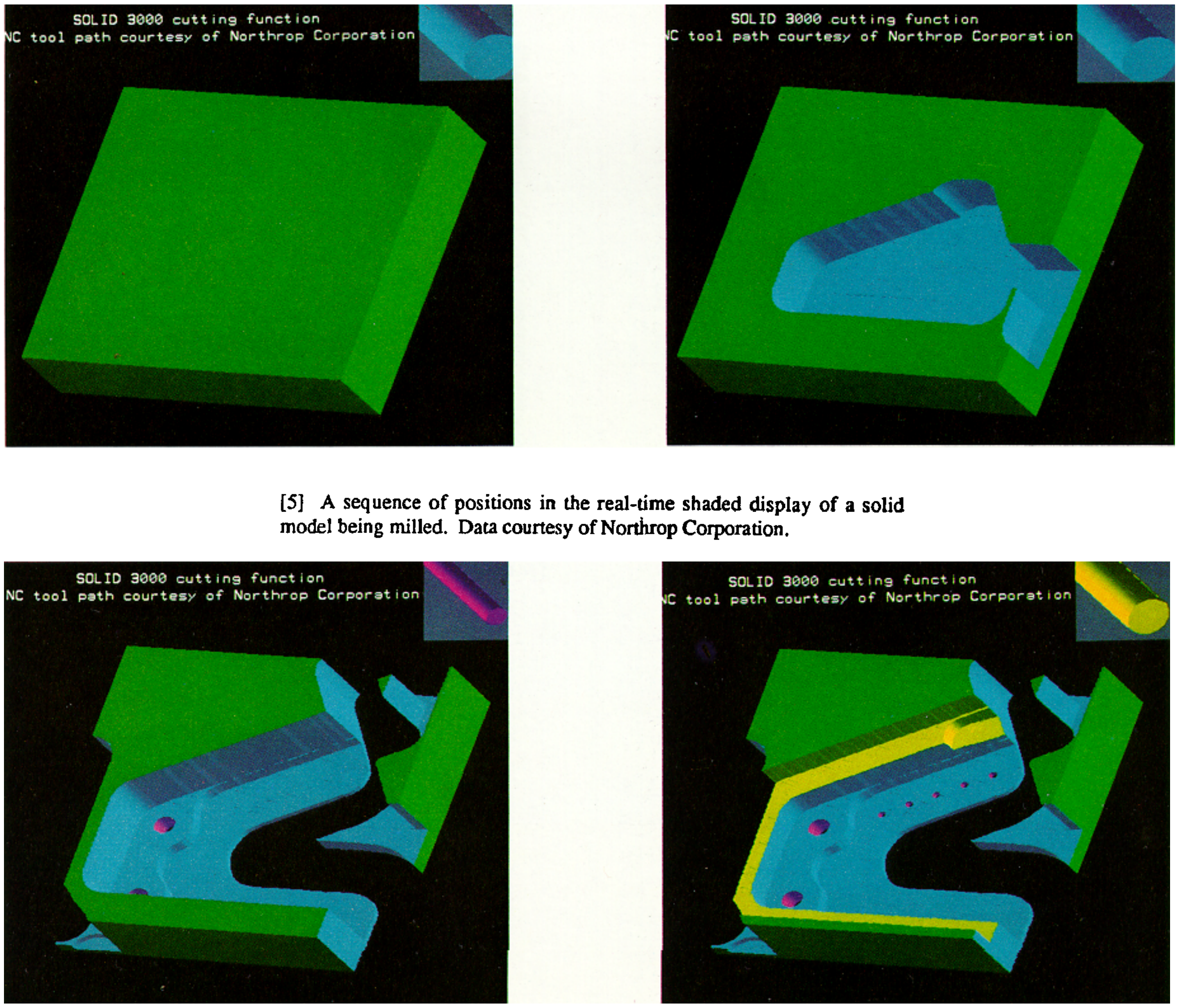
[15] Fujimoto, Akira, Tanaka, Takayuki, and Iwata, Kansei, "ARTS:Accelerated Ray-Tracing System", IEEE Computer Graphics and Applications, Vol 6 No 10, April 1986.



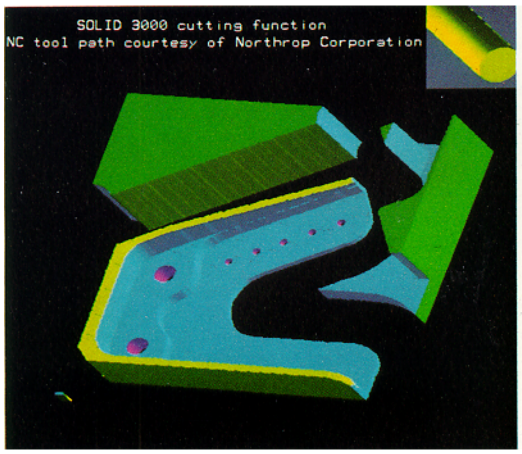
[1] An NC milling path displayed with lines connecting tool positions. Data courtesy of Northrop Corporation.

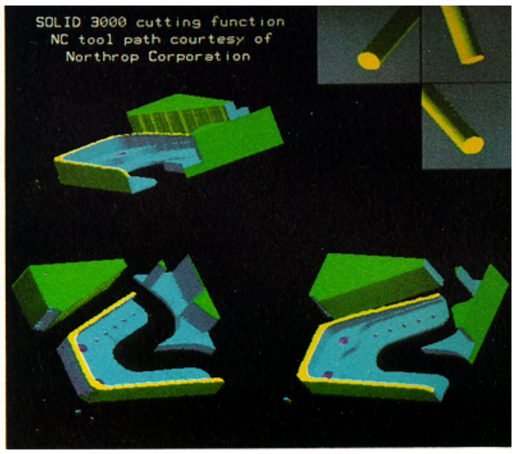


[4] The relation between a block of material and a cutting tool at one tool position. The block and tool as solid objects (left), the subtraction of the tool from the block (right), and the cutting operation for that subtraction (bottom).

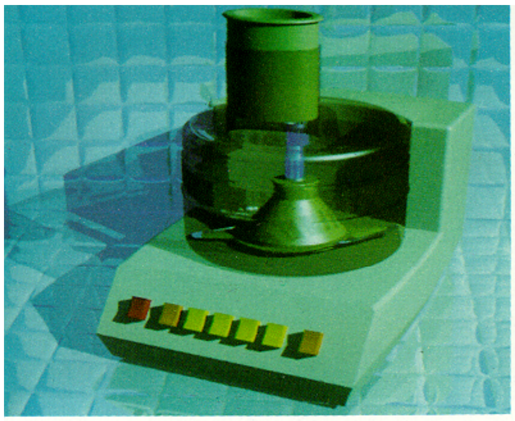

SOLID 3000 cutting function
NC tool path courtesy of Northrop Corporation

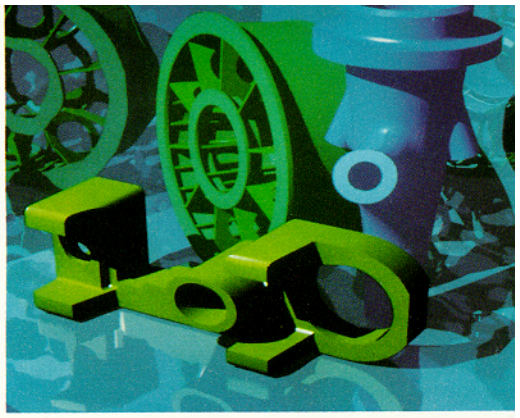
[5] A sequence of positions in the real-time shaded display of a solid model being milled. Data courtesy of Northrop Corporation.


SOLID 3000 cutting function
NC tool path courtesy of Northrop Corporation



[6] Simultaneous milling display from multiple viewpoints. Data courtesy of Northrop Corporation.





[7] Ray traced display of dexel structure. Bicubic patch data courtesy of PDA Engineering.

[8] Ray traced display of dexel structure. CAM-I test part data courtesy of Matra Datavision.