GRAPHICAL VERIFICATION OF NUMERICALLY CONTROLLED MILLING PROGRAMS FOR SCULPTURED SURFACE PARTS

By

James Herman Oliver

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Mechanical Engineering

1986

This is to certify that the
dissertation entitled

GRAPHICAL VERIFICATION OF NUMERICALLY CONTROLLED
MILLING PROGRAMS FOR SCULPTURED SURFACE PARTS

presented by

James Herman Oliver

James H. Oliver
James Herman Oliver
Doctor of Philosophy Candidate

11/12/86
Date

has been accepted towards fulfillment
of the requirements for

Doctor of Philosophy Degree in Mechanical Engineering

Michigan State University

by

Erik D. Goodman
Professor
Dissertation Advisor
Department of Electrical Engineering and
System Science

11-12-86
Date

John R. Lloyd
John R. Lloyd
Professor and Chairman
Department of Mechanical Engineering

11-13-86
Date

ABSTRACT

GRAPHICAL VERIFICATION OF NUMERICALLY CONTROLLED MILLING PROGRAMS FOR SCULPTURED SURFACE PARTS

by

James Herman Oliver

Verification of numerically controlled (N/C) machining programs prior to actual milling is often a very tedious and costly step in the manufacturing process. This dissertation describes an algorithm and its implementation in computer software which allows cost-effective checking of N/C milling programs for complex, sculptured surface parts. The current implementation is applicable to three-axis milling operations. The algorithm combines techniques originally developed for accurate, non-polygonal, surface shading and elements of B-rep solid modeling technology, to produce graphical output depicting the desired part as shaded surfaces with out-of-tolerance areas highlighted. Complexity analysis shows the algorithm to beof order N, where N is the cardinality of the tool path program. This result is compared with a direct solid modeling approach to the problem which has been shown to have complexity $O(N^4)$.

To Vince and Caroll...

ACKNOWLEDGEMENTS

I would like to extend special thanks to Dr. Erik D. Goodman, my major professor, for his guidance and support throughout my Ph.D. program. Erik provided steady encouragement when the chips were down and insightful criticism when I was sure I was right. His perpetual enthusiasm towards work and life in general has been inspirational.

The other members of my Doctoral Guidance Committee also deserve my thanks. Drs. Clark Radcliffe, Ronald Rosenberg, James Bernard, and Jacob Plotkin were all helpful, and their comments and observations concerning this work were very useful. In particular, I would like to thank Jim Bernard whose interest and initial ideas in this area launched my trek through this research.

Thanks to the people at Chrysler Corporation who provided funding for this work and the data for an excellent application example.

My family has always been a source of strength, love, and moral support for me, especially over the last five years. I only hope that I can put back as much as I have drawn. To my father and mother, Vincent and Caroll, my sisters and brothers, Linda, Gene, Deborah, Cynthia, Janet, Barbara, Michael, and Diane... thank you, I couldn't have done it without you.

Finally, a sincere thanks to my friends. As technical peers and sounding boards for (or sources of) many of my harebrained ideas, I would like to thank Ace Sannier, Jane Hawkins, and Paul Haas. To all my other friends, thanks for being there when I need you.

TABLE OF CONTENTS

Page

LIST OF FIGURES viii

LIST OF TABLES x

CHAPTER I - INTRODUCTION 1
1.0 Research Objective 1
1.1 N/C Program Verification 2
1.2 Problem Definition 3
1.2.1 N/C Geometric Verification Problem 4
1.2.2 N/C Geometric Verification Severity Problem 8
1.3 Algorithmic Implementation 10
1.4 Overview of the Dissertation 11

CHAPTER II - CURRENT N/C VERIFICATION TECHNIQUES 12
2.0 Review of N/C Verification Technology 12
2.1 Traditional N/C Verification 13
2.2 Verification Via Solid Modeling Technology 14
2.2.1 Early Solid Modeling Techniques 14
2.2.2 An Alternative Solid Representation Scheme 16
2.3 Recent Advances in Solid Modeling 19
2.4 Limitations of Direct Solid Modeling Approaches 21
2.5 Image Space Boolean Operations 22
2.6 A Surface Based Approach 24

CHAPTER III - N/C GEOMETRIC VERIFICATION - A NEW APPROACH 25
3.0 Avoiding the Constraints of Direct Solid Modeling 25
3.1 Introduction to the N/C Geometric Verification Technique 26
3.2 Input Data Requirements 28
3.2.1 Additional Input 30
3.3 User Interaction 32
3.4 Coordinate Systems 33
3.5 Comparison of Algorithmic Implementation and Problem Definition 34

CHAPTER IV - VERIFICATION PROCEDURE 35
4.0 Overview of the Verification Procedure 35
4.1 Preprocessing 35
4.1.1 Workpiece Surface Model Discretization 36
4.1.2 Sorting of Workpiece Coordinate Data 39
4.1.3 Effects of Discretization Element Size 44
4.2 Tool Path Processing 46
4.2.1 Vector/Solid Intersection 52
4.3 Postprocessing 57

CHAPTER V - APPLICATION EXAMPLES 59
5.0 Application of the N/C Geometric Verification Problem ... 59
5.1 Application One: Contouring Operation 60
5.2 Application Two: Turbine Blade 62
5.3 Application Three: Automobile Hood 67

CHAPTER VI - ALGORITHM ANALYSIS 71
6.0 Performance Analysis of the N/C Geometric Verification Algorithm 71
6.1 Order of Complexity Analysis 72
6.1.1 Growth Rate 73
6.1.2 Constant of Proportionality 76
6.2 Performance Comparison 78

CHAPTER VII - SUMMARY AND CONCLUSIONS 81
7.0 Review of the Dissertation 81
7.1 Future Research 82

APPENDIX A - LOOK AHEAD AND LOOK BACK DISTANCES 84

APPENDIX B - VECTOR/PLANE INTERSECTION 90

APPENDIX C - EXAMPLE OF THE ROBERTS ALGORITHM 92

APPENDIX D - INTERSECTION OF A VECTOR WITH SPHERICAL AND CYLINDRICAL SURFACES 95

LIST OF REFERENCES 101

LIST OF FIGURES

Page

Figure 1 Workpiece model surface normal pierces mill tool swept volume 26

Figure 2 Conversion of discretized model coordinate data from view space to mill axis space 41

Figure 3 Multiple pixels in view space can map to the same bin in mill axis space 43

Figure 4 Nearest neighbor interpolation for cut value smoothing 45

Figure 5 Construction of the swept area in mill axis space 48

Figure 6 Three cases of consecutive swept areas showing varying degrees of overlap 50

Figure 7 Example of a swept area which falls outside of the active area in mill axis space 51

Figure 8 Parallelepiped surrounding a tool swept volume 53

Figure 9 Subregions of a side face of a parallelepiped which surrounds a mill tool swept volume 55

Figure 10 N/C geometric verification of a contouring operation 61

Figure 11 Turbine blade N/C geometric verification, Case 1 63

Figure 12 Turbine blade N/C geomteric verification, Case 2 64

Figure 13 Turbine blade N/C geometric verification, Case 3 65

Figure 14 Global view of automobile hood N/C geometric verification . 68

Figure 15 Zoomed view of automobile hood N/C geometric verification . 69

Figure 16 Pseudo-code representation of N/C geometric verification algorithm 74

Figure A.1 Pseudo-code algorithm for look ahead and look back distance 86

Figure A.2 Calculation of the look ahead distance 87

Figure A.3 Calculation of the look back distance for $0 < \beta < \pi/2$ 88

Figure A.4 Calculation of the look back distance for $\pi/2 < \beta < \pi$ 89

Figure C.1 Example application of the Roberts algorithm 93

Figure D.1 Intersection of a vector with a sphere 96

Figure D.2 Intersection of a vector with a cylinder, coplanar case .. 98

Figure D.3 Intersection of a vector with a cylinder, noncoplanar case 99

LIST OF TABLES

Page

Table 1 Computation Time in CPU Minutes for N/C Geometric Verification - Turbine Blade Example 65

CHAPTER I

INTRODUCTION

1.0 RESEARCH OBJECTIVE

Numerically controlled (N/C) machining processes are used in a wide variety of manufacturing industries. In the automotive industry, for instance, N/C milling is applied in the creation of stamping dies and injection molds which are then used in the mass production of various components. Aerospace industries often machine parts directly with N/C milling technology, typically operating on very expensive materials. Regardless of the end product, N/C machining processes generally form a large portion of the total manufacturing effort.

This dissertation describes an algorithm which, when implemented within a CAD/CAM system, provides a tool for significantly improving manufacturing productivity. This work focuses on the problem of verifying the correctness of N/C milling programs prior to actual milling, particularly when dealing with complex, sculptured surface parts. N/C program verification has traditionally been a very expensive step in the manufacturing process. The algorithm described here allows cost-effective N/C program verification by efficiently checking N/C tool paths against the desired part model (and fixturing, if necessary), to

produce graphical output depicting the desired part as shaded surfaces with out-of-tolerance areas highlighted.

The N/C program verification algorithm presented here is application (product) independent; it can be applied to any three-axis N/C milling application. Through the efficient use of available model and milling information, this algorithm provides a tool for improving N/C milling capabilities and manufacturing productivity.

1.1 N/C PROGRAM VERIFICATION

Although use of N/C technology is widespread, the methods used for generation and verification of N/C tool paths vary widely depending on the end product. N/C tool paths for sculptured surface parts are typically generated via either turnkey systems, which operate on mathematical models as input, or software such as APT-SS [1], capable of model as well as tool path generation. However, current generation methods cannot generally guarantee that the tool path will properly mill the desired surface model.

Improper part programming, or flaws in path generation software, can result in areas of the part surface being overcut (gouged) or undercut (missed) relative to the desired part surface model. Also, the cutting tool may interfere with the fixtures holding the part or with other obstacles. Material cost, setup time and milling time make such errors very expensive. So regardless of the method of generation, the N/C program is generally checked before final milling. This remains one of the most difficult aspects of machining sculptured surface parts.

Current verification techniques are either labor and capital intensive, inadequate for complex geometries, or computationally uneconomical.

1.2 PROBLEM DEFINITION

Many high level general-purpose part-programming systems operate in two distinct steps; processing and postprocessing. [2] In the processing phase, mathematical models of the desired part, the milling tool, and generally, several limiting surfaces are used to generate an intermediate set of data points called the cutter location data, or CL-data. The CL-data file is sufficient to define the final shape (or geometry) of the desired part. Postprocessing involves incorporation of CL-data with machine specific factors such as tool feed and speed rates and coolant flow requirements. With some combinations of languages and milling machines, postprocessing may also involve the introduction of approximations needed to drive a particular mill: straight-line approximations to arcs, for instance.

Postprocessing parameters can, of course, affect the dimensional quality of the final part. For example, tool size and shape combined with feed and speed rates determine the material removal rate. The rate of material removal is limited by tool deflection, tool wear, and coolant flow rates. Postprocessing parameters are often determined based on conservative empirical data (tables) and the experience of the mill operator; they are typically verified via trial and error. Although these parameters are certainly an important part of the N/C program, their importance is secondary to the dimensional (or geometric) quality of the tool paths (the CL-data).

Previous computational techniques for automated N/C program verification [4]-[7] have approached the problem as a simulation problem, conducting all of the operations needed to check material removal rates, while interested only or primarily in the final geometric quality of the part. This leads to unnecessary constraints and a heavy computational load. In this work, the problem is broken into two parts: N/C geometric verification, in which the geometric quality of the milling program is checked, and N/C simulation, in which material removal rates and other such factors may be considered, but final geometric dimensions are not explicitly checked. (The term N/C simulation as used here implies calculation of either the union of all material removed or the workpiece remaining after each tool motion.) Dissection of the verification problem along these lines allows for solution of the N/C geometric verification problem independent from the constraints imposed by the N/C simulation problem. This dissertation deals exclusively with the N/C geometric verification problem, since it is the primary concern in evaluating the quality of an N/C program.

1.2.1 N/C Geometric Verification Problem

The following discussion provides a definition of the N/C geometric verification problem.

Let P represent a geometric description of trimmed, oriented surfaces comprising pertinent exterior portions of the desired milled part,

H represent a geometric description of trimmed, oriented surfaces comprising pertinent exterior portions of the necessary holding fixtures, (which are not to be cut),

N represent an N/C program including n distinct positions of the tool center, combined with appropriate specification of the paths to be followed between positions (i.e. interpolation rules),

Q represent a tool geometry,

T_{in} be a uniform tolerance limit inside the desired part and,

T_{out} be a uniform tolerance limit outside the part.

Define the workpiece model as the union of the desired part and holding fixture models. For every point p_i on (P U H), let ν_i represent the corresponding outward-directed surface unit normal vector.

For the j^{th} step in program N, combine Q with N_j and N_{j+1} and the interpolation rule, to define S_j, the surface definition of the j^{th} "swept volume".

Let $I_{i,j}$ equal the directed distance along ν_i from p_i to intersection with S_j. Then, define cut value $C(p_i) = \min_j \{ I_{i,j} \}$.

Define a <u>verification</u> mapping $V : P \rightarrow \{ -1, 0, 1 \}$ such that

$$V(p_i) = \begin{cases} -1 \text{ if } C(p_i) < -T_{in} \\ 0 \text{ if } -T_{in} \leq C(p_i) \leq T_{out} \\ 1 \text{ if } T_{out} < C(p_i) \end{cases}$$

Also, define a fixture cut mapping $E : H \rightarrow \{ -1, 0 \}$ such that

$$E(p_i) = \begin{cases} -1 \text{ if } C(p_i) < 0. \\ 0 \text{ otherwise.} \end{cases}$$

Then, the program N is geometrically verified iff for all $p_i \in P$, $V(p_i) = 0$, and for all $p_i \in H$, $E(p_i) = 0$. Program N gouges P if there exists p_i such that $V(p_i) = -1$, and misses P if there exists p_i such that $V(p_i) = 1$. The gouged region is defined as the set of points $R_g = V^{-1}(-1)$. Similarly, the missed region is defined as the set of points $R_m = V^{-1}(1)$.

Program N interferes with (cuts) holding fixtures H if there exists $p_i \in H$ such that $E(p_i) = -1$, and the fixture collision region is defined as the set of points $R_f = E^{-1}(-1)$.

Note that the tolerance model used in this definition is limited to surface features; i.e., T_{in} and T_{out} define a general unequally disposed bilateral tolerance zone relative to the desired part surface model. This tolerancing scheme conforms to the ANSI standard Y14.5M-1982. [3] However, characteristic tolerances relative to other geometric features (e.g., cylindricity, perpendicularity, concentricity, etc.) as well as material condition modifiers, are not addressed by this definition.

This definition allows each point (theoretically infinitely many) on the workpiece model, p_i, to be affected by multiple tool motions (swept volumes, S_j), resulting in multiple intersections of the normal vector and swept volumes, $I_{i,j}$. However, only the closest or deepest excursion of the tool toward or into the surface is retained for each point as the cut value, $C(p_i)$. The cut values may then be evaluated to

determine regions of the part model which have been gouged, missed, or cut within tolerance limits.

An interesting comparison can be drawn between this definition of the N/C geometric verification problem and another definition based on the direct application of solid modeling technology. Voelcker and Hunt [4] [5] define the problem as a "null-object" calculation which is required when comparing an "as milled" solid model of the part with an "as desired" solid model. For example, let A represent the "as milled" part model and B represent the "as desired" part; then if the Boolean difference (A-B) is non-null, an undercut (missed) condition exists. Similarly, if the result of the difference (B-A) is non-null, an overcut (gouged) condition exists. The N/C program mills the desired part exactly if the symmetric difference, (A-B) U (B-A), is null. In the terminology of this dissertation, calculation of the "as milled" part is called N/C simulation, while the symmetric difference null-object calculation is called N/C geometric verification.

Although both definitions of the N/C geometric verification problem are useful, they are fundamentally different. For example, the definition presented by Voelcker and Hunt does not explicitly deal with milling tolerance. Their definition could, of course, be extended to include solids at various tolerance limits but such an extension would drastically increase the size of the problem (i.e. the number of Boolean operations) to be solved. In contrast, the definition presented in this dissertation deals with direct comparison of the desired part with the action of the mill; this allows the straight-forward incorporation of a tolerance band. The two definitions theoretically agree in classifying the part as missed or gouged in the ideal case in which the tolerance

band is set to zero. Of course, this comparison is possible only when the parts to be verified are defined as complete solids, which is not a requirement of the technique presented here.

1.2.2 N/C Geometric Verification Severity Problem

A useful extension to N/C geometric verification involves consideration of the degree or magnitude of the deviations (misses and/or gouges) of the tool path from the desired part model. It involves first defining a "range of interest" which bounds the maximum miss or gouge which is to be portrayed distinctly from less severe ones; that is, more severe misses or gouges will be "lumped" as severe misses or gouges and further degree of severity information will be lost. Using the same terminology as the above formulation, the N/C geometric verification severity problem is defined as follows.

Let R_{int} denote a range of interest which designates the magnitude of the maximum distinguishable miss or gouge,

$L_{out} = R_{int} + T_{out}$, be the cut value limit for the maximum distinguishable miss, and,

$L_{in} = R_{int} + T_{in}$, be the cut value limit for the maximum distinguishable gouge.

Define a degree of cut mapping,

$D : P \rightarrow \{ -1, G(p_i), 0, M(p_i), 1 \}$, such that,

$$D(p_i) = \begin{cases} -1 & \text{if } C(p_i) \le -L_{in} \\ G(c_i) & \text{if } -L_{in} < C(p_i) < -T_{in} \\ 0 & \text{if } -T_{in} \le C(p_i) \le T_{out} \\ M(c_i) & \text{if } T_{out} < C(p_i) < L_{out} \\ 1 & \text{if } L_{out} \le C(p_i) \end{cases}$$

where, $G(p_i)$ and $M(p_i)$ are functions of the cut value, for example,

$$G(p_i) = (C(p_i) + T_{in}) / R_{int}, \text{ and}$$
$$M(p_i) = (C(p_i) - T_{out}) / R_{int}.$$

The program N is <u>geometrically verified</u> iff for all $p_i \in P$, $D(p_i) = 0$, and for all $p_i \in H$, $E(p_i) = 0$. P is <u>gouged</u> if there exists p_i such that $D(p_i) = G(p_i)$ and <u>gouged severely</u> if there exists p_i such that $D(p_i) = -1$. Similarly, P is <u>missed</u> if there exists p_i such that $D(p_i) = M(p_i)$ and <u>missed severely</u> if there exists p_i such that $D(p_i) = 1$.

The definition of holding fixture interference is the same as the one described above for direct N/C geometric verification. Also, the regions of miss and gouge are constructed in an analogous manner except that cut values are interpolated according to an interpolation rule. This rule incorporates information concerning the magnitude of the deviations between the as milled part and the desired part. The linear interpolation rule presented above is only an example, it could be replaced with other functions, based perhaps on tool geometry or a logarithmic function of the cut value.

Note that this definition of the "severity" problem reduces to the more simple N/C geometric verification problem if the range of interest,

R_{int}, is set equal to zero. Another interesting case occurs if the tolerance band is set to zero ($T_{in} = T_{out} = 0$) and the R_{int} is set equal to the tool radius. Under these conditions, very detailed features of the surface (as affected by the mill) would be highlighted, e.g. cutter cusps.

This extended definition of the N/C geometric verification problem is even more distinct from the direct solid modeling problem definition. The solid modeling approach results in mathematical models resulting from discrepancies between the tool path and the desired part. However, the solid modeling problem definition does not explicitly address the quantification of the degree of discrepancy which is represented by these solids. In contrast, incorporation of the "degree of" miss and gouge information into the problem definition described in this work is a simple and natural extension.

1.3 ALGORITHMIC IMPLEMENTATION

These definitions of N/C geometric verification lend themselves to an algorithmic implementation for their approximate solution. Such an implementation requires a method for discretization of the workpiece model {P U H} into a finite number of surface points p_i and normals ν_i. Also, efficient techniques for calculating the intersections, $I_{i,j}$, of the normals with the surfaces defining sequential swept volumes of the mill tool, S_j, are required. A method for effective and efficient display of the various regions of the workpiece model (as affected by the mill) is the final requirement. The thrust of this research is to

design, implement, and demonstrate algorithms which meet these requirements.

The verification technique described here is a new application of the technology underlying surface and solid modeling. The algorithm is designed with the goal of minimizing unnecessary computations. For instance, a convenient spatial transformation is applied to reduce the area of the workpiece model that must be considered for each motion of the mill. Also, the vector/solid intersection calculation is hierarchically refined, utilizing progressively tighter envelopes around the tool swept volume. The result is an N/C verification tool which combines ease of interpretation with computational efficiency superior to existing techniques based on traditional solid modeling approaches.

1.4 OVERVIEW OF THE DISSERTATION

Chapter Two presents a review of techniques in current use for N/C program verification and discusses applicable research efforts. Chapter Three introduces the aforementioned algorithm for N/C geometric verification and describes input requirements and necessary user interaction. The details of the verification algorithm itself are outlined in Chapter Four. Chapter Five presents several application examples of N/C program geometric verification via this new algorithm. A performance analysis of the verification algorithm is presented in Chapter Six, including a comparison of this technique with a standard solid modeling approach. Finally, Chapter Seven summarizes the work, suggests future research directions and draws conclusions.

CHAPTER II

CURRENT N/C VERIFICATION TECHNIQUES

2.0 REVIEW OF N/C VERIFICATION TECHNOLOGY

This chapter presents a review of current N/C program verification technology. The review begins with the widely used but often very inefficient and inaccurate techniques which are in current use in many manufacturing facilities. Next, some initial research efforts toward the automation of N/C verification through application of solid modeling technology are discussed. Then, more recent research is discussed which deals with improving the performance of solid modeling techniques for this application. The distinction between N/C simulation and N/C geometric verification is drawn in a discussion of the nature of solid modeling technology as applied to the general N/C verification problem. A final discussion deals with some very recent research on a surface based approach to N/C verification which is similar, in some respects, to the work presented here.

2.1 TRADITIONAL N/C VERIFICATION

The two principal methods in current use for N/C verification both depend heavily upon skilled human observation and intuition. Probably the most common method of N/C tool path verification is simply an experimental trial or "proofing" run of the program. The mill is programmed and set up with a workpiece often made of wood, high density foam or some other relatively soft, inexpensive material. After milling is complete, the prototype is measured (either manually or with a coordinate-axis measuring machine, "CMM") to determine the quality of the N/C program. Changes to the program are often necessary and several test runs may be required before an acceptable part can be milled. This is labor-intensive, and may also be quite capital-intensive, particularly when both milling and CMM equipment are involved.

The second common approach involves visual checking of two-dimensional drawings of N/C tool paths. In this method a part programmer examines line drawings representing the path of the mill in areas considered to be prone to error, in order to judge the acceptability of the tool path. The success of this method depends on the skill of the user in choosing suspect areas and in interpreting the complicated line drawings associated with these areas. Since the surface itself is represented by a line drawing, interpretation becomes increasingly difficult as the surface and its associated tool path become more complex. Though this method can be effective in identifying gross milling errors, it is by no means comprehensive, and must generally be supplemented by the trial milling process described above.

2.2 N/C VERIFICATION VIA SOLID MODELING TECHNOLOGY

Principal research efforts in the automation of N/C program verification rely extensively on the use of solid modeling technology. The typical procedure for N/C verification via solid modeling involves creation of a complete geometric solid model representing the part as milled and a comparison between this model and another representing the desired part. These calculations are accomplished through the application of the regularized set operations union, difference, and intersection. [8] For example, an "as milled" part model can be obtained by subtracting (via Boolean difference operations) solids representing the volumes swept out by each motion of the mill tool from a solid model of the workpiece.

Using the terminology introduced in Chapter One, straightforward application of solid modeling techniques involves performing the Boolean operations required for solution of the N/C simulation problem, even if only the solution of the N/C geometric verification problem is desired.

2.2.1 Early Solid Modeling Techniques

Most currently available solid modeling systems can be described as primarily either constructive solid geometry (CSG) systems or boundary representation (B-rep) systems. [8] [9] With CSG modelers, objects are represented as collections of primitive solids (such as cylinders, cones, blocks, etc.) connected via Boolean operations. B-rep modelers represent solids as structures which incorporate collections of faces described by surface equations (usually planar or quadric), bounding

edges, and vertices. B-rep models are typically constructed with functions called "Euler operators" [10] which define and maintain an object's topological boundary in a hierarchical form. Through various conversions, many modelers maintain both of these data structures internally; using, for instance, CSG for Boolean operations and B-rep for shaded display purposes.

Voelcker and Hunt [4] [5] were among the first to suggest the application of solid modeling to the N/C verification problem. Using a CSG-based solid modeler they demonstrated verification capabilities for very simple parts milled with "2-1/2 D" N/C programs. This work showed that the bulk of the (heavy) computational load for N/C verification results from the null-object calculation (see Chapter One) which is required when comparing the "as milled" part model with the "as desired" model. Although this work was exploratory in nature, dealing with simple models and N/C programs, it illustrates many of the problems inherent in the application of solid modeling to this problem.

Similar direct solid modeling approaches to N/C verification were extended to three and five-axis milling applications. [6] [7] These techniques allowed more sophisticated milling operations and tool libraries, but were still based on simple CSG modelers, thus limiting their application to parts which can be defined with collections of relatively simple geometric primitives. The N/C verification technique used in these papers is essentially the same as that presented by Voelcker and Hunt, but the added complexity of the swept volume models incurs a very heavy computational load. Also, the method used to create a shaded display of the part after verification is based on a ray

tracing algorithm. [11] Such rendering algorithms are typically more computationally intensive than other surface shading techniques.

Standard primitive-based solid modeling technology could theoretically be applied to N/C program verification of complex, sculptured surface parts. However, to represent such a solid with this type of system would require an excessive number of primitives. The overall size of typical industrial parts is often many orders of magnitude larger than the tolerance required for milling. An accurate primitive-based solid representation of a part with these properties would require a very large data set. The Boolean operations necessary for verification of N/C tool paths via solid modeling are computationally intensive even for simple geometries. The added burden of applying such algorithms to very large data sets makes this approach practically intractable.

2.2.2 An Alternative Solid Representation Scheme

A data structure which is much more efficient at Boolean operations than CSG- and B-rep- based solid modelers is a hierarchical spatial decomposition, such as the octree representation. [12] [13] This technique involves the hierarchical encoding of a solid based on recursive subdivision of a three-space volume into eight equal octants. At each level of subdivision, an octant is labeled as either empty, full, or partially full. An octant is subdivided if it is both partially full and larger in size than the desired resolution. The hierarchy of labels thus created forms the octree model of the solid.

The primary advantage of octree encoding is efficiency in performing Boolean operations. Given several octrees representing several different solids, Boolean operations can be performed by simultaneously traversing each octree and comparing at each level the labels which represent the status of each octant. An octree representing the resulting solid is built level by level based on these comparisons. Each comparison is simply a logical operation and the regular order of the data structure is exploited, so Boolean operations are typically very efficient computationally. In fact, the regular structure of the octree representation may soon lead to hardware implementation of Boolean capabilities using VLSI technology.

Although this method of solid representation is not as common as classical CSG or B-rep techniques, many feel that the computational efficiency afforded by octrees offers the greatest hope in representing and manipulating complex objects. For example, in his work on tool path generation for CSG solids, Bobrow [14] conjectures that octrees may someday provide "the ability to verify tool paths to within normal machining tolerances".

However, significant problems arise when the octree solid modeling approach is extended from simple, easily representable objects to the complicated parts required by current users of N/C technology. The octree data structure, though computationally efficient, requires a prohibitive amount of memory to accurately represent complex solids, especially when dealing with the tolerances required in N/C machining applications. The size of the octants at the deepest level of the octree must be on the order of the milling tolerance, while the octant at the root of the octree is on the order of the size of the largest

dimension of the part. For N/C verification of complex parts, the difference in these dimensions guarantees an extremely large octree model.

A quantitative example will illustrate this point. Consider a solid part model which resembles a box composed of five planar surfaces and one bi-cubic Bezier-type sculptured surface. Suppose that the entire solid nearly fills a cube which is one inch on a side and that the desired milling tolerance ($T_{in} - T_{out}$) is one thousandth of an inch (0.001). In B-rep form this model could be defined with as few as 68 real numbers; four constants defining each of the five planes and three 4X4 coefficient matrices (one each for X,Y, and Z) for the Bezier surface. An equivalent octree model would require much more memory. Meagher [12] shows that the memory requirement for a 3-D objects modeled with octrees is on the order of the surface area of the object. In this example, the necessary units are thousandths of an inch, so that the surface area of the solid is approximately $6 \times 1000 \times 1000 = 6 \times 10^6$. Thus, even such a simple solid yields a difference in memory requirement of approximately five orders of magnitude.

Another significant problem in application of octrees to sculptured surface N/C verification is the actual creation of the octree models themselves. To create an octree representation of a solid composed of sculptured surfaces requires decisions at each level of subdivision as to whether points defining the octants are inside or outside the solid. These decisions would require either significant user interaction or the use of a very sophisticated sculptured surface solid modeler at each step of the subdivision. Alternatively, assuming the availability of a sufficiently accurate CSG or B-rep approximation of a sculptured surface

solid, one could apply algorithms designed to convert these data structures into octree form. [15] [16] However, application of such conversion algorithms would incur a significant computational burden on the verification process since they must operate on very large CSG or B-rep databases.

For these reasons, octree modeling technology in its current state is not well suited for application to the N/C geometric verification problem. Although octree encoding could provide efficient Boolean operations between solids derived from sculptured surfaces, overhead cost in memory and preliminary computational load in creating the requisite octrees apparently outweighs the computational gains the method may afford. However, as memory cost continues to decline and development of more sophisticated octree software and hardware continues, this technology may emerge as a viable solution to the N/C geometric verification problem.

2.3 RECENT ADVANCES IN SOLID MODELING

More recent advances in solid modeling research have led to systems capable of incorporating sculptured surfaces as part of their data structures. Kimura [17] [18] uses an enhanced B-rep data structure in which Bezier-type sculptured surfaces are allowable as faces. Casale and Stanton [19] present a unique data structure called the tricubic hyperpatch which is neither CSG nor B-rep based. Although these authors do not explicitly address N/C verification, both claim to incorporate Boolean operation capabilities. Thus, sculptured surface N/C verification via such modeling systems is possible.

By incorporating a more general data structure, these methods would be capable of accurately representing sculptured surface solids with a much smaller database than the aforementioned techniques. However, the smaller size of the database comes at the expense of increasing the complexity of performing Boolean operations on objects in that database. Specifically, Boolean operations on sculptured surface solids require calculation of the curves defining the intersection of two or more parametric sculptured surfaces. In general, calculating either exact or approximate solutions is a very difficult problem, requiring intensive computations.

Kimura proposes the use of subdivision techniques to approximate sculptured surface intersections. Most subdivision techniques are recursive in nature and are computationally intensive, depending on the curvature of the intersecting surfaces and the accuracy desired. [20] [21] Butterfield [22] presents a more precise, albeit more costly, alternative for sculptured surface intersections using either a Newton-Raphson-type iteration method or some form of distance minimization. Both these alternatives introduce even greater computational requirements. Since many Boolean operations are necessary to perform N/C geometric verification of a sculptured surface part and the tolerances involved may be quite small, these methods, though theoretically possible, are not well suited to current manufacturing practice.

2.4 LIMITATIONS OF DIRECT SOLID MODELING APPROACHES

The primary problem encountered in the direct application of any of the solid modeling techniques to N/C geometric verification of realistically complex parts lies in the trade-off between, on the one hand, the size and complexity of the database, and on the other, the difficulty inherent in the manipulation of that database. Typically, the smaller the size of the database used to describe a given solid, the greater the complexity of the algorithm used for its manipulation, and vice versa. For N/C verification applications, where complex parts with tight tolerances are involved, one can choose a solid representation that is either efficient in performing Boolean operations but requires a very large database, or a scheme which allows a relatively compact database at the expense of very complex algorithms for Boolean operations. Both of these options require intensive computational effort.

In the context of the definitions given in Chapter One, the direct application of solid modeling techniques to the N/C geometric verification problem leads to many unnecessary Boolean operations and to unnecessarily precise computation of intermediate solids. For example, many of the swept volumes representing tool motions, when subtracted from the workpiece model, result in bounding surfaces which are nowhere near the final surfaces of the part, but are fully calculated nonetheless.

A more fruitful approach would seek to eliminate unnecessary calculations by skipping those evaluations which are distant from the desired part surface, and to use increasingly more precise evaluation

methods as the desired part surface is approached. While the octree encoding technique appears to offer strong possibilities for implementing such an approach, it would require special-purpose algorithms to do so. The direct application of Boolean operations on arbitrarily complex solids appears to be ill-suited to the N/C geometric verification problem.

The technique presented in this dissertation takes advantage of the special nature of the N/C geometric verification problem and applies the ideas sketched out above for improved computational efficiency. The details of this algorithm are presented in Chapters Three and Four.

2.5 IMAGE SPACE BOOLEAN OPERATIONS

One of the most promising developments in solid modeling research applicable to N/C verification is the so called "visual solid modeler". In this scheme, Boolean operations are computed "on-the-fly" during image rendering (shading). Roth [11] was among the first to suggest that the three-dimensional Boolean operation could be reduced to a one-dimensional problem by considering the intersections of sightlines (rays) from each image picture element through a CSG solid model. Atherton [23] extends this idea to a polygon-based scan-line hidden surface removal algorithm, using various coherence techniques to gain significant performance improvements over the simple ray-casting approach. These techniques provide a relatively fast means for displaying solids resulting from Boolean operations because they do not calculate and maintain the complete geometrical definition of the resulting solid. Thus, the result is completely view dependent; if

another view is selected, the entire procedure (Boolean operation and rendering) must be repeated.

Independent of Atherton, Wang [24] developed a similar scan-line based rendering algorithm for image space Boolean operations with the specific application of N/C milling in mind. In this initial work, Wang presents a mathematical basis for mill tool swept volumes and develops a prototype system for simulating three-axis N/C milling. The simulation involves relatively simple parts composed of CSG primitives. Also, the simulation stops short of N/C geometric verification, calculating and displaying the "as milled" part resulting from the Boolean difference of each tool swept volume from the stock material model.

However, more recent work by Wang [25] shows that many of these shortcomings have been addressed and eliminated. For instance, Wang now claims to have the capability for full N/C verification (including overcut and undercut conditions) for five-axis milling applications. Sightline intersections with swept volumes resulting from general five-axis mill motion requires the solution of nonlinear equations which contain transcendental functions. Also, the full null-object Boolean operations must be performed to produce models of the missed and gouged portions of the workpiece. Despite these computational requirements, Wang apparently obtains results with impressive computational speed. Wang's group is currently developing capabilities for sculptured surface solid modeling [26], which will undoubtedly be incorporated with his work.

2.6 A SURFACE BASED APPROACH

Some very recent research by Jerard [27] addresses the N/C geometric verification problem with a surface-based approach. This research has been sponsored by Ford Motor Company and deals specifically with three-axis milling of sculptured surface parts. Jerard applies a first order curvature approximation to evaluate a grid of points on the surface of the workpiece which is dense enough to account for surface irregularities relative to the size of the mill tool. He does not evaluate surface normals but instead applies a Z-buffer algorithm [28] on the point grid itself as he evaluates the tool paths. (Jerard assumes that the mill axis remains parallel to the global Z axis.) The surface "cut values" are thus measured relative to the mill axis.

Although Jerard's work is substantially different from the technique presented in this dissertation, it is interesting to note that he had independently reached a similar philosophy: quickly eliminating motions of the mill which do not directly affect the final finished part. Both Jerard's technique and the method presented here take advantage of the fact that the final desired surface model is the basis for the N/C geometric verification problem.

CHAPTER III

N/C GEOMETRIC VERIFICATION - A NEW APPROACH

3.0 AVOIDING THE CONSTRAINTS OF DIRECT SOLID MODELING

In Chapter One, the problem of N/C program verification was divided into two distinct parts: N/C geometric verification and N/C simulation. Chapter Two described several attempts at direct solid modeling solutions of the general N/C verification problem. Because these solid modeling techniques generate information required for the N/C simulation problem while solving the N/C geometric verification problem, they tend to be very intensive computationally. This Chapter introduces a new algorithm for solving the N/C geometric verification problem. Exploiting the distinct properties of the geometric verification problem, the algorithm gains efficiency by operating free from the constraints of the N/C simulation problem.

Chapter Three begins with a general introduction to a new method for the solution of the N/C geometric verification problem. Next, input data requirements are discussed, emphasizing the differences between this technique and solid modeling methods. Finally, the necessary user interaction is described as well as its effect on the performance of the algorithm.

3.1 INTRODUCTION TO THE N/C GEOMETRIC VERIFICATION TECHNIQUE

The approach to N/C program verification described in this dissertation was developed from a synthesis of sculptured surface rendering (color shading) techniques and solid modeling. The method involves intersection of workpiece surface normal vectors with "implicit" solid models which represent the motions of the mill (see Figure 1). The swept volume solid models are never calculated explicitly. They are maintained, instead, as collections of progressively tighter enveloping surfaces.

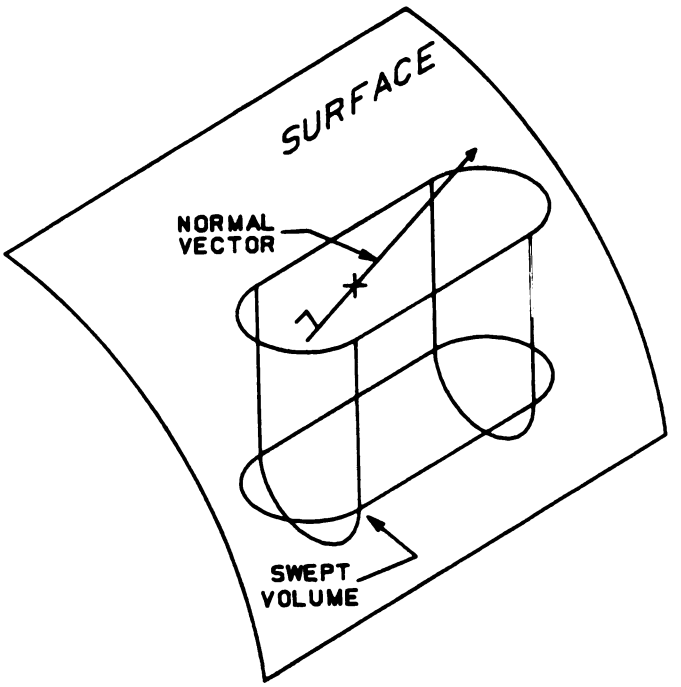


Figure 1. Workpiece model surface normal pierces mill tool swept volume

Techniques developed originally for accurate (non-polygonal) color shading of sculptured surfaces are applied for systematic calculation of surface normal vectors at each picture element (pixel) of a raster-type display device. The verification procedure is, however, fundamentally independent of the particular algorithms used in this preprocessing phase. Any other method could be used to discretize the part model so long as it resulted in arrays containing surface coordinates and the corresponding normal vectors.

Pixel-based discretization of the part model is used here for ease of presentation and user convenience. In particular, it provides the user with two ways to affect the relative resolution of the check. The first is through view selection. The larger the object appears in the view, the greater the number of pixels associated with a given area of the model, and thus, the more accurate the calculation. Similarly, views from further away, or from oblique angles, reduce the number of pixels considered, decreasing computation time at the expense of resolution.

The second way that the user can control checking resolution is via a discretization element size. Although normal vectors are calculated for each visible pixel on the part model, the user has the option of including only a subset of the total in the actual intersection calculation. In a manner similar to view selection, the user may trade off checking resolution (and hence accuracy) for computational speed. The details of how the discretization element size affects the verification algorithm are discussed in Chapter Four.

The mill path is modeled as a series of B-rep solids which represent sequential volumes swept out by the motions of the mill; each solid is compared with appropriate workpiece surface normals to calculate the distance between tool and workpiece, then discarded. A convenient spatial transformation is used to insure that a relatively small subset of normal vectors is considered for each sweep of the mill tool.

The intersection calculation for any given normal vector proceeds in hierarchical steps as far as needed through a series of progressively more exact definitions of the shape of the tool swept volume. The results of intermediate calculations are used to determine if further, more sophisticated swept volume intersection calculations are required. This structure insures that redundant or superfluous calculation of vector/solid intersection is minimized. The resulting image shows the interaction of the mill with the desired part model by color shading based on the signed distance between the desired part surface and the surface actually milled. There is a distinction, in the intersection calculation, between vectors which have been missed and those gouged; both conditions are computed simultaneously and displayed together in the output.

3.2 INPUT DATA REQUIREMENTS

The desired part model used in this method consists only of those (trimmed) surfaces which define the exterior of the desired part (and holding fixtures, if desired) which are to be checked for proper cutting or for unwanted violation. The part surfaces would typically be

directly produced by a CAD modeling package. Construction planes and other intermediate surfaces, (i.e. portions of surfaces which may have been trimmed away in solid modeling Boolean operations) must be identified as such to the processor or they will also be checked and shaded as missed, gouged, or within tolerance, perhaps confusing the user. Workpiece surface definitions which form complete closed solids (such as B-rep outputs from a solid modeler) are acceptable, though by no means necessary. The minimal part model is a single sculptured surface patch.

Only the rational (nonuniform or uniform) B-spline surface type is used in this work. [29] [30] It is a very flexible surface representation which facilitates the use of fully sculptured, as well as quadric and planar surfaces. This allows for easy incorporation of simple models of holding fixtures or other obstacles into the workpiece model. A very efficient technique developed by Pickelmann [31] is used here to evaluate rational B-spline surfaces (points, parametric derivatives and normals). For a typical bi-cubic rational B-spline surface, this evaluation technique is approximately three times faster than the commonly used Cox-DeBoor algorithm [32] [33]. Furthermore, like the Cox-DeBoor algorithm, Pickelmann's technique avoids the numerical instabilities associated with evaluation of nonuniform B-splines; hence it is much more accurate than methods which apply the definition of the B-spline directly.

Since a sculptured surface part is milled from a solid piece of stock, and the workpiece model used in this work need not be a complete model of a closed solid, a convention is necessary to distinguish the "outside" of a surface from the "inside", to yield oriented surfaces

similar to those used in "half-space" solid modelers. In this case, the primary purpose of the convention is to allow distinction between gouges and misses. On parametrically defined surfaces, normal vectors are defined by the vector cross product of tangent vectors in each parametric direction. The outward facing side of a surface is defined, in this work, by the order in which the surface data were input. Outward pointing normal vectors are defined as the cross products of tangents in the first parametric direction with those in the second. (It is simple to transform improperly oriented surfaces which may be created by a modeler.)

3.2.1 Additional Input

In addition to the desired part model, other input required for this method includes the cutter location data (CL-data), the dimensions of the cutting tool, the desired tolerances, the range of interest, and the discretization element size. The CL-data file contains the spatial coordinates defining the position of the mill. It is assumed that any nonlinear point to point interpolation rules in the CL-data file have been postprocessed into linear segments. The development presented here deals with three-axis milling. Thus, the order of the points in the CL-data file completely determines the motion of the mill, as it moves in a straight line from point to point. Necessary cutting tool dimensions include the height of the tool and the radius, R, of the spherical end. For simplicity, the spherical or ball end tool is used as the basic tool geometry in the initial development of this work. However, other tool geometries can be accommodated easily since the subroutines which calculate the mill swept volume are modular.

The milling tolerances are used to determine the hue of pixels which map to the outside of the workpiece model in the selected view. Two tolerance values are required as input; one (T_{out}) representing a distance outside the surface, the other (T_{in}) a distance inside the surface, which together define the limits of an acceptable cut. (See Chapter One for a precise tolerance definition.) A pixel's cut value is defined as the signed distance, calculated along the normal vector from the surface to the mill swept volume which yields the smallest (or most negative) length. (Refer again to Chapter One.) If a cut value falls between the limits defined by T_{in} and T_{out}, the pixel is assigned the hue representing an acceptable cut.

Recall that the range of interest, R_{int}, represents a distance outside of the surface and its tolerance limits, beyond which the user is unconcerned about quantifying the relative distance between the mill and the tolerance limit from the surface. Also recall that the maximum distinguishable (quantifiable) miss is defined as $L_{out} = R_{int} + T_{out}$. The primary use of the range of interest is in enabling quicker handling of passes of the mill which are rough cuts or positioning motions. The range of interest is also used as part of the display technique. Any pixel with a cut value greater than L_{out} is assigned the hue representing the maximum distinguishable miss. Pixels with normals which intersect a swept volume in a distance less than L_{out}, but greater than T_{out}, are assigned a hue based on interpolation between the hues representing "complete miss" and "just beyond tolerance". The procedure is analogous for gouging, where the deepest or maximum distinguishable gouge is defined as $L_{in} = R_{int} + T_{in}$.

The discretization element size is used to reduce the number of normal vectors which participate in the precise intersection calculation. It is restricted to integer values between one and sixteen, inclusive. Let the discretization element size be denoted by M; it is applied by sending the normal vectors at every M^{th} pixel on every M^{th} scanline into the vector/solid intersection portion of the algorithm. Cut values are then interpolated for pixels whose normals were not intersected exactly. The interpolation is a linear, nearest neighbors scheme, based on the distance from the pixel in question to the (up to) four closest pixels which were calculated exactly.

3.3 USER INTERACTION

Before N/C geometric verification begins, the user is required to select a view of the workpiece as it is displayed in wireframe (or flowline) form. A typical view would show most, or all, of the milled surfaces from outside the workpiece, although zoomed views of critical areas could also be chosen. Milling operations such as pocketing or contouring, in which vertical "walls" (i.e. surfaces parallel to the mill axis) are to be machined, may require several views and verification runs to check completely. Only surfaces visible in the selected view are included in the verification procedure.

After view selection, the user may choose a portion of the screen, called a subwindow, which contains the entire workpiece wireframe or some portion of it. The subwindow can significantly reduce the number of pixels which must be considered, thus reducing computation time. For example, the subwindow is particularly useful when a zoomed view

completely fills the screen with the wireframe drawing, but only a portion of the workpiece model is of interest.

3.4 COORDINATE SYSTEMS

Four coordinate systems are used in this work: world space, view space, screen space, and mill axis space. World space refers to the three-dimensional cartesian coordinate system in which the workpiece model and the tool path CL-data are defined. Milling operations are often programmed after a part has been rotated from the cartesian coordinate system in which it was designed into a more convenient orientation for milling. This "fixtured" orientation will be called world space. The mill axis is assumed to remain parallel to the global (world space) Z axis. Milling operations which require changes in workpiece orientation and refixturing can be handled by preprocessing the part model with an appropriate transformation for each change in orientation.

Upon view selection, the world space data are linearly transformed (rotated, translated, and scaled) into an equivalent three-dimensional space called view space. An axonometric projection of view space data onto the two-dimensional grid of pixels representing the raster-type display device yields screen space. Mill axis space is similar to screen space, but results from the projection of view space data onto a plane which is perpendicular to the mill axis. The development and use of mill axis space is discussed in detail in the following chapter.

3.5 COMPARISON OF ALGORITHMIC IMPLEMENTATION AND PROBLEM DEFINITION

The algorithm introduced in this chapter (and detailed in the next) is a practical implementation of procedures to solve the general N/C geometric verification problem as defined in Chapter One. However, the infinitesimal part model discretization implied in the problem definition is approximated, in the algorithmic implementation, by a view-based, pixel level discretization. This approximation introduces some practical limitations on the accuracy of the algorithm.

Any method of surface discretization, including pixel level discretization, results in an approximation of the properties, e.g., surface coordinate and normal vector, of a finite area of the surface with those of a single point within the area. The crucial factor in pixel level discretization is the maximum physical distance between adjacent pixel centers. Consider, for example, a model of a ship hull which is displayed in its entirety via pixel level discretization on a raster type device. The distance between pixel centers may be so large that significant surface irregularities, such as slope discontinuities or areas of high curvature, may not be visible. In the extreme case any object, regardless of size, can be viewed from a point distant enough so that the entire object maps onto one pixel. Such a view is, of course, of little use.

The accuracy of the N/C geometric verification algorithm presented here is limited by the accuracy of the method used for surface discretization. If a view is selected such that the maximum distance between adjacent pixels is greater than the minimum local radius of curvature, the algorithm may produce misleading results.

CHAPTER IV

VERIFICATION PROCEDURE

4.0 OVERVIEW OF THE VERIFICATION PROCEDURE

In the last Chapter, a new technique for solution of the N/C geometric verification problem was introduced. This Chapter describes the algorithm in detail. The process of N/C geometric verification is divided into three phases: a preprocessing phase, in which the surfaces of the workpiece model are discretized and normal vectors are calculated at each pixel; a tool path processing phase, in which cut values are calculated for each pixel; and a postprocessing phase, in which the cut value and normal vector are used to assign a hue and intensity at each pixel.

4.1 PREPROCESSING

The preprocessing phase entails two tasks: discretization of the workpiece model resulting in surface points and normal vectors, and sorting of these data to allow for more efficient tool path processing. The majority of the computational effort in the preprocessing phase results from the first of these tasks.

4.1.1 Workpiece Surface Model Discretization

Discretization of the workpiece model into pixel-size elements is accomplished with an algorithm which was originally designed for accurate shading of sculptured surfaces. The following discussion provides some background on the development of surface shading algorithms and describes the state-of-the-art technique used in this work for surface discretization.

Many shading algorithms apply various techniques to approximate a sculptured surface with polygon "tiles", in order to simplify and speed the shading process. [34] A simple method for "tiling" a bivariate parametric surface is to evaluate a grid of points on the surface at regular parametric intervals and to connect the points linearly to form polygons. Many algorithms exist for shaded display of objects composed of polygons. One of the most widely used methods for shaded polygon display is the scan line technique [35].

A scan line is a row of pixels in a raster display. Scan line algorithms involve processing an image from left to right, top to bottom. For each pixel, a view space vector is constructed from the viewpoint (at infinity) through the pixel in the direction of the surface model. If this vector, called a sightline, intersects the surface model, the (view space) coordinates of that surface point and the normal vector there are calculated and stored. If there are multiple intersections on one sightline, the point closest to the viewer is used. Scan line methods gain efficiency by making use of scene coherence, i.e. information about surface intersections of the previous

sightline is used in calculating the intersections of the current sightline.

The primary problem in applying polygon shading techniques to sculptured surfaces is that the number of polygons necessary to represent a complex surface accurately can become excessive. Also, to avoid a faceted appearance, polygon shading techniques typically involve interpolation or smoothing of normal vectors [36] [37] which can potentially result in loss of important surface features. For example, silhouette edges appear as piecewise linear edges instead of smooth curves. Also, surface slope discontinuities can be washed out by the "tiling" process or by normal vector smoothing.

Catmull [38] was among the first to address direct shading of sculptured surfaces. He developed an algorithm which recursively subdivides a bicubic parametric surface patch until the size of a subpatch is on the order of the size of a pixel. However, the performance of this algorithm degrades if the surface has high curvature or is viewed from a poor orientation. Also, since this method does not assign pixels in an orderly fashion, it is not well suited to raster-type display environments.

Catmull's technique was later extended into a scan line type algorithm by Lane and Carpenter [39]. At about the same time, other direct scan line algorithms for sculptured surface shading were introduced by Blinn [40] and Whitted [41] [42]. Each of these techniques, however, make various approximations to the exact surface definition in order to gain computational efficiency. Lane and Carpenter apply a patch subdivision technique which terminates when a

"flatness" criterion is met, writing subpatches not on the current scan line to an inactive patch list. Blinn computes sightline intersections only at so-called key visual points and interpolates normal vectors between them (along a scan line). Whitted's technique is similar to Blinn's except that silhouette edges are first fitted with cubic splines and patches are divided into forward- and rearward-facing components before scan line processing begins.

Vanderploeg [43] developed an "exact" sculptured surface shading technique which avoids the approximations described above yet maintains considerable computational efficiency. He uses a scan line algorithm similar to one developed by Blinn, but takes advantage of efficient patch preprocessing and a virtual memory environment to gain improvements in speed and accuracy. This algorithm calculates an accurate surface normal at each sightline intersection. Vanderploeg's initial work was limited to bicubic Coons-type surface patches. Pickelmann [31] extended this algorithm to provide accurate and efficient shading of rational B-spline surfaces.

In this work, the Vanderploeg/Pickelmann algorithm is applied to the workpiece surface model as the initial step in the preprocessing phase. The procedure results in stored surface coordinates and corresponding normal vectors for each pixel in the user-selected subwindow. Portions of the workpiece model which are not visible in a given view are not retained in this phase and thus are not included in the verification procedure.

4.1.2 Sorting of Workpiece Coordinate Data

Since the pixel discretization of the part model is based on an arbitrary view selected by the user, the coordinate data must be transformed into a more convenient space in order to process the tool path in the most efficient manner. This step in the preprocessing phase was motivated by results of some early work by this author. [44] In this initial approach, the subset of pixels to be considered for a motion of the mill was based on the projection of the three-dimensional swept volume defined in view space onto the two-dimensional screen space. This technique effectively considered all the pixels which could possibly have been affected by a mill motion; it was, however, much too conservative. In certain views, a pixel could be included in the intersection calculation for many different swept volumes when it was actually affected by only a few. Even though the vector/solid intersection process was designed to quickly eliminate normals intersected out of range, the initial subset of normals considered was too large for computational efficiency.

From an efficiency standpoint, the best viewpoint from which to watch a milling operation -- the position from which the cutting tool occludes the smallest area of the workpiece -- is from a point above the workpiece looking down along the mill axis. If the user were restricted to choosing only this view, then the efficiency problems of the earlier approach discussed above would be effectively avoided. A more elegant solution is to maintain arbitrary view selection, but to map the discretized coordinate data from view space into an effective screen space, (henceforth called "mill axis space") which would result if the pixel data were viewed from a "mill's eye view". The tool path may then

be processed (followed) in mill axis space. A milling tool swept volume projected onto mill axis space will occlude only those areas of the surface which it can possibly cut. Mill axis space is used only to determine the smallest subset of normal vectors which must be considered for any given motion of the mill. The vector/solid intersection calculation takes place in view space.

Mill axis space is two-dimensional since the third coordinate, corresponding to the mill axis, is not necessary to determine the projected area that the mill occludes for a given motion. An alternative approach would be to convert the entire problem into a three-dimensional mill's eye view, performing both path following and vector/solid intersections in such a space. However, the computational expense of correlating the mill axis space data to equivalent view space data is significantly less than the cost of calculating the three-dimensional components of each normal vector in a "mill's eye" view as well as the third coordinate of each surface point (corresponding to the mill axis).

The view space coordinates of each pixel are mapped into mill axis space via a two-step process (see Figure 2). First, the transformation matrix which converts world space data into view space is inverted and applied to the array of view space pixel coordinate data to yield the (two-dimensional) world space equivalent of the discretized part model. Only pixels which map onto the outside of the part model (i.e., those with normals pointing out of the screen) are eligible for this transformation. Only two coordinate values need be calculated for each pixel since the coordinate corresponding to the mill axis is unnecessary. This step effectively converts the discretized data into

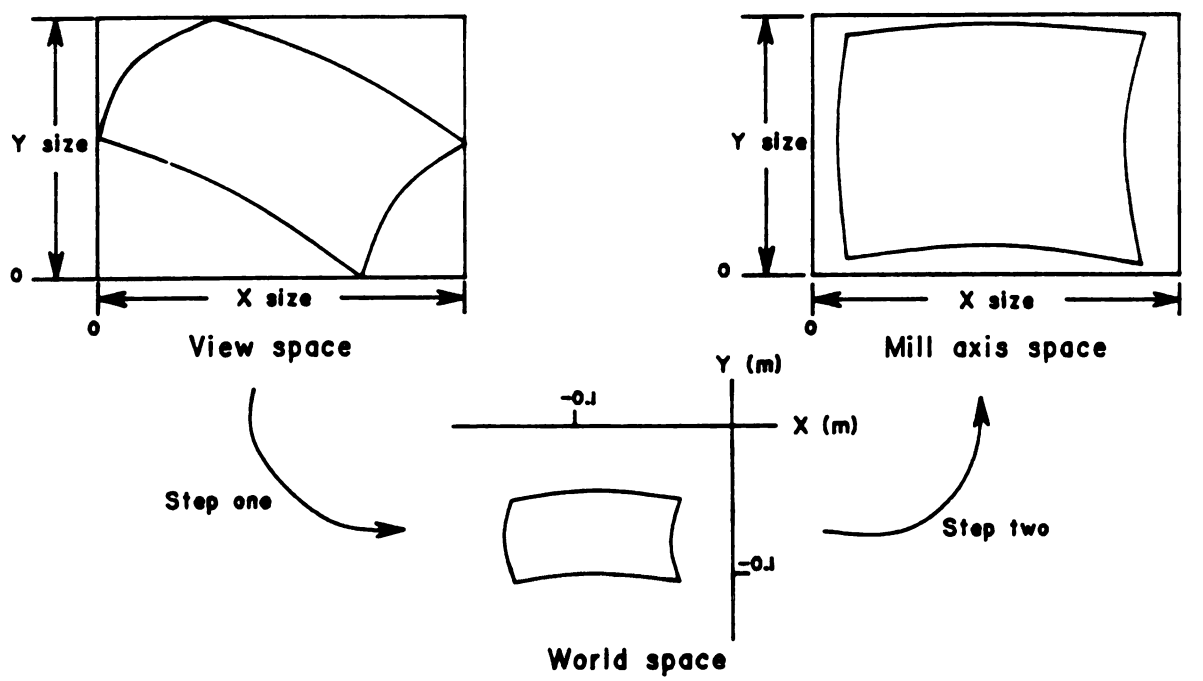


Figure 2. Conversion of discretized model coordinate data from view space to mill axis space

the desired orientation, but in doing so, it destroys the scan line coherence that it had in the view space in which it was defined. This point will be clarified in the following discussion.

The second step in the transformation is motivated by the need to access the view space coordinates of a pixel given its coordinates in the mill's eye view. This requires sorting the transformed data to create a list of pointers back to the view space coordinate arrays. The sorting process is straightforward if the transformed data can be accessed by positive integer indices. However, since the model may be defined relative to an arbitrary origin, and the dimensional units used to define world space may be numerically diverse (millimeters or feet, for instance), the world space equivalents of the pixel coordinate data are scaled and translated into a normalized frame of reference. A

reasonable choice for the basis of this scaling is the screen space subwindow size selected by the user.

Thus, mill axis space coordinate data are obtained from the two-dimensional world space version of the original pixel data via a scaling and translating transformation such that the resulting data are within the same physical dimensions as the original subwindow. For example, if the subwindow was defined as 200 pixels high by 300 pixels wide, then the pixel coordinate data would be mapped to a region in mill axis space which is 200 units by 300 units in area. This region in mill axis space will be called the "active area". These dimensions also define the range of the (integer) indices used to define a point in mill axis space. An addressable point in the active area of mill axis space will be called a bin, referenced by row and column indices. The mill axis space coordinates of a pixel are rounded to the nearest integer in assigning it to a bin.

An interesting consequence of this transformation is that multiple pixel coordinates from view space can map onto the same bin in mill axis space. An example of this case is shown in Figure 3. Here, the user has selected a reasonable view of a contouring operation, where the side of the tool is the primary cutting surface. When the pixel coordinate data are transformed into mill axis space, pixels which lie on the "wall" of the part model in a line parallel to the mill axis, will map onto the same bin. This condition, and the need to access the original pixel data given a bin number, require the development of a new data structure. The final portion of the preprocessing phase deals with the creation of this structure.

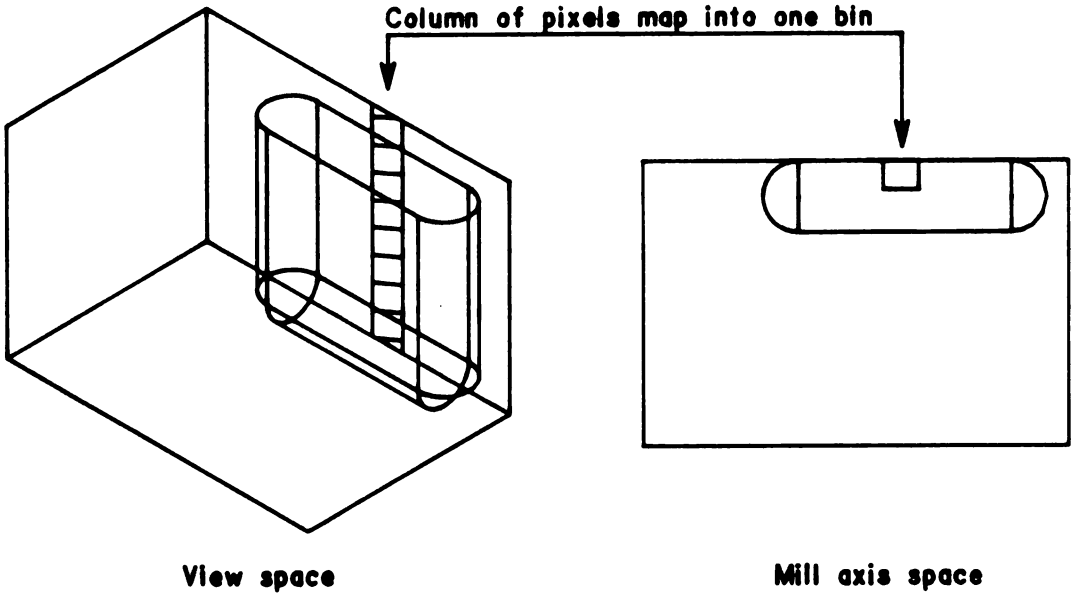


Figure 3 Multiple pixels in view space can map to the same bin in mill axis space

For each bin in mill axis space, the array COUNT contains an entry denoting the number of pixels which were mapped to it. Similarly, for each bin, an entry in the array START points to a location in an array called FNDPNT which contains pointers back to the original pixel data. If a bin is empty then the appropriate entries in both COUNT and START are both set to zero. To construct this data structure, the FORTRAN environment in which this work was developed requires several passes through the pixel coordinate arrays and the corresponding mill axis space bins. Since the implementation of this data structure is language specific, the details of its creation are omitted. Coordinate data transformation and creation of the corresponding data structure typically requires only a small fraction of the total preprocessing computation time.

4.1.3 Effects of Discretization Element Size

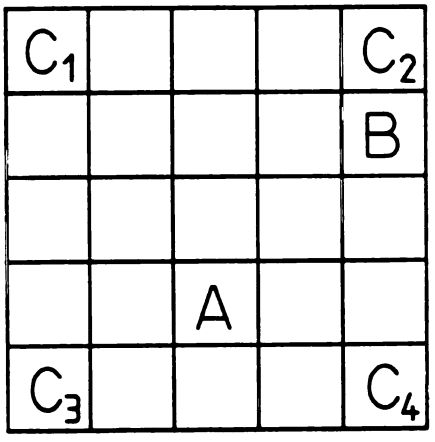
The data transformation and conditioning described above are modified somewhat if the user selects a discretization element size other than one. The purpose of this parameter is to reduce the size of the data set (surface coordinates and corresponding normals) to which each tool swept volume must be compare. Since the tool path is processed in mill axis space, the most direct way to accomplish this reduction is to transform only every M^{th} pixel on every M^{th} scanline into this space. This effectively imposes a grid or lattice structure upon the screen space image of the workpiece such that a cell of the lattice may be defined by four "corner" pixels. Cut values for pixels which are not mapped into mill axis space may then be interpolated using the (up to) four closest corner pixels, resulting in overall computational savings.

Note that this parameter does not affect the calculation of a normal vector at each pixel since calculation of these accurate normals is relatively inexpensive and they may be used in the output phase to enhance the resulting image with a simple light reflectance model. The discretization element size is used only to reduce the number of surface normals which participate in the most computationally intensive portion of the algorithm, vector/solid intersection.

Interpolation of cut values for pixels which lie within a lattice cell is simple if all four corner pixels map onto the surfaces of the workpiece. However, if a lattice cell lies near the edge of the model, so that it includes both background and part model pixels, the interpolation is not so clear. To handle this situation, each cell in

the lattice is processed individually. If the cell is completely full, i.e. all of the pixels in it map onto the part model, then only the four defining corner pixels are transformed into mill axis space. If the cell is partially full, i.e. it contains both background and part model pixels, then all the pixels which are not background are sent to mill axis space for precise intersection calculation. Obviously, if a cell is composed completely of background pixels, then none of its pixels are transformed.

Care is taken to avoid repetitive consideration of pixels which lie on the edges of cells since adjoining cells share common boundaries. So, pixels which lie near the edges of the workpiece model (in a given view) are processed via precise normal vector intersection, while those in the interior are "smoothed" by a linear interpolation based on the (up to) four closest corner pixels. Two examples of this interpolation are shown in Figure 4. The cut values of the corner pixels in the cell



$$C_A = 0.091C_1 + 0.091C_2 + 0.409C_3 + 0.409C_4$$

$$C_B = 0.750C_1 + 0.750C_2$$

Figure 4. Nearest neighbor interpolation for cut value smoothing

are denoted by C_1 through C_4 ; pixel A lies in the interior of the cell and pixel B lies on a cell edge. The cut value interpolating equation for both pixels is shown on the figure. This interpolation is only necessary, of course, if the discretization element size is set greater than one.

4.2 TOOL PATH PROCESSING

N/C geometric verification proceeds by following the tool path in mill axis space. A rectangular solid surrounding each swept volume (representing a tool motion) is projected onto mill axis space, resulting in a rectangular envelope which surrounds the projection of the actual swept volume. Each bin which lies within this bounded region is processed by locating the pixel(s) which mapped there and intersecting each pixel's normal vector with an appropriate (view space) model of the swept volume to calculate its cut value. When all the bins which lie within the rectangle have been processed in this manner, the next tool motion is considered. This procedure continues until all tool motions have been considered, i.e. until the CL-data file is completely processed.

The swept volume envelope which is projected onto mill axis space is enlarged to account for the tolerance (T_{out}) and the range of interest (R_{int}) selected by the user. However, this entire volume is never actually calculated; instead, its projection is calculated directly from the world space coordinates of two consecutive tool path points. Since the mill tool is assumed to remain parallel to the world space Z axis, only the X and Y coordinates of the two path points need

be considered. The four corner points of a rectangle enclosing the "swept area" can be constructed using the coordinates of the path points and a length, called the "swept area range" or R_{sa}, equal to the sum of the tool radius, T_{out}, and R_{int}. The rectangle (defined by its corner points) is then converted into mill axis space with the same scaling and translating factors used in the second step of the pixel data transformation described above.

If the swept area were constructed by simply projecting the entire swept volume at each step, then consecutive swept areas would overlap an area of as much as $(2R_{sa})^2$, causing many bins to be considered redundantly. Alternatively, if the swept area considered only the area which lies strictly between tool path points, then the overlap area would be small but significant portions of the surface would be neglected at each tool motion. The technique used here falls between these two extremes; it results in a swept area of sufficient size to include all portions of the surface that could be affected by a tool motion while minimizing the swept area overlap of consecutive tool sweeps. This is accomplished by exploiting the fact that the tool occupies the same space at the terminal position of the current sweep as it does in the initial position of the next sweep.

Construction of this "minimal" swept area is shown schematically in Figure 5. The tool path vector, τ_i, is calculated from the difference in the world space coordinates of two consecutive path points, t_i and t_{i+1}. Dividing the components of τ_i by its length, L_p, yields the normalized path vector, λ_i. The unit vector, γ_i, is normal to the plane defined by τ_i and the unit mill axis vector, ζ.

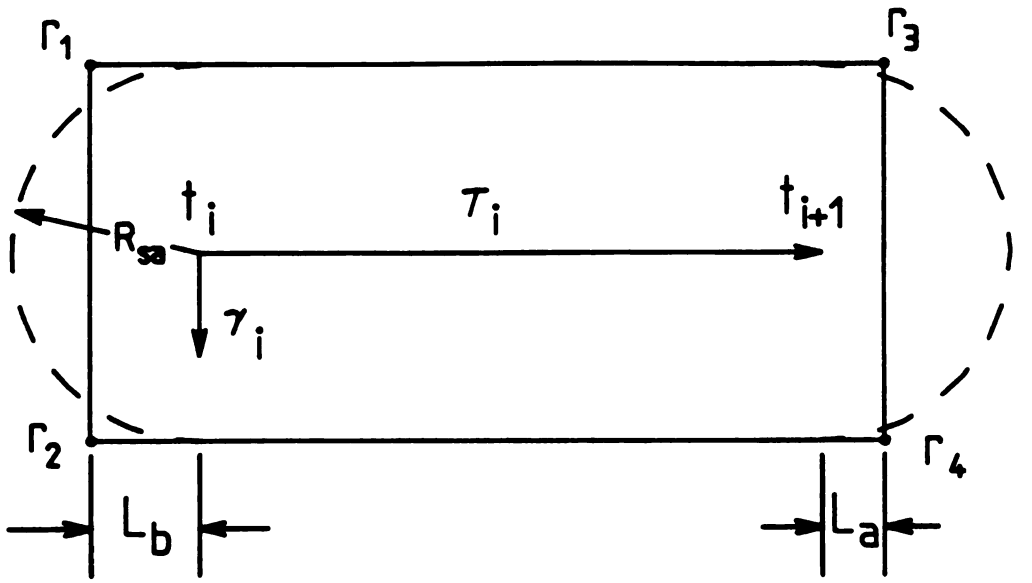


Figure 5. Construction of the swept area in mill axis space

The crucial parameters in this construction are the "look ahead distance", L_a, and the "look back distance", L_b. The look ahead distance is based on the angle between the current path vector and next path vector; the look back distance is based on the angle that the current path vector makes with the mill axis and the angle between the current and previous path vectors. Details of this calculation are given in Appendix A. The computational cost of calculating L_a and L_b is quite small: two vector dot products and two intrinsic function calls for each tool motion. However, this cost is more than offset by the significant reduction in the redundant and/or unnecessary consideration of bins (and hence pixels) which lie in the vicinity of tool path points.

Although they are based on three-dimensional world space data, L_a and L_b are calculated as projected onto a plane which is perpendicular to the mill axis. By neglecting the Z (mill axis) coordinate, a similar

projection is applied to the (three-dimensional) world space tool path data, t, λ, and γ, in preparation for construction of the swept area. Given L_a, L_b and the projected two-dimensional equivalent path points and vectors, the four points which define the swept area may be defined (using the terminology introduced above) as follows:

$$r_1 = t_i - L_b\lambda_i - R_{sa}\gamma_i$$
$$r_2 = t_i - L_b\lambda_i + R_{sa}\gamma_i$$
$$r_3 = t_{i+1} + L_a\lambda_i - R_{sa}\gamma_i$$
$$r_4 = t_{i+1} + L_a\lambda_i + R_{sa}\gamma_i$$

The points r_1 through r_4, defining the rectangular swept area envelope in a two-dimensional projection of world space, are then transformed into mill axis space with the same scaling and translating factors used in the second step of the pixel data transformation (see Figure 2).

Some results of this technique are depicted in Figure 6 which illustrates the effect of the look ahead distance as applied to three typical cases of sequential mill tool motions. The shaded regions represent the swept areas for each tool motion; the swept area for the current motion is surrounded by a solid line while the swept area associated with the next tool motion is surrounded by a dashed line. The cross-hatched areas are overlap regions considered in both swept areas. Figure 6a shows the optimum case where the tool does not change directions and there is no swept area overlap. Figures 6b and 6c depict progressively worse cases in which the size of the overlap region increases proportional to the change in mill direction.

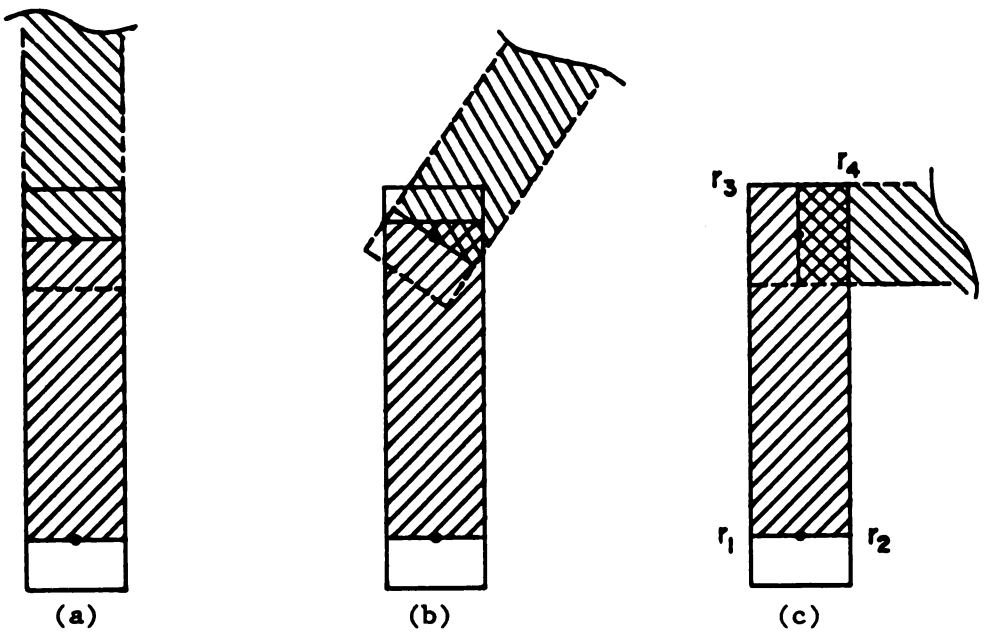


Figure 6. Three cases of consecutive swept areas showing varying degrees of overlap

Given a swept area in mill axis space, the algorithm proceeds by processing the bins which fall within it. However, if the swept area falls outside of the active area, as shown in Figure 7, then this motion of the mill is disregarded since its affect on the part model is invisible in the view selected by the user; the next tool motion would then be considered. If any of the points defining the swept area fall within the active area, then the points are sorted by their mill axis space Y coordinate from smallest to largest. Linear equations which define the bounds of the swept area in mill axis space are calculated such that, given a Y value they return an X value. Each of these lines divides mill axis space into a two-dimensional "half space", such that when considered together, they form a logical boundary of the swept area envelope.

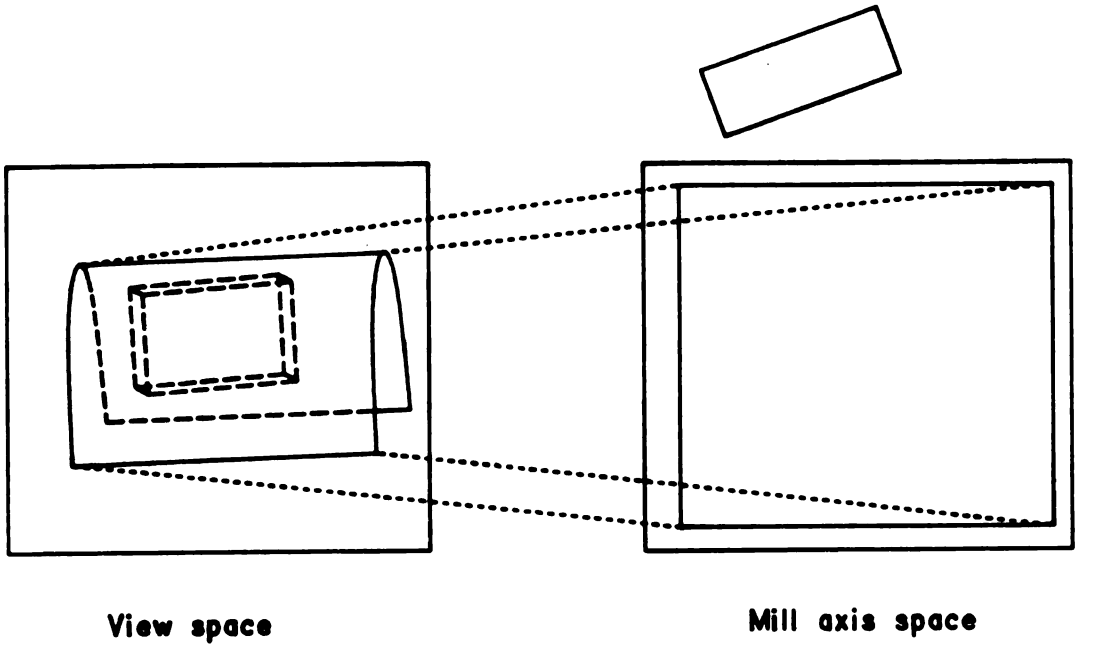


Figure 7. Example of a swept area which falls outside of the active area in mill axis space

The bins within the swept area are processed from the corner with the smallest Y value (rounded down to the nearest integer) to the corner with the largest Y value (rounded up). Nested within this Y loop is a loop on X where the starting and ending X value is determined from the bounding linear equations. The rounding required to access the bins at the boundaries of the swept area is always done in a conservative manner; leading edge X values are rounded down, while trailing edge X values are rounded up. If a swept area lies partially outside of mill axis space, then the limits of the X and/or Y loop are dictated by the limits on the active area of mill axis space.

As each bin in the swept area is accessed, its COUNT value is checked to see if any pixels mapped onto it. If the COUNT value is nonzero, then pixel and normal data are accessed through the pointers START and FNDPNT. The normal vector for each pixel in the bin is

intersected (in view space) with an appropriate solid model of the swept volume. Initially, all pixels sent into mill axis space are assigned a cut value equal to the sum of the tool radius, the tolerance T_{out}, and the range of interest, R_{int}. If, during processing, a normal intersection is calculated yielding a cut value which is less than the one previously stored for that pixel, the new value is saved, overwriting the previous value. This process continues until all the pixels in all the bins within the swept area have been processed. A new swept area is then calculated and the process continues until the entire CL-file has been considered. Thus, on completion of processing, only the deepest excursion of the mill toward or into the surface is saved for each pixel.

4.2.1 Vector/Solid Intersection

The intersection of a surface normal vector with a mill tool swept volume is calculated efficiently by modeling the swept volume solid with progressively more precise collections of bounding surfaces. The calculation begins by modeling the swept volume as a parallelepiped. If necessary, the model is refined, but only in the region where the vector intersection could possibly occur. More precise bounding surfaces, (cylindrical and spherical) are added to the model, in the region of the planar intersection, if the parallelepiped model yields a cut value which is less than the maximum miss, L_{out} (as defined in Chapter One).

Figure 8 depicts the smallest parallelepiped which surrounds a tool swept volume. Three vectors are required to define the six planes which make up the parallelepiped. The first one, γ, introduced in the

previous section as the vector normal to the plane defined by the tool path vector, τ, and the mill axis vector, ζ, is used to construct the side planes of the parallelepiped. The cross product of ζ and γ results in the vector normal to the end faces of the parallelepiped, and the cross product of γ with the path vector, τ, yields a vector normal to the top and bottom faces. Cases where the path vector is coincident with the mill axis vector are handled as a special case. In constructing the top plane of the parallelepiped, the angle that the path vector makes with the mill axis (lift or dive) is checked to insure that the volume includes the flat "top" of the mill (refer again to Figure 8).

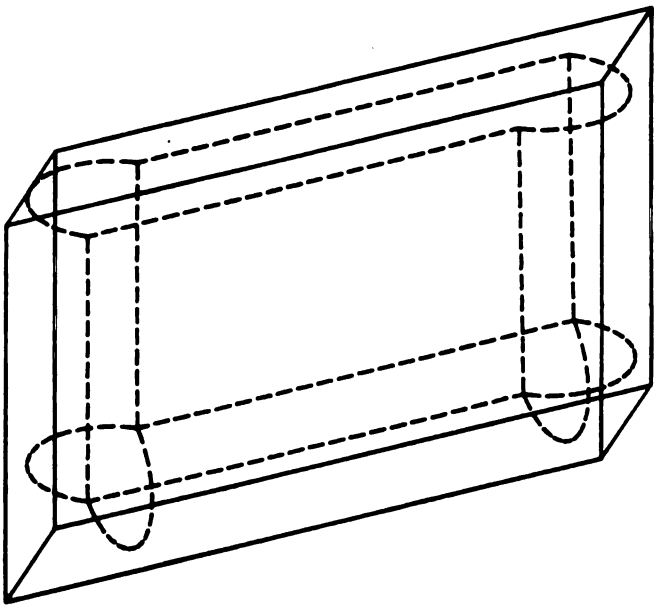


Figure 8. Parallelepiped surrounding a tool swept volume

A distance, L_{int}, is calculated from the surface pixel, along the normal vector, to each of the six planes which bound the parallelepiped. The formula for this intersection calculation is described in Appendix B. If none of the L_{int} values is less than the maximum miss, L_{out}, then

processing for the pixel ends and the next one may be considered. Otherwise, the subset of intersection distances which are less than L_{out} is sorted from smallest to largest.

The sorted list of possible vector/plane intersections is processed to determine the first intersection which takes place within the plane boundaries defining the parallelepiped. This check is accomplished with a modified version of the Roberts algorithm, which was originally applied to three-dimensional hidden line removal. [42] In this application, the algorithm is simplified to two dimensions, and used to determine if a point on a plane lies within four bounding half-spaces (i.e. on a face). The parallelepiped is constructed so that the normals of each plane point toward the interior of the volume. Dot products are calculated between the vector representing the intersection point (in homogeneous coordinates) and vectors containing the coefficients of the bounding planes. (See Appendix C for a numerical example of this application of the Roberts algorithm.)

If any of the four dot products yields a negative value, the intersection point is outside the face boundaries. The planes are processed in order, i.e., the ones with the smallest L_{int} values are considered first. If there is an intersection within a face, then the normal vector at this pixel must be considered further; if not, the next plane in the list is checked, until the list of eligible planes is exhausted. If none of the plane intersection points falls within a face, then processing for the pixel is terminated and the next available one may be considered. Also, if the top of the parallelepiped is found to be the closest intersecting plane, the user is warned of possible

tool or mill interference with the workpiece, and processing for the pixel ends.

Since the list of eligible planes is sorted by normal vector intersection distance, from smallest to largest, the first one which yields an intersection within a face must be processed further to determine if a more accurate swept volume model is necessary. This is accomplished with further applications of the Roberts algorithm, this time with bounding planes moved inside the swept volume to represent the transitions from planar to cylindrical or from cylindrical to spherical surfaces.

For example, Figure 9 shows a side face of the parallelepiped surrounding a swept volume, divided into subregions labeled 1 through 8. The Roberts algorithm described above is performed with the intersection point and the planes which bound subregion 1. If the intersection point

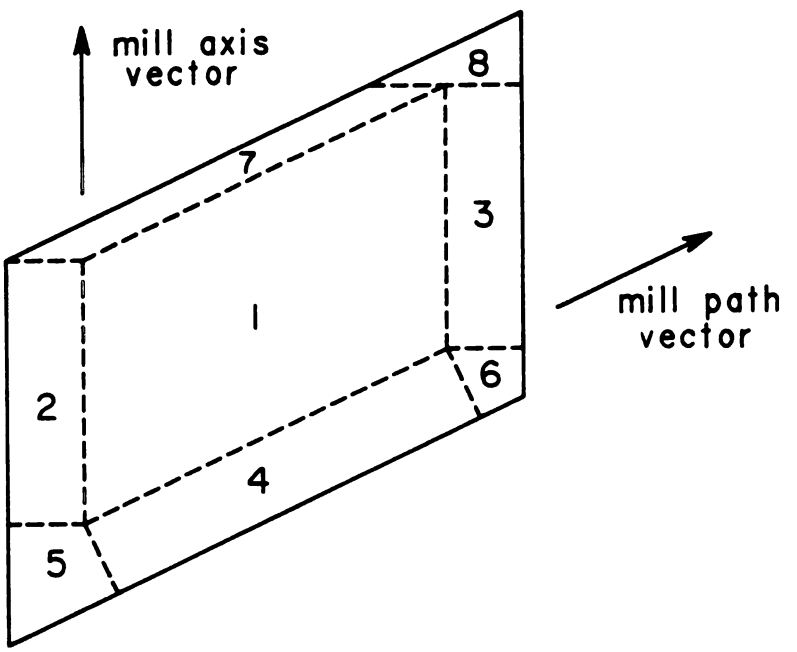


Figure 9. Subregions of a side face of a parallelepiped which surrounds a mill tool swept volume

falls within this subregion, processing for this pixel ends, since the planar intersection is already exact. If it is not within subregion one, the dot products can be interpreted to determine which subregion should be examined next. For instance, if the plane separating subregion 1 and subregion 4 yields a negative value, then another application of the Roberts algorithm with the planes which bound subregion 4 will determine if the intersection took place within subregions 4, 5, or 6. At most, two applications of the Roberts algorithm are sufficient to determine the subregion of the vector intersection for this (side) face of the parallelepiped.

Referring again to Figure 9, if subregions 2, 3 or 4 bound the intersection point, then the normal vector may intersect a cylindrical surface of the mill swept volume. If subregions 5 or 6 bound the point, a spherical surface must be considered. If the normal vector intersects the plane in subregion 7, the user is warned that the part may have been cut with the rearward facing top edge of the tool (or forward facing if the mill dives), and processing for the pixel is terminated. Similarly, if subregion 8 bounds the point, the normal has missed the actual swept volume and processing terminates. Note that subregions 7 and 8 do not exist if the mill does not change its Z coordinate (height) during a motion. An analogous (although somewhat simpler) procedure is applied if the intersecting plane is an end face or the bottom face of the parallelepiped, except that only cylindrical or spherical subregions are possible.

Intersection of a vector with a sphere or cylinder requires a relatively straightforward application of vector algebra. Details of these vector/surface intersection calculations are given in Appendix D.

The calculations are not computationally intensive, but are considerably more complex than the planar approximations used up to this point.

The data which are sent to the precise surface intersection routines depends on which subregion of the parallelepiped face has been pierced. For instance, along with the surface point and normal vector under consideration, the routine for cylindrical surface intersection requires a point (vector) defining the base of the cylinder and a vector defining the principal axis of the cylinder. Using the example shown in Figure 9, if subregion 2 is pierced, the base point is the initial tool path point (for the current swept volume) and the principal vector is the mill axis. For subregion 3, the principal vector is the same but the base point is the terminal tool path point. Similarly, if subregion 4 is pierced, the base point is the initial path point and the principal vector is the tool path vector.

A normal vector can pierce the parallelepiped in a cylindrical or spherical subregion yet miss the actual volume completely. This condition can be determined without calculating the intersection point, so is checked first in the initial portion of the exact intersection calculation, to further reduce unnecessary computations.

4.3 POSTPROCESSING

The output image of the sculptured surface part, as milled by the tool path, is generated using the cut value and normal vector saved at each pixel. The color of each pixel is made up of a hue and an intensity. As described above, pixels on the outside of surfaces in the

current view are assigned a hue based on the cut value. Areas of the surface cut within tolerance are assigned the hue green. Overcut (gouged) areas are shaded via hue interpolation between red and yellow, where red represents "just beyond tolerance" (cut < - T_{in}) and yellow represents "maximum gouge" (cut < L_{in}). Undercut (missed) areas of the model are shaded via hue interpolation between blue and magenta, where blue represents "just beyond tolerance" (cut > T_{out}) and magenta represents "maximum miss" (cut > L_{out}). When the user selects a range of interest, R_{int}, equal to zero, information regarding the "degree of" miss or gouge is no longer applicable, so that overcuts are assigned the hue red and undercuts blue. Pixels with normal vectors which point into the screen represent areas of the surface which are viewed from the inside and are shaded with black.

Independent of hue, the intensity at each pixel is based on the angle between the normal vector and a user-defined light source. This calculation is inexpensive, since the normals are necessary for, and available from, the cut depth calculation. The intensity calculation greatly enhances surface feature recognition. Thus, two pieces of information are combined in the same image: the desired surface model and the effect of the mill on it.

CHAPTER V

APPLICATION EXAMPLES

5.0 APPLICATION OF THE N/C GEOMETRIC VERIFICATION ALGORITHM

In Chapters Three and Four a new algorithm for N/C geometric verification was described. This chapter deals with the application of the algorithm to realistic industrial parts. Three example applications are described. First, a portion of a simple manufactured part is considered to illustrate the interpretation of the graphical output. This example also demonstrates verification of contour milling operations and the incorporation of holding fixtures as part of the workpiece model. Next, a part from the aerospace industry is considered. In this example, several test runs are made illustrating the effects of the user-selectable parameters (i.e., range of interest and discretization element size) on both graphic output quality and overall computation time. Tradeoffs between the conflicting goals of rapid computation time and high quality output are discussed, as are the limitations of the user-selectable parameters. The third example, from an automotive application, demonstrates how the algorithm could be applied in an actual CAM environment. A problem area on the desired

part is located in a global view of the entire part, then a more precise zoomed view of the area is examined.

All application examples were run on a Prime 750 superminicomputer with 6 megabytes of physical memory; results were displayed on a Tektronix 4129 color terminal. In all of the application examples the computation time for the preprocessing phase of the algorithm, i.e., calculation of surface point coordinates and normals, is reported separately from the computation time required for N/C geometric verification. This is done because preprocessing time is a fixed cost (independent of the size of the CL-data file) and the verification procedure is essentially independent of the method used for surface discretization and generation of normals. For example, a more traditional polygon tiling of the surface could be used, resulting in faster computation time due to the calculation of (generally fewer) normal vectors.

5.1 APPLICATION ONE: CONTOURING OPERATION

Although it could represent an actual manufactured part, the part model (and associated N/C program) considered in the first application example was created by the author for demonstration purposes. This example features N/C geometric verification of a contouring operation on a workpiece model that includes two holding fixtures. The part is essentially a three inch thick slab of material with one corner rounded into a three inch radius cylindrical surface. The part model (including holding fixtures) consists of 14 rational B-spline surfaces; the associated CL-data file contains 19 points. A spherical end tool of

radius 0.5 inch and height 3.0 inches was used to mill the contour. The user selectable parameters were set as follows: discretization element size equal to one, milling tolerances equal to 0.01 inch (inside and outside), and range of interest equal to 0.49 inch.

Results of the N/C geometric verification algorithm as applied to this model are shown in Figure 10. Since the N/C program only mills (contours) the sides of the part, the entire top of the slab appears as missed by at least the maximum amount, 0.5 inch. The walls or sides of the part appear to be milled within tolerance except in the area of the cylindrical surface where both slightly missed and gouged regions are apparent. In this example, the circular arc tool path necessary to mill this area precisely was approximated by eight linear segments. The errors were intentionally introduced to show the verification

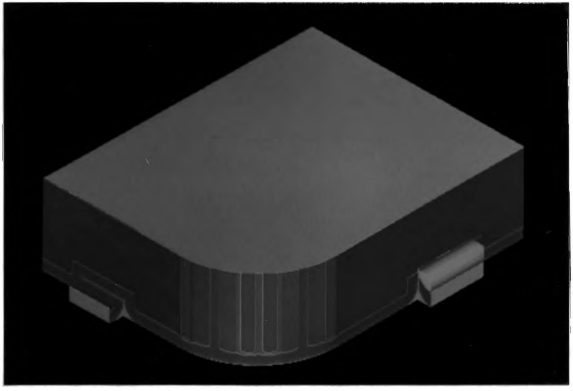


Figure 10. N/C geometric verification of a contouring operation

capability. The band, ranging from blue to magenta, around the bottom of the part shows the effect of milling a corner with a ball end tool.

Figure 10 also shows the effect of the mill on holding fixtures. Both fixtures extend 0.5 inch out from the sides of the part. The fixture on the lefthand side of the part measures 0.5 inch in height, while the one on the right has a height of 1.0 inch. The mill successfully avoids the fixture on the lefthand side since it is shaded in blue and magenta with only a small green area that was affected within the tolerance limit. However, on the righthand side of the part, the tool interferes with the fixture. Note that the fixture is shaded as modeled (a planar box), but that its yellow hue indicates a severe gouge.

The computation time required for this application example was approximately 25 CPU minutes for the preprocessing phase and 19 CPU minutes for N/C geometric verification.

5.2 APPLICATION TWO: TURBINE BLADE

The second N/C geometric verification example considers a faulty part program (generated by N/C software which was then under development) which was used to machine an actual prototype. The desired part in this example is modeled by a single rational B-spline surface patch representing the convex side of a turbine blade. The part is approximately 1.75 inches wide by 3.5 inches long. It is to be milled (on a three-axis milling machine) with a ball end tool of radius 0.25 inch and height 1.0 inch. The CL-data for this part consists of 3034

distinct positions of the mill tool. In each of the example runs, the milling tolerance was $T_{in} = T_{out} = 0.001$ inch.

Results of the N/C geometric verification of this N/C program are shown in Figures 11, 12 and 13; associated computation times are given in Table 1. Figure 11 shows the results of Case 1, the most accurate, and hence, the most computationally intensive run. In this case, the discretization element size was set to 1 (i.e. every pixel was considered in the intersection calculation) and the range of interest was set to 0.01 inch. Figure 12 shows results of Case 2; the same conditions as above, but with the range of interest set to zero. Finally, for Case 3, the results of the same part in the same view, with discretization element size set to 4 and range of interest set to 0.01 inch, are shown in Figure 13.

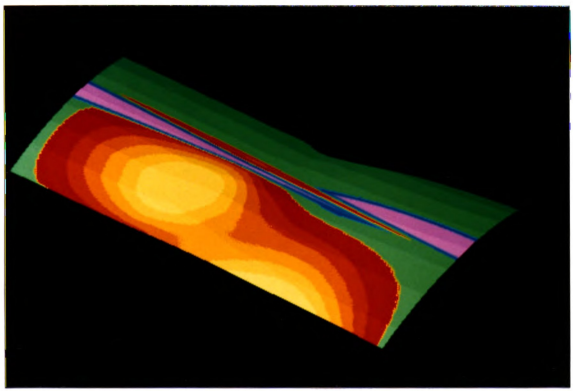


Figure 11. Turbine blade N/C geometric verification, Case 1

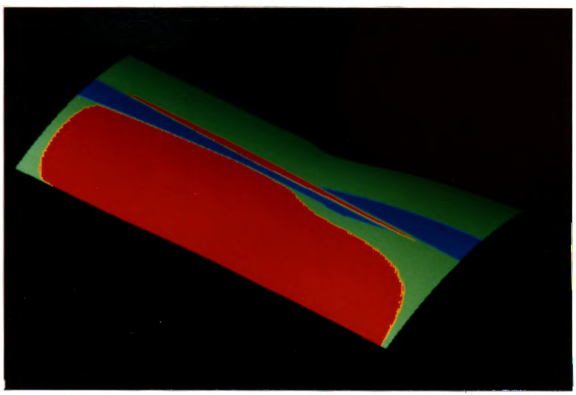


Figure 12. Turbine blade N/C geometric verification, Case 2

It should be noted that the "color banding" apparent in Figures 11 and 13 is a result of the limitations of the display terminal used for this work. The terminal is limited to display at most 256 colors at a time. When the user selects a nonzero range of interest, 15 hues are used to depict depth of cut (7 each for miss and gouge and one for "in tolerance"). The color limitation results in only 16 intensity levels being available for each hue. The result shown in Figure 12 appears smoother because only three hues are used if the user selects a range of interest equal to zero. For this case, there are 80 intensity levels available for each hue.

Table 1

Computation Time in CPU Minutes for N/C Geometric Verification
Turbine Blade Example

Task -->	A	B	C	D	E	Total
Case 1	1.4	59.8	132.7	159.3	50.8	404.0
Case 2	1.5	61.4	130.7	155.0	47.1	395.7
Case 3	1.3	10.7	15.2	18.6	5.4	51.2

Case 1 ==> Discretization element size = 1
Range of interest = 0.01 inch
Case 2 ==> Discretization element size = 1
Range of interest = 0.0 inch
Case 3 ==> Discretization element size = 4
Range of interest = 0.01 inch

Task A: transform pixel data into mill axis space, sort, construct swept area, construct parallelepiped and auxiliary planes
Task B: process swept area, retrieve view space coordinate and normal data given mill axis space bin addresses
Task C: intersect normal with planes of parallelepiped
Task D: apply parallelepiped face boundaries for face determination, calculate plane subregion
Task E: intersect normal with cylinder or sphere

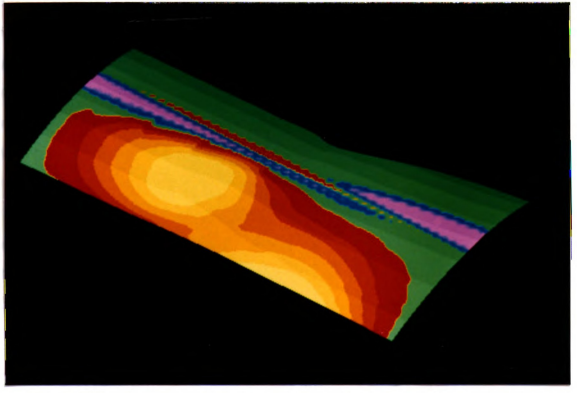


Figure 13. Turbine blade N/C geometric verification, Case 3

Table 1 summarizes the results of these three cases and breaks the total computation time down by function. In each case the total CPU time spent to calculate the surface normal vectors was about 21 minutes.

These results show the dramatic effect that the discretization element size has on the performance of the algorithm. Computation time was reduced by a factor of approximately 8 when the discretization element size was changed from 1 to 4. Of course, this change has an adverse effect on the resolution and accuracy of the resulting output image, but it is still quite apparent that the improper milling of the part would be discovered in this case.

Reducing the range of interest also cuts computation time although not nearly as dramatically as the discretization element size change. This small change is expected since this reduction of R_{int} results in a swept area envelope size reduction of only about 4 percent. This example shows that the computational price for the "degree of" miss or gouge information is really quite small. The effect of the discretization element size is much more profound since it actually reduces the size of the data set (pixels and normals) with which the tool path must be compared.

Variation of the range of interest and discretization element size, along with prudent view selection, can be applied simultaneously to increase computational efficiency. Alternatively, if the user is prepared to pay the computational price, he can opt for precision limited only by the resolution of the display device.

5.3 APPLICATION THREE: AUTOMOBILE HOOD

This example illustrates the potential use of the N/C geometric verification algorithm in an actual CAM environment. The workpiece model represents one half of a stamping die for an automobile hood. The model is composed of eight rational B-spline surface patches. Two CL-data files are used to mill the part. The first contains 4972 points and operates with a ball end tool of radius 1.0 inch and height 3.0 inches. The second CL-data file contains 1112 points and its smaller tool, 0.125 inch radius by 0.5 inch height, is used for more detailed finishing work.

In an actual production environment, a part programmer might first wish to see the entire part as affected by the N/C program. Such a run is depicted in Figure 14. The centerline of the automobile runs along the upper lefthand edge of the model in this figure, while the front of the car is at the upper righthand edge. In this example, T_{in} and T_{out} were set equal to 0.005 inch, the range of interest was set to 0.045 inch, and the discretization element size was set to one. The band of magenta colored surface along the lower edge of the model shown in Figure 14 represent portions of the model which are trimmed away in the actual design database; they are remnants of the design process, not part of the model and are not meant to be cut.

The computation time required to produce this output was approximately 33 CPU minutes for preprocessing and 285 CPU minutes for N/C geometric verification. When compared to the computational performance of the previous example, these results may appear spurious. Although the total size of the CL-data in this example is approximately

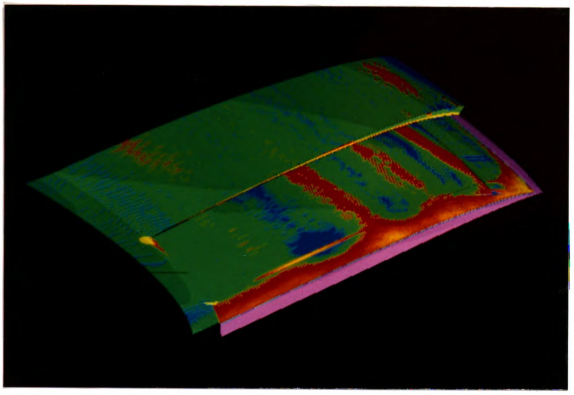


Figure 14. Global view of automobile hood N/C geometric verification

twice as large as that of the previous example, the computation time is approximately 25 percent less. This result is probably due to the fact that, in these two examples, the average number of pixels considered for each tool motion is quite different. Assuming the number of pixels on each model is approximately equal, since the automobile hood model is much larger than the turbine blade, the distance between pixels is much larger, resulting in fewer pixels being considered for a given motion of the mill. Thus, although the hood example appears more efficient computationally on a per tool motion basis it is, in this view, less accurate than the turbine blade example.

Upon examination of the global view of the N/C geometric verification of the hood model, the part programmer may wish to take a

closer look at some critical areas of the model. One such critical area, as evidenced by the band of yellow, is the crease running the length of the hood roughly parallel to the centerline of the car. Figure 15 shows the results of an N/C geometric verification run on a zoomed view of the hood featuring this crease. In this example a subwindow around the feature of interest (the crease) was selected to reduce computation time. Preprocessing for this example took about 16 CPU minutes while N/C geometric verification required approximately 187 CPU minutes.

Many features of the milled surface which are barely visible in Figure 14 are visible in much more detail in Figure 15. With this

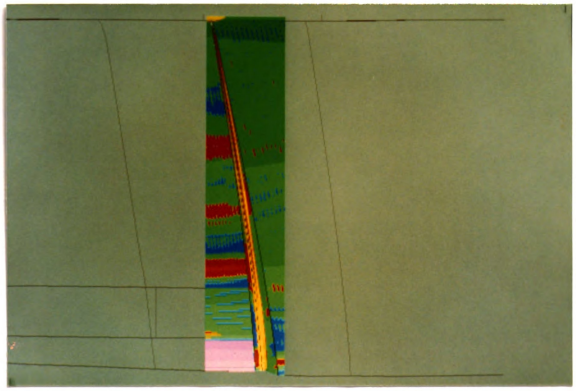


Figure 15. Zoomed view of automobile hood N/C geometric verification

information the part programmer could return to the tool path generation phase and attempt to remedy the problems in the N/C program. More detailed zoomed views of other flawed areas could be also be run.

CHAPTER VI

ALGORITHM ANALYSIS

6.0 PERFORMANCE ANALYSIS OF THE N/C GEOMETRIC VERIFICATION ALGORITHM

In the preceding chapter, several application examples of the N/C geometric verification algorithm were presented and discussed. Some of the example cases presented were said to be relatively efficient computationally. This chapter serves to support this claim by presenting a performance analysis of the algorithm. First, an order of complexity analysis is developed and discussed. This analysis provides an upper bound on the execution time of the algorithm based on the input to it. These results are then compared to a similar order of complexity analysis which has been developed for a solid modeling approach to N/C geometric verification through N/C simulation. The algorithm presented here for direct N/C geometric verification is shown to be much less demanding computationally than the solid modeling approach, especially as the number of tool path points (i.e. size of the CL-data file) increases.

6.1 ORDER OF COMPLEXITY ANALYSIS

Recall from Chapter Two the distinction between N/C geometric verification and N/C simulation. The original solid modeling approach to the problem can provide N/C geometric verification but only after completion of full N/C simulation followed by computationally intensive "null object" calculations. The algorithm presented in this dissertation differs dramatically from the solid modeling approach; it addresses the N/C geometric verification problem directly, reducing the complexity of the problem by avoiding the constraints imposed by N/C simulation. Intuitively, direct N/C geometric verification should afford an efficiency improvement over the solid modeling approach. Whereas the solid modeling approach is concerned with calculation of many intermediate solids, representing various stages of the "as milled" part and the material removed from it, direct N/C geometric verification deals only with the final "desired part" surface model and its interaction with the tool path. In this section the performance of the direct N/C geometric verification algorithm is quantified, with a standard technique for algorithm analysis, so that it can be compared to that of the solid modeling approach.

To guarantee accurate results, any algorithm that provides N/C geometric verification must consider each motion of the tool regardless of whether it is intended as a rough or final cut. Assuming that the algorithm treats each tool motion similarly, one would expect its computation time to be strongly dependent on the number of tool motions or the number of points in the CL-data file (as is actual milling time). The following analysis deals with the nature of this dependency.

Comparison of different algorithms according to relative computational efficiency requires definition of a standard measure which is independent of the hardware and software used to implement the algorithms. Typically, algorithm performance is measured and compared through the use of an order of complexity or "big oh" analysis. [45] The result of such an analysis is a function called the time complexity of the algorithm, T(n), which is proportional to some function of the size of the input data set "n". In this work, T(n) will be categorized by calculating a bound, C(f(n)), on the time complexity; i.e., the "worst case" is characterized by the inequality $T(n) \leq C(f(n))$, where C is a constant of proportionality and f(n) is called the "growth rate".

The time complexity analysis for the direct N/C geometric verification algorithm will proceed by considering first its growth rate and then its constant of proportionality.

6.1.1 Growth Rate

For the purposes of this analysis, the N/C geometric verification algorithm may be depicted in "pseudo-code" as shown in Figure 16. The algorithm is shown broken down into nested tasks. The time complexity of each task is denoted by the term "O(g(n))", where g(n) is the growth rate of the individual task. The term O(1) indicates that the task has constant order time complexity; i.e., it is independent of input size. Each task in the algorithm will be examined individually to determine the growth rate of the algorithm as a whole. The reader may refer to Chapter Four for precise details on each task.

```
sort.pixels.into.bins  O(1)
For every CL-point from 1 to N
    make.swept.area  O(1)
    make.box  O(1)
    For every bin in swept area from 1 to BINSA
        getpix  O(1)
        For every pixel in bin from 1 to PIXBIN
            boxint  O(1)
            exactint  O(1)
        EndFor
    EndFor
EndFor
```

Figure 16. Pseudo-code representation of N/C geometric verification algorithm.

The first task in the algorithm, sorting the pixels into mill axis space bins, depends only on the number of pixels/normals participating in the exact intersection calculation. This term is affected by user view and subwindow selection as well as the discretization element size, but is independent of the size of the CL-data file, N. The sort is done only once, before path processing begins and therefore is of constant order time complexity. Each of the remaining tasks operates inside of the loop which considers every tool path point.

The swept area creation task is of constant order; it involves calculation of four two-dimensional points given a tool position (as shown in Appendix A). If the swept area overlaps the active area in mill axis space then the parallelepiped which surrounds the swept volume

is constructed. This task entails calculation of nine planes, six for the parallelepiped itself and three auxiliary bounding planes. Since the normals of these planes are already known, this task reduces to a calculation of nine constants; thus it is also of constant order time complexity. The following tasks are undertaken for every bin in the swept area which is also in the active area of mill axis space.

The task called "getpix" entails a manipulation of the data structure used to correlate view space with mill axis space. More specifically, given a bin address, getpix looks up the number of pixels mapped there, then uses the linked list data structure to retrieve the address of the pixel (and normal) data in storage arrays. This task is also of constant time complexity. The following tasks are then applied for each pixel in the bin.

The task "boxint" refers to intersection of a (normal) vector with six planes which bound the parallelepiped, a task of constant time complexity. (Details are shown in Appendix B.) Typically, only a subset of the six planes is considered further in the next task "exactint". This task entails application of the Roberts algorithm (Appendix C) to determine whether the vector intersection occurs within a planar face; if successful, another application of the Roberts algorithm to determine the subregion; and finally, if necessary, intersection of the normal with a cylindrical or spherical surface model (Appendix D). Even if all six planes must be considered and each normal vector passes through to the most complex surface intersection calculation, the entire task is still of constant time complexity.

Of the three control loops shown in Figure 16, only the outermost loop depends on the size of the input. The limits on the two interior loops, the number of bins in a swept area, BINSA, and the number of pixels per bin, PIXBIN, depend on such factors as view and subwindow selection, discretization element size, range of interest, and the tool motion (CL-point) under consideration. BINSA and PIXBIN are independent of the size of the input (the number of CL-data points) and can be bounded. Thus, since each of the tasks within the outermost loop is of constant time complexity, the growth rate for entire N/C geometric verification algorithm is O(N), where N is the number of tool path points.

6.1.2 Constant of Proportionality

The above development shows that the time complexity of the N/C geometric verification algorithm is bounded by a linear function of the number of tool path points in the N/C program. This is sufficient to describe its asymptotic behavior. However, calculation of a bound on the rate of that growth (i.e. the slope of the line), is a more difficult task. Each of the O(1) tasks shown in Figure 16 has an associated constant. An operation count on each task could serve to estimate these constants, but the limits on the interior loops, BINSA and PIXBIN, are difficult to quantify. A total operation count on an algorithm of this size and complexity would be very difficult. Furthermore, it would be of little use since none of the alternative algorithms for this task have similar analyses published. The purpose of this section is to quantify the dominant constant in the algorithm.

To get some idea of the magnitude of this constant of proportionality, a "worst case" application of the algorithm will now be examined.

Consider a contour type milling operation similar to the one shown in Figure 3. Suppose a user selects a view perpendicular to both the mill path and the "wall" being milled. Also, suppose that the view is zoomed sufficiently so that the entire screen is filled with pixels which map onto the "wall" and a single mill motion proceeds from outside of the screen on the lefthand side, across the entire screen to a point outside on the righthand side. Futhermore, allow that the subwindow covers the entire screen, the discretization element size is set to one, and the range of interest is set to a large value. Finally, suppose that the tool path consists of multiple rough cuts parallel to the "wall" such that initial tool sweeps are far from the "wall" and each sweep gets progressively closer to it.

In this scenario all of the pixels on the screen will map into one row of bins in mill axis space. If the screen is SCRSIZ pixels in width by SCRSIZ pixels in height, then each of the SCRSIZ bins in this row will contain SCRSIZ pixels. Assume that the range of interest is large enough that this row of bins is included in the swept areas generated by each and every tool motion.

Consider again the two interior loops of the algorithm as shown in Figure 16. Given the case described above, since every pixel is considered for every tool motion, it is reasonable to combine these interior loops into one loop which has as its limit the product of the limits of the original two. In other words, the product of the number of bins per swept area BINSA and the number of pixels per bin, PIXBIN

results in a value for the number of pixels per swept area, PIXSA. The maximum value that PIXSA can take is the maximum number of pixels, $SCRSIZ^2$.

The above discussion is intended more as a thought experiment than as a rigorous analysis of the worst case proportionality constant. An analysis at this level, however, is sufficient for comparison of this algorithm to other techniques. Verification of an actual milling operation in the above described worst case scenario would represent a very inefficient use of the algorithm. In actual practice, the number of pixels considered per swept area depends loosely on the number of path points and the selected view. If the number of CL-data points approaches $SCRSIZ^2$ (without rough cuts) and the user chooses a view such that the entire workpiece is visible then the number of pixels per swept area is on the order of one. The typical N/C geometric verification application falls somewhere between this case and the worst case scenario described above; the time complexity of a general case is certainly much less than $SCRSIZ^2 \cdot N$.

6.2 PERFORMANCE COMPARISON

In this section the performance of the direct N/C geometric verification algorithm described in this dissertation is compared to several other techniques.

Consider first the application of CSG solid modeling. Hunt and Voelcker [8] report that N/C geometric verification via direct application of CSG solid modeling has time complexity of $O(N^4)$ in the

general case and $O(N^3 \log(N))$ under certain special circumstances. Compared to the O(N) time complexity of the algorithm described above, this represents a dramatic difference in computational effort. This difference is probably due to the added effort in the solid modeling approach involved with N/C simulation (which must be done in conjunction with N/C geometric verification). Note that the technique described in this dissertation makes no attempt to verify such milling parameters as tool feed and speed rates. Obviously, the price paid for information concerning the amount of material removed with each motion of the tool is quite high. The computational efficiency of direct N/C geometric verification comes at the expense of neglecting the feed and speed rate factors. Much of the effort expended in a solid modeling N/C verification calculation has nothing to do with the final shape of the part.

Using the time complexities of both techniques, an interesting comparison is to calculate the value of N at which the lower order (simpler) algorithm becomes more efficient. Although Hunt and Voelcker did not address the constant of proportionality for their solid modeling approach, for the purposes of comparison, suppose the constant is equal to one. Applying the worst case time complexity for the direct N/C geometric verification technique described here with a realistic screen size of 512 by 512, the value of N at which this algorithm becomes more efficient than the solid modeling approach is 64. Although this is a very loose comparison, the order of magnitude of the result implies that for any but the simplest of milling operations, the direct N/C geometric verification technique would be the more efficient method. Application of the standard solid modeling approach to realistic milling operations

involving thousands of tool motions is probably not economically practical.

The other published techniques for N/C geometric verification are also (probably) more efficient than the direct solid modeling approach since they were designed with the application in mind. The image space solid modeling approach developed by Wang [24] [25] as well as Jerard's surface based Z-buffer technique [27] are probably of O(N), although neither author has published an algorithm analysis. However, Jerard's technique calculates cut values measured vertically from the surface regardless of the normal, thus introducing inaccuracies and limiting its possible future application to five-axis milling. Wang's technique seems to be one of the most promising. He claims to have five-axis verification capability already, but has yet to publish true sculptured surface capability.

CHAPTER VII

SUMMARY AND CONCLUSIONS

7.0 REVIEW OF THE DISSERTATION

This dissertation introduced a new way to consider the problem of automated N/C program verification. The general verification problem was dissected into two component problems: N/C geometric verification and N/C simulation. This work focused on the N/C geometric verification and a complete definition of this problem was developed and discussed. Previous analytical techniques based on solid modeling technology were discussed. The limitations of these approaches were examined with respect to the dissection of the problem introduced here. The computational inefficiency of the general solid modeling approach to N/C geometric verification was attributed to the fact that the N/C simulation problem must be considered simultaneously.

The approach presented in this dissertation was to attack N/C geometric verification directly, independent of the N/C simulation problem. An algorithm for this purpose was developed from a synthesis of sculptured surface shading techniques and elements of B-rep solid modeling. The algorithm was designed with the goal of minimizing unnecessary computations; it also incorporates options which allow the

user to trade accuracy for computational speed. The design and operation of the algorithm was discussed at length and several application examples were presented and discussed.

A performance analysis of this algorithm for direct N/C geometric verification showed that it behaves with a time complexity of O(N), where N is the number of tool path points. This compares to a time complexity $O(N^4)$ for the general solid modeling approach to the problem.

Thus, the algorithm developed in this dissertation is a viable solution to the problem of accurate and efficient geometric verification of N/C milling programs. It offers distinct advantages over the existing analytical techniques and could provide a significant productivity improvement over traditional manual verification methods.

7.1 FUTURE RESEARCH

One portion of this algorithm which shows promise for further development is in the application of the discretization element size. The smoothing technique described in Chapter Four demonstrates the feasibility of cut value interpolation as a means to increase computational efficiency, but it is flawed in at least two ways. First, for any given case, a limiting cell (discretization element) size exists, beyond which the size of the effective data set (and hence computation time) increases instead of decreases. This is a result of considering all the pixels in cells which fall partially on the background. The second problem with this interpolation technique is that some cells may fall partially on two different surfaces or across

surface discontinuities on the same surface, resulting in undesirable interpolation. Both of these could be remedied with more sophisticated sampling and interpolation schemes.

Alternatively, a completely different discretization scheme could be developed based, for instance, on Jerard's [27] technique of using local surface curvature and tool geometry to discretize the part only once, in world space. Verification could proceed much the same way as described here, except it would be done in world space. Then, a sophisticated mapping and interpolation scheme would be necessary to display the results on a raster device.

The direct N/C geometric verification technique presented here also shows some promise in extension to five-axis milling applications. For example, Wang's [24] [25] method of modeling five axis swept volume solids could theoretically be applied to intersect surface normals (as in this work) instead of sightlines. However, the entire portion of the this algorithm dealing with mill axis space would probably not be useful in such an extension since the mill axis changes at almost every motion of the mill.

Finally, the techniques developed in this dissertation may be useful in robotics applications. Problems such as collision avoidance and interference detection are some possible application areas. Also, with some modification, these N/C verification techniques could be applied to verify the film thickness performance of spray painting or coating robots. The geometric techniques presented in this dissertation may be useful in many other applications in the general areas of robotics and manufacturing.

APPENDICES

APPENDIX A

LOOK AHEAD AND LOOK BACK DISTANCES

In this appendix, the details behind the calculation of the "look ahead distance", L_a, and the "look back distance", L_b, are discussed. These parameters are used in the tool path processing phase of the N/C geometric verification algorithm as described in Chapter Four. L_a and L_b are used to construct the smallest mill tool swept area which is of sufficient size to include all portions of the surface that could be affected by a given tool motion. The look ahead distance is based on the angle between the current tool path vector and the next one, while the look back distance depends on the angle between the current path vector and the mill axis and the angle between the current and previous path vectors.

The following terminology is necessary for this discussion:

define,

- t a tool path point,
- τ a tool path vector,
- λ a unit tool path vector,
- θ angle between current and next path vectors,
- β angle between previous and current path vectors,

ζ the unit mill axis (Z) vector,

α angle between current path vector and mill axis,

R_{sa} swept area range (see Chapter Four).

The procedure for calculation of the look ahead and look back distances is most easily presented in the form of a "pseudo-code" algorithm, as shown in Figure A.1. Each step of the algorithm will be discussed. Control statments in this "pseudo-code" are shown in bold face type. The reader is encouraged to refer to Figure A.2, which depicts L_a, and Figures A.3 and A.4, which depict L_b, as necessary, to follow the following discussion.

The procedure outlined above begins with calculation of the current and next path vectors and the corresponding unit path vectors. The unit path vectors are then used to calculate the angle between the current and the next path vector, θ, as well as the angle between the current path vector and the mill axis, α. Note that the function $\cos^{-1}$ returns a value between zero and pi (π) radians, but that θ is restricted to be at most $\pi/2$.

The look ahead distance is then calculated as a simple function of θ and the swept area range R_{sa}, as shown schematically in Figure A.2. This relationship guarantees that L_a will be a maximum if the angle between consecutive path vectors is greater than or equal to $\pi/2$. On the other hand, if θ is zero, then the look ahead distance is set to zero since the bins which are "ahead" of the current terminal path point will automatically be considered in the next swept area (refer to Chapter Four, Figure 5). Note that when the final tool path point is considered, then L_a is set to the maximum value.

For each tool path point **i = 1** to **number of path points - 1**

---- Calculate path vector

If (first point) **then**

$\tau_i = t_{i+1} - t_i, \quad \lambda_i = \frac{\tau_i}{||\tau_i||}, \quad \beta = \pi/2$

Else

$\tau_i = \tau_{i+1}, \quad \lambda_i = \lambda_{i+1}, \quad \beta = \theta$

Endif

---- Next path vector and angle between consecutive paths

$\tau_{i+1} = t_{i+2} - t_{i+1}, \quad \lambda_{i+1} = \frac{\tau_{i+1}}{||\tau_{i+1}||}$

$\theta = \text{Max}(\cos^{-1}(\lambda_i \cdot \lambda_{i+1}), \pi/2), \quad \alpha = \cos^{-1}(\lambda_i \cdot \zeta)$

---- Look ahead distance

If (last point) **then**

$L_a = R_{sa}$

Else

$L_a = R_{sa}\sin(\theta)$

Endif

---- Look back distance

If (first point) **then**

$L_b = R_{sa}$

Elseif $(\pi/2 < \alpha < \pi)$ **then**

If $(0 < \beta < \pi/2)$ **then**

$L_b = R_{sa}\tan(\beta/2)$

Else

$L_b = R_{sa}\sin(\beta)$

Endif

Else

$L_b = 0$

Endif

---- Remainder of N/C geometric verification algorithm (Chapter Four)

-Calculate swept area
--Process bins
---Process pixels
----Vector solid intersection
---End pixels in bin
--End bins in swept area
-End swept area

Endfor

Figure A.1 Pseudo-code algorithm for look ahead and look back distance

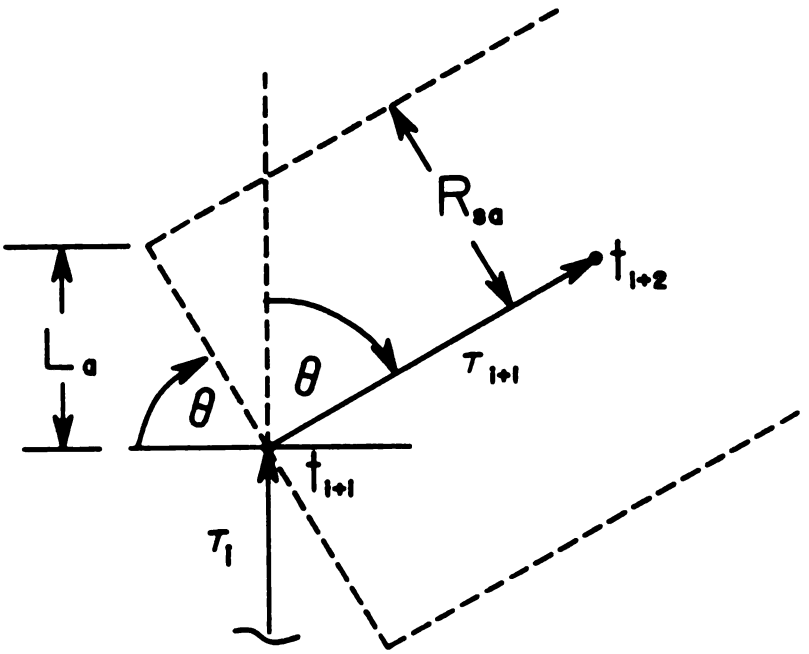


Figure A.2 Calculation of the look ahead distance, L_a

The look back distance involves a slightly more complex calculation. If the first path vector is under consideration, then L_b is set equal to R_{sa}. For a general tool motion, the look back distance depends first upon the angle that the current path vector makes with a plane which is normal to the mill axis. If the tool "lifts", i.e. α is less than $\pi/2$, then L_b is set to zero because the area behind the initial path point has been considered by the previous tool motion and cannot be cut deeper. If the tool "dives" then α will be greater than $\pi/2$ and the area behind the initial path point may have been cut deeper by the current tool motion than it was by the previous one.

Given that α is greater than $\pi/2$, the look back distance depends on the angle between the current path vector and the previous one, β. Note that for a general tool motion, β is set equal to θ before θ is updated for the next tool motion. Thus the angle is not recalculated but saved.

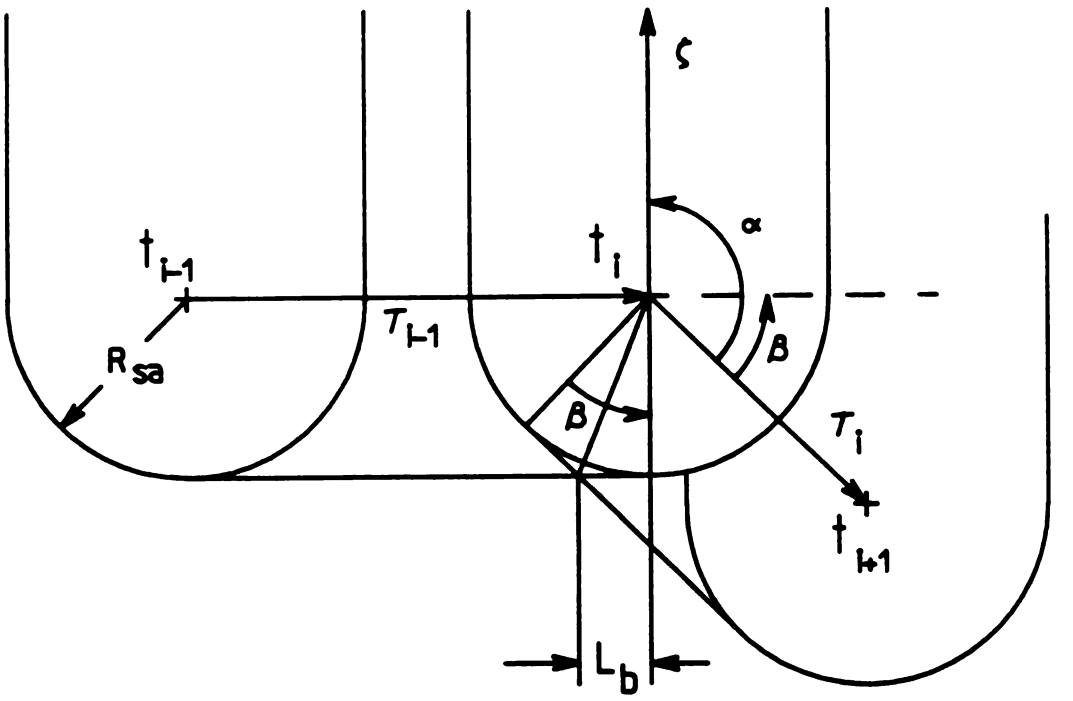


Figure A.3 Calculation of the look back distance, L_b, for $0 < \beta < \pi/2$

For $\pi/2 < \alpha < \pi$, Figure A.3 depicts calculation of the look back distance when $0 < \beta < \pi/2$, and Figure A.4 shows the case where $\pi/2 < \beta < \pi$. In either case, as β approaches $\pi/2$, L_b approaches the maximum value of R_{sa}. Thus when the mill "plunges", the algorithm will always reconsider bins "behind" the initial path point since they may be affected by this type of motion. On the other hand, as β approaches π, i.e., the tool retracts along nearly the same path, then L_b approaches zero since the area behind the current path point was already considered in the "look ahead" area of the previous tool motion.

The final portion of the pseudo-code procedure, "Remainder of the N/C geometric verification algorithm", was included for completeness. The details of the entire algorithm are covered in Chapter Four and Appendices B through D.

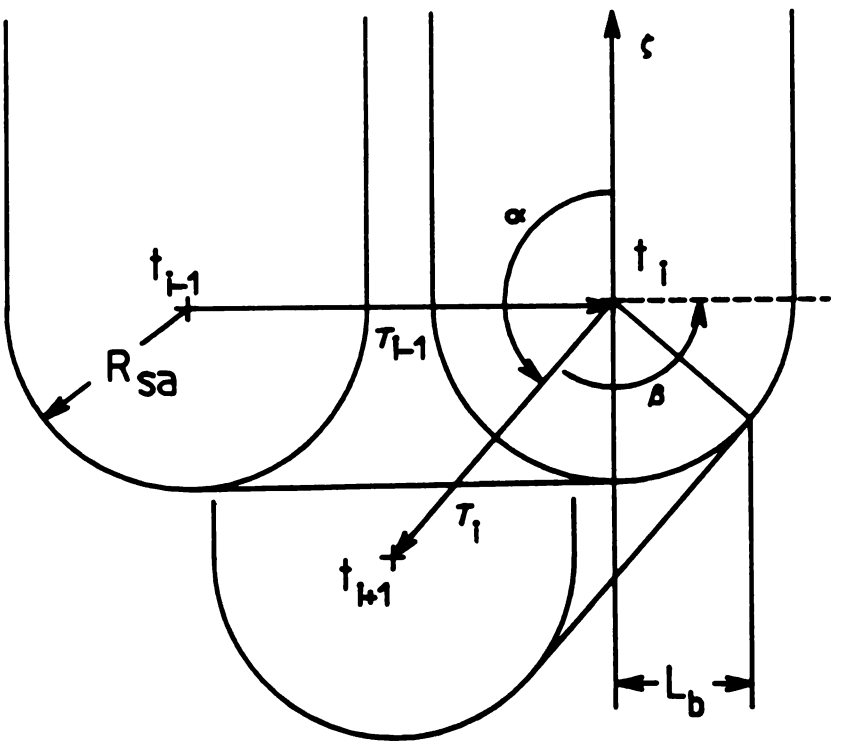


Figure A.4 Calculation of the look back distance, L_b, for $\pi/2 < \beta < \pi$

APPENDIX B

VECTOR/PLANE INTERSECTION

Consider an infinite plane in the form;

$$Ax + By + Cz + D = 0$$

where:

A, B, and C are the scalar components of a unit vector, η_p, which is normal to the plane.

The cartesian distance, L_{int}, from a point, p, to the plane, along the vector ν, is given by the following formula:

$$L_{int} = ((\eta_p \cdot p) + D) / (\eta_p \cdot \nu)$$

Note that the absolute value of the numerator in the above formula is simply the normal distance from the point, p, to the plane. The sign of L_{int}, given by the denominator, is the direction from p, along ν towards the intersection.

To save computations, in this application, one might wish to reject a point as "missed completely" if the normal distance (the numerator

above) is less than the sum of the range of interest, R_{int} and the outer tolerance, T_{out}. However, since the orientation of the point relative to the outward pointing normal (ν in this description) is not considered in the numerator, such an assumption would be in error. If such an approximation were applied, the algorithm could neglect cases of severe gouge.

APPENDIX C

EXAMPLE OF THE ROBERTS ALGORITHM

In this work, the Roberts algorithm is used to determine if a point on a plane lies within a closed bounded region on the plane called a face. The boundaries which define a face are oriented planes perpendicular to the plane of the face. Bounding planes must be constructed such that their associated normal vectors point towards the interior of the face. The following example will serve to illustrate the use of the Roberts algorithm.

Figure C.1 shows a face bounded by four planes labeled A through D. For simplicity, this example considers a face plane which lies parallel to the "Z = constant" plane. Hence it is a two-dimensional example in the X-Y plane, with bounding planes parallel to the Z axis. The algorithm works equally well for general three-dimensional planes.

The equations of each bounding plane are given below;

plane A: $4x - y = 0$

plane B: $-x + 6y = 0$

plane C: $-4x + y + 23 = 0$

plane D: $x - 6y + 23 = 0$

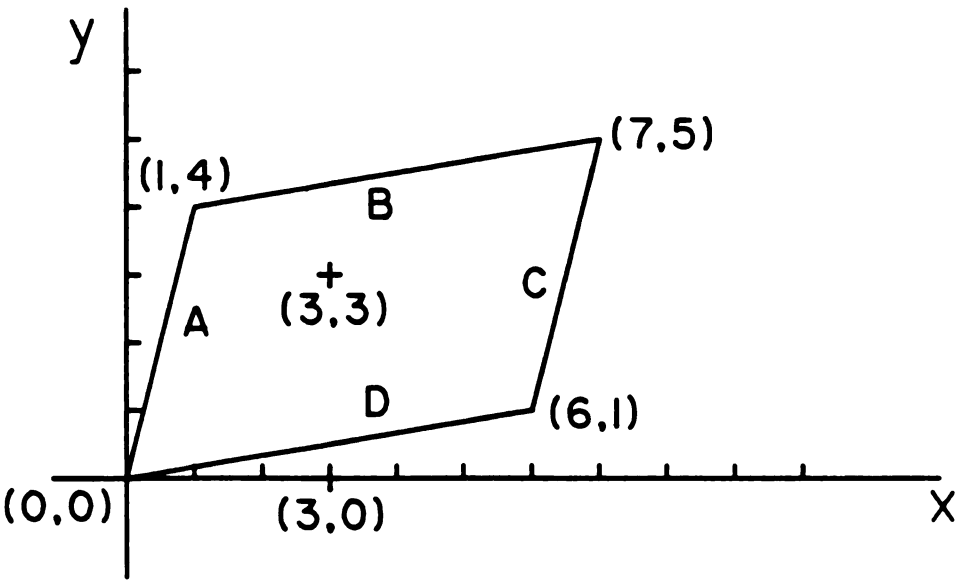


Figure C.1 Example application of the Roberts algorithm

The Roberts algorithm proceeds by taking the vector dot product of the homogeneous coordinates of the point in question with each of the four vectors composed of the coefficients of the bounding planes.

Consider the point on the plane given by (x,y) = (3,3). The homogeneous coordinates of this point are [3 3 1]. Taking dot products yields;

$$\begin{bmatrix} 3 & 3 & 1 \end{bmatrix} \begin{bmatrix} 4 & -1 & -4 & 1 \\ -1 & 6 & 1 & -6 \\ 0 & 0 & 23 & 23 \end{bmatrix} = \begin{matrix} A & B & C & D \\ [\ 9 & 15 & 14 & 17\ ] \end{matrix}$$

Since each bounding plane dot product yields a positive value, the point (3,3) is within the face defined by the above boundaries.

Alternatively, consider the point (x,y) = (3,0). This point yields a value of -3 for its dot product with the coefficients of plane B. This point therefore is out of bounds and is not on the face.

APPENDIX D

INTERSECTION OF A VECTOR WITH SPHERICAL AND CYLINDRICAL SURFACES

Part 1: Vector/Sphere Intersection

Figure D.1 shows the intersection of a surface normal vector with a spherical mill tool surface model. The following terminology defines the geometry of this situation;

given: p, a surface point,
ν, the corresponding (unit) normal vector,
s, the point at the center of a sphere of radius, R_s,

define: $\omega = s - p$, as a "connecting vector".

The perpendicular (shortest) distance, L_s, from the center of the sphere, s, to the normal vector, ν, is given by the magnitude of the vector resulting from the cross product of ω and ν, thus:

$$L_s = || \omega \times \nu ||$$

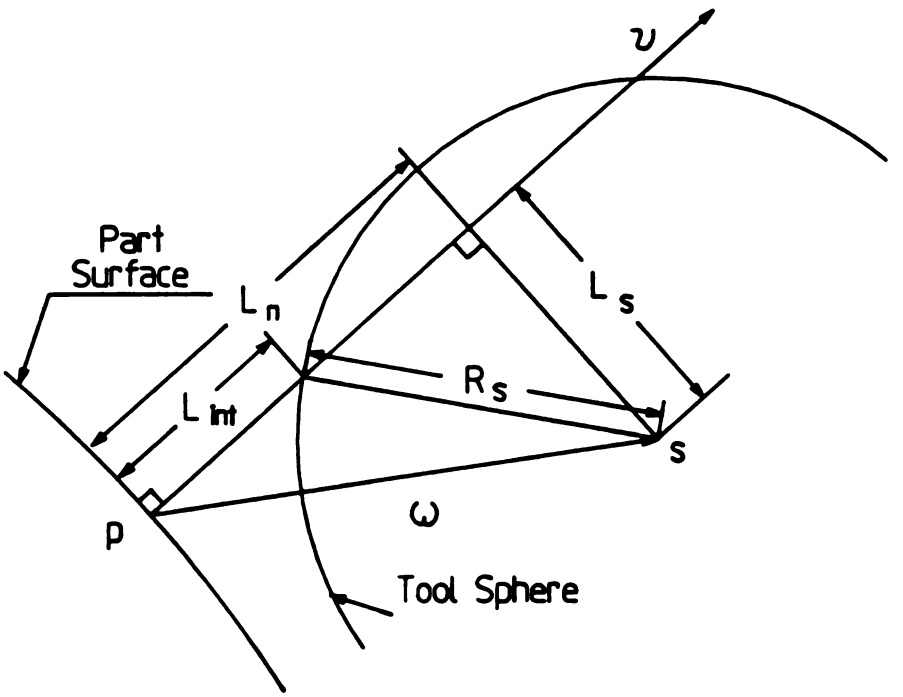


Figure D.1 Intersection of a vector with a sphere

If L_s is greater than the sphere radius, R_s, then the vector has missed the sphere model and there is no need to process the intersection calculation further. Otherwise, the distance, L_n, from p to the point on ν closest to the sphere center is calculated as follows,

$$L_n = (\omega \cdot \nu)$$

Now the precise directed distance, L_{int}, from p along ν to intersection with the spherical surface model can be obtained from,

$$L_{int} = L_n - \left[R_s^2 - L_s^2 \right]^{1/2}$$

Part 2: Vector/Cylinder Intersection

Intersection of a surface normal with a cylindrical surface model involves a somewhat similar calculation. The terminology defined above for a surface point and normal vector, along with definitions of the following terms, are sufficient to discuss this geometry.

given: c, the initial point (base) of the cylinder,
ϕ, the principal (unit) vector of the cylinder,
(the vector which the cylinder surrounds),
R_c, the radius of the cylinder,
define: $\psi = p - c$, a "connecting vector"

The first step of the intersection calculation is to determine if the normal vector, ν, and the principal vector, ϕ, are coplanar. This is done by calculating the following scalar triple product,

$$\sigma = (\phi \times \nu)$$
$$K_c = (\psi \cdot \sigma)$$

If the result of this product, K_c, is equal to zero (within the error bounds for the computer arithmetic), the normal vector, ν, and the principal vector, ϕ, are coplanar. This case is shown in Figure D.2. If K_c is not equal to zero, the vectors are noncoplanar as shown in Figure D.3.

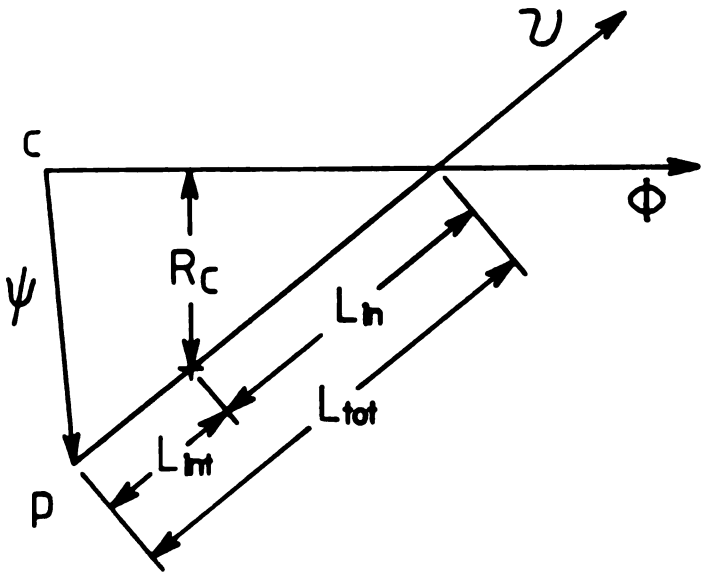


Figure D.2 Intersection of a vector with a cylinder, coplanar case

In the noncoplanar case, the perpendicular distance between ν and ϕ (i.e., the length of vector σ) can be determined as follows,

$$L_c = \frac{|K_c|}{||\sigma||}$$

If the shortest distance between the normal and principal vectors, L_c, is greater than the cylinder radius, R_c, the normal vector has missed the cylinder model and further processing is unnecessary. If L_c is less than R_c (or if the normal and principal vectors are coplanar), the intersection calculation proceeds.

The same vector loop equation is used for both the coplanar and noncoplanar case of normal vector intersection with a cylinder, with slightly different inputs. For the noncoplanar case, the distance, L_{in}, along the normal vector from the cylinder wall to the intersection with vector σ, is given by,

$$L_{in} = \left[\frac{R_c^2 - L_c^2}{(1 - D_{np}^2)} \right]^{1/2}$$

where:

$D_{np} = (\nu \cdot \phi)$, the dot product of normal and principal vectors.

For the coplanar case, the distance along the normal vector from the cylinder wall to the normal's intersection with the principal vector is given by,

$$L_{in} = \frac{R_c}{\sin(\theta)}$$

where:

$\theta = \cos^{-1}(D_{np})$, the angle between the normal and principal vectors.

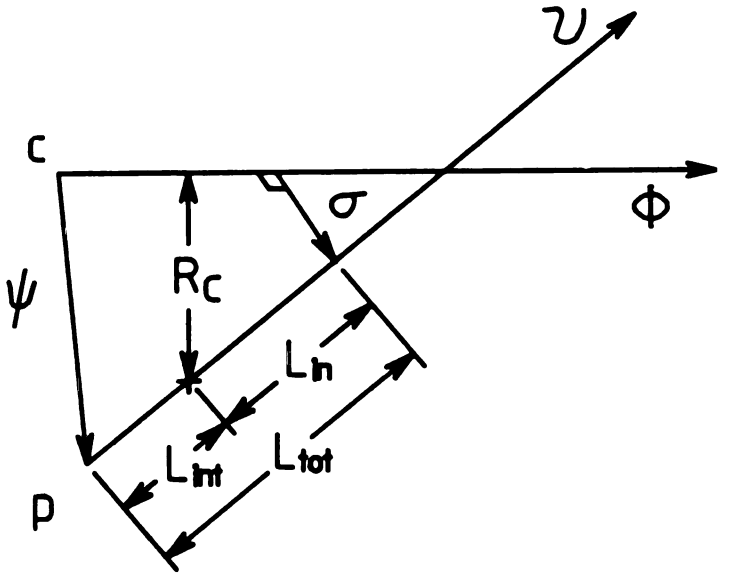


Figure D.3 Intersection of a vector with a cylinder, noncoplanar case

The other case-specific input required in the intersection formulation is the "known vector" in the loop equation. These inputs are given below,

$$\kappa = \psi + L_c\sigma \quad \text{for noncoplanar case}$$

$$\kappa = \psi \quad \text{for coplanar case}$$

The normal vector intersection with the cylinder model is then given by:

$$L_{tot} = \frac{\kappa \cdot (\nu - D_{np}\phi)}{(1 - D_{np}^2)}$$

$$L_{int} = L_{tot} - L_{in}$$

where:

L_{tot} is the directed distance along the normal vector from surface point to either, ϕ if coplanar, or σ if noncoplanar.

L_{int} is the directed distance along the normal vector to intersection with the cylinder wall.

LIST OF REFERENCES

LIST OF REFERENCES

[1] CAM-I Inc. "APT4 Sculptured Surfaces Part Programmers Manual," August, 1984.

[2] Smith, D.N. and Evans, E., "Management Standards for Computer and Numerical Control," Institute of Science and Technology, Industrial Development Division, Ann Arbor, Michigan, 1977.

[3] ANSI, "Dimensioning and Tolerancing, ANSI Y14.5M-1982," The American Society of Mechanical Engineers, New York, N.Y., 1982.

[4] Voelcker, H.B. and Hunt, W.A., "The role of Solid Modelling in Machine-Process Modelling and NC Verification," Procedings of SAE 1981 International Congress and Exposition, Detroit, Michigan, February, 1981.

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

[6] Ruberl, S.T., "Verification of NC Part Programs with Interactive Computer Graphics and Solid Geometric Modeling," Proceedings of CAM-I 10th Annual Meeting and CAD/CAM Graphics User's Exposition, Fort Worth, Texas, October, 1981.

[7] Fridshal, R., Cheng, K.P., et al, "Numerical Control Part Program Verification System," Proceedings of the Conference on CAD/CAM Technology in Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, March, 1982.

[8] Requicha, A.A.G. and Voelcker, H.B., "Solid Modeling; A Historical Summary and Contemporary Assessment," IEEE Computer Graphics and Applications, Vol.2, No.2, March 1982.

[9] Requicha, A.A.G. and Voelcker, H.B., "Solid Modeling: Current Status and Research Directions," IEEE Computer Graphics and Applications, Vol.3, No.10, October, 1983.

[10] Eastman, C.M. and Weiler, K., "Geometric Modeling Using the Euler Operators," Proceedings of First Annual Conference on Computer Graphics CAD/CAM Systems, Massuchusetts Institute of Technology, Cambridge, MA, April, 1979.

[11] Roth, S.C., "Ray Casting for Modeling Solids," Computer Graphics and Image Processing, 18, February, 1982.

[12] Meagher, D., "Geometric Modeling Using Octree Encoding," Computer Graphics and Image Processing, Vol. 19, 1982.

[13] Jackins C.L., and Tanimoto, S.L., "Oct-Trees and Their Use in Representing Three-Dimensional Objects," Computer Graphics and Image Processing, Vol. 14, 1980.

[14] Bobrow, J.E., "NC Machine Tool Path Generation From CSG Part Representations," Computer Aided Design, Vol. 17, No. 2, March, 1985.

[15] Y.T. Lee and A.A.G. Requicha, "Algorithms for Computing the Volume and Other Integral Properties of Solids: I - Known Methods and Open Issues; II - A Family of Algorithms Based on Representation Conversion and Cellular Approximation," Communications of the ACM, Vol. 25, No. 9, September, 1982.

[16] Tamminen, M. and Samet, H., "Efficient Octree Conversion by Connectivity Labeling," Computer Graphics, Proceedings of SIGGRAPH, Vol. 18, No. 3, July, 1984.

[17] Chiyokura, H. and Kimura, F., "Design of Solids with Free-Form Surfaces," Computer Graphics, Proceedings of SIGGRAPH, Vol. 17, No. 3, July, 1983.

[18] Kimura, F., "Geomap-III: Designing Solids with Free-Form Surfaces," IEEE Computer Graphics and Applications, Vol. 4, No. 6, June, 1984.

[19] Casale, M.S. and Stanton, E.L., "An Overview of Analytic Solid Modeling," IEEE Computer Graphics and Applications, Vol. 5, No. 2, February, 1985.

[20] Cohen, E., Lyche, T. and Riesenfeld, R.F., "Discrete B-splines and Subdivision Techniques in Computer Aided Geometric Design and Computer Graphics," Computer Graphics and Image Processing, Vol. 14, 1980.

[21] Hanna, S.L., Abel, J.F. and Greenberg, D.P., "Intersection of Parametric Surfaces by Means of Look-Up Tables," IEEE Computer Graphics and Applications, Vol. 5, No. 2, February, 1983.

[22] Butterfield, K.R., "The Development and Application of Algorithms Associated with Surface Representation," Ph.D. Dissertation, Brunel University, Uxbridge, U.K., 1978.

[23] Atherton, P.R., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry," Computer Graphics, Proceedings of SIGGRAPH, Vol. 17, No. 3, July, 1983.

[24] Wang, W.P., "Solid Geometric Modeling for Mold Design and Manufacture", Ph.D. Dissertation, Cornell University, 1984.

[25] Wang, W.P., "Integration of Solid Geometric Modeling for Computerized Process Planning," Computer-Aided/Intelligent Planning - PED, Vol. 19, Book No. G00334, American Society of Mechanical Engineers, 1985.

[26] Farouki, R.T. and Hinds, J.K., "A Hierarchy of Geometric Forms", IEEE Computer Graphics and Applications, Vol. 5, No. 5, May, 1985.

[27] Jerard, R.B., Hauck, K., and Drysdale, R.L., "Simulation of Numerical Control Machining of Sculptured Surfaces," Proceedings of International Symposium on Automotive Technology and Automation, Flims, Switzerland, October, 1986.

[28] Catmull, E., "Computer Display of Curved Surfaces," Proceedings of IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure, May, 1975.

[29] Riesenfeld, R.F., "Applications of B-Spline Approximation to Geometric Problems of Computer Aided Design," Ph.D. Dissertation, Syracuse University, 1973.

[30] Versprille, K.J., "Computer Aided Design Applications of the Rational B-Spline Approximation Form," Ph.D. Dissertation, Syracuse University, 1975.

[31] Pickelmann, M.N., The Design of Rational B-Spline Algorithms for Interactive Color Shading of Surfaces," Ph.D. Dissertation, Michigan State University, 1985.

[32] Cox, M.G., "The Numerical Evaluation of B-Splines," Journal of the Institute of Mathematics Applications, Vol. 10, pp. 134-147, 1972.

[33] DeBoor, C., "On Calculation with B-Splines," Journal of Approximation Theory, Vol. 6, pp. 50-62, 1972.

[34] Coviak, R.A., "Color Graphics in Engineering Design," Masters Thesis, Michigan State University, 1981.

[35] Watkins, G.S., "A Real-Time Visible Surface Algorithm," Technical Report, UTEC-CSC-70-101, Computer Science Department, University of Utah, 1970.

[36] Gouraud, H., "Continuous Shading of Curved Surfaces," Ph.D. Dissertation, University if Utah, 1971.

[37] Phong, B., "Illumination for Computer Generated Images," Ph.D. Dissertation, University of Utah, 1973.

[38] Catmull, E., "A Subdivision Algorithm for Computer Display of Curved Surfaces," Ph.D. Dissertation, University of Utah, 1974.

[39] Lane, J.M. and Carpenter, L.C., "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," Computer Graphics and Image Processing, Vol. 11, 1979.

[40] Blinn, J.F., "Computer Display of Curved Surfaces," Ph.D. Dissertation, University of Utah, 1978.

[41] Whitted, T., "A Scan-line Algorithm for Computer Display of Curved Surfaces, Computer Graphics, Proceedings of SIGGRAPH, Vol. 12, 1978.

[42] Lane, J.M., Carpenter, L.C., Whitted, T., and Blinn, J.F. "Scan Line Methods for Displaying Parametrically Defined Surfaces," Communications of the ACM, Vol. 23, No. 1, January, 1980.

[43] Vanderploeg, M.J., "Surface Assessment Using Color Graphics," Ph.D. Dissertation, Michigan State University, 1982.

[44] Oliver, J.H. and Goodman, E.D., "Color Graphic Verification of N/C Milling Programs for Sculptured Surfaces," 10th Annual ESD/ACM Automotive Computer Graphics Conference and Exposition, Engineering Society of Detroit, Detroit, Michigan, December, 1985.

[45] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., "Data Structures and Algorithms," Addison-Wesley Publishing Co., Reading, MA, 1983.