This is to certify that the

dissertation entitled

ON LOOSELY COUPLED PARALLEL IMPLEMENTATION OF ALGORITHMS FOR COMPUTER AIDED MANUFACTURING

presented by

Ming Bao

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Electrical Engineering

Major professor

Date Aug. 19, 1997

ON LOOSELY COUPLED PARALLEL IMPLEMENTATION OF ALGORITHMS FOR COMPUTER AIDED MANUFACTURING

By

Ming Bao

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1997

ABSTRACT

ON LOOSELY COUPLED PARALLEL IMPLEMENTATION OF ALGORITHMS FOR COMPUTER-AIDED MANUFACTURING

By

Ming Bao

Many manufacturing processes, such as numerically controlled (NC) machining and robotic spraying, are conducted under direct numerical control of computers. The process of creating such numerical control (NC) programs remains quite complex and error-prone for many sophisticated tasks. The final result of tool motions under control of such programs in machining complex-shaped parts is often uncertain. Proofing runs on actual NC machines are time consuming and expensive. The high cost of conducting test runs of such programs has created a strong demand for simulators and verifiers for such programs. Current NC simulation software on sequential machines greatly improves the verification procedure. However, such NC simulators are still inaccurate and most can handle only 3-axis milling with the simplest types of cutters. Given the former dearth of sophisticated software for generating optimal 5-axis toolpaths for more complex tools, this inadequacy of simulators or verifiers was not very noticeable. However, the current trend toward higher-capability NC programming systems raises the strong need for more

sophisticated and accurate verification. This trend highlights the tradeoff between verification time and accuracy of verification, strongly raising the desire for parallelization of NC verification or simulation processes.

This research presents an algorithm which can address both concerns -- high efficiency and accuracy, using parallel (or distributed) processing. It not only guarantees that tessellated surfaces are within a user-specified tolerance, but also pre-estimates work load directly from the original geometry parameters, such as surface order, control points and knots. The proposed algorithm pre-evaluates the sculptured surfaces to be machined for parallel processing, estimates the load balance for the processors, discretizes the nominal sculptured surfaces based on the surface curvature and user-defined error tolerance, distributes the computational job onto the given number of processors or workstations, and uses a parallel processing approach to directly compute the possible interference between the surfaces being machined and the envelope of the moving tool, without solving the swept volume problem as a solid modeling operation. The geometric model uses the ruled surface defined by the axis of the cutting tool to define the center of the tool envelope. The surface pre-evaluation guarantees that near-optimal computational performance will be achieved on a given number of processors. We believe the proposed algorithm could be easily implemented on a distributed system.

Some results from a sequential 5-axis NC toolpath verification system implemented by the authors are presented first, and this is used as the basis for the parallelization work described here. Some published parallel approaches for NC machining are reviewed, then the new scheme for parallelization is presented. Finally, the performance of this scheme on parallel vs. single CPU machines is reported and future work is discussed.

To my parents, Erlan Ming and Shiqiang Bao

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to Prof. Erik D. Goodman, to whom I owe a lot for his consistent guidance, support and willingness to share his time and knowledge through this research. I cannot possibly enumerate all the ways in which he as a teacher and as a friend has made my graduate education a resourceful and exciting experience. His confidence in providing me the freedom to explore and his tactful participation in keeping my work in the proper perspective are deeply appreciated. Furthermore, I would like to thank him for presenting to me the challenges in the manufacturing area with parallel computation.

I would like to thank Prof. Lionel M. Ni who has brought me into the fascinating world of parallel processing. His broad knowledge and experience in parallel computation have been one of the most valuable sources of ideas for the research. I am grateful to Prof. Michael A. Shanblatt and Prof. David H.Y. Yen for taking time to serve on the guidance committee and overseeing this work. I especially thank Prof. Stephen V. Dragosh for stepping in on short notice and reading the thesis when Prof. Yen became ill.

Special thanks also are given to my wife Li Liu and to my parents, whose support during this study were greatly useful and inspirational.

Sincere thanks should be conveyed to my friends, Leslie Hoppensteadt, Ki-Yin Chang, etc., for providing many sources of my ideas.

Finally, I'd like to thank Erik Veenstra and the Advanced NC Technology team at Ford Motor Company for their support on this study.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

It is crucial to develop methods for planning and simulating motions of objects among obstacles in order to fulfill the constant demand for ever higher levels of manufacturing automation. Motion planning and motion simulation methods have found important roles in many manufacturing processes, such as generating collision-free workpaths for robot manipulators, generating gouge-free toolpaths for numerical control machines, etc.

1.1 Numerically Controlled Milling Machines

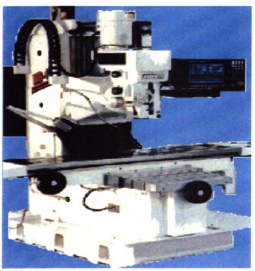
Numerically controlled (NC) milling is a common manufacturing process. NC programs are able to guide machining operations for removal of materials from parts (or tools) to be machined. Advanced NC systems such as computer numerical control (CNC) and direct numerical control (DNC) systems, combined with other computer technology, opened the door for computer-aided manufacturing (CAM) to do automatic control of manufacturing processes and systems. NC technology is not only used extensively in the metal-removing milling process, but also applied in variety of other manufacturing processes.

An NC machine is characterized by the motions it can perform. According to the changes of the relative positions and orientations of the tool and workpiece allowed, it can

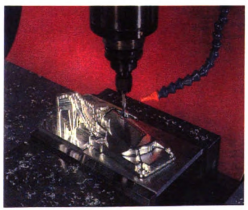
be classified as a two-axis, three-axis, or multi-axis (four or more) milling machine. Two-axis milling indicates that the contouring capability of the machine tool is limited to motions with respect to X and Y axes, i.e. in a fixed Z plane. This mode of operation is frequently referred to as two-dimensional operation. Three-axis milling refers to a cutting tool moving simultaneously in the X, Y and Z axes under complete control of the NC program. The tool axis orientation of the three-axis milling machine does not change relative to the workpiece during the entire tool motion. Figure 1.1 shows an example of machining on a typical three-axis NC machine. This type of NC machine is the most commonly used in industry today. When more complicated parts with complex shapes are desired, movement capability in only the X, Y and Z axes may require repeated refixturing of the part on the machine, and also excessive numbers of tool passes to meet the required tolerances. Rotation of the tool or part about various of their axes while the tool is moving may greatly improve the manufacturability of such parts. The multi-axis milling machine is designed for this purpose. In the aerospace and automotive tooling industries, there have long been demands for machines that have 360^{0} contouring with simultaneous control of the Z axis; this is usually accomplished with a five-axis milling machine which has two rotary axes. The five-axis milling machine can continuously re-orient the axis of the tool as it follows the contour of the part, or alternatively, re-orient the part. In general, the machine axes, or the location and the orientation of the cutter, change with almost every motion of the mill during a multi-axis milling process. Five-axis machining is in increasing demand in automotive and aerospace industries.

Despite the widespread use of multi-axis milling, the correctness of the process is still a subject of considerable interest. Many researchers focus on the generation of gouge-

free toolpaths. Others focus on the verification of the quality of the generated toolpath for correction of toolpaths. We intend to build a closed-loop NC system which generates, verifies and corrects toolpaths. Parallel NC verification is thus a component of our future system. The objective of parallel verification of NC toolpaths is to evaluate, with high performance and accuracy, the geometric quality of the real process, before running it on the actual machines. This work does not, currently, consider the problems of tool deflection, tool wear, chatter, tool breakage, or part deflection. Thus, it cannot be used to verify the correctness of "feeds and speeds".



(a) 3-Axis Machine



(b) 3-Axis Machining

FIGURE 1.1 Example of 3-axis Machine

1.2 NC Program Verification

Although the principle of the NC process is simple, the practice is significantly more complex, and a number of common pitfalls can lead to major process failures. Since an NC program for sculptured surface machining consists of a sequence of (typically) lin-

early interpolated tool positions, measures must by taken to avoid overcutting and undercutting, to the extent required by the tolerances. Undercutting can easily occur when there is a discontinuity among multiple surfaces or the stepover distance is too large for the local surface curvature. Overcutting, or gouging, is a particularly painful problem in sculptured surface machining. Gouging is often encountered when the cutter size is too large relative to the concave radius of curvature, when there is a lack of continuity among multiple surfaces or when there are (sometimes minute) gaps between CAD-defined surfaces. Figure 1.2 shows various cases of undercutting and gouging for a ball-end cutter. Figure 1.2 (a) shows the undercutting caused by a large stepover for either zig-zag or box cutting. In Figure 1.2 (b), undercutting of area B and gouging of area C are caused by use of a large cutter. D is caused by a surface discontinuity, and E results from surface gaps.

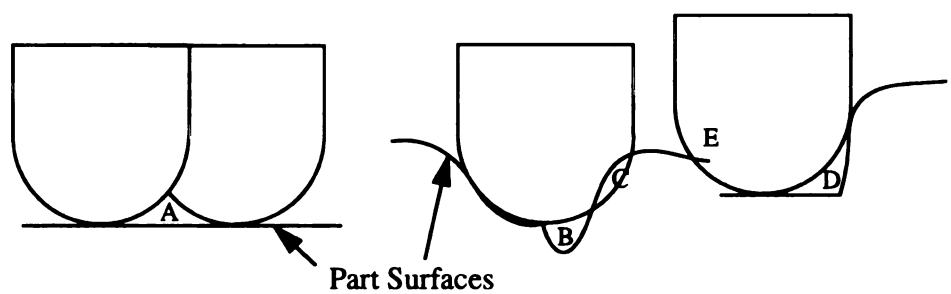


FIGURE 1.2 Undercutting and Gouging for Ball Cutter

When given true position tolerances, overcutting and undercutting of design part surfaces are defined as shown in Figure 1.3, where ABCD is the as-machined surface by a particular toolpath. Two tolerance limits, the outside- and inside-tolerance limits, are

defined. If a machined surface point is between the two tolerance limits, the point is considered within tolerance. If a machined surface point is outside the outside tolerance limit, it is considered an undercut surface point. Similarly, if a machined surface point is beyond the inside tolerance limit, it is considered a gouged surface point. In the example in Figure 1.3, the machined surface point A is within tolerance, B and C are undercut, and D is gouged.

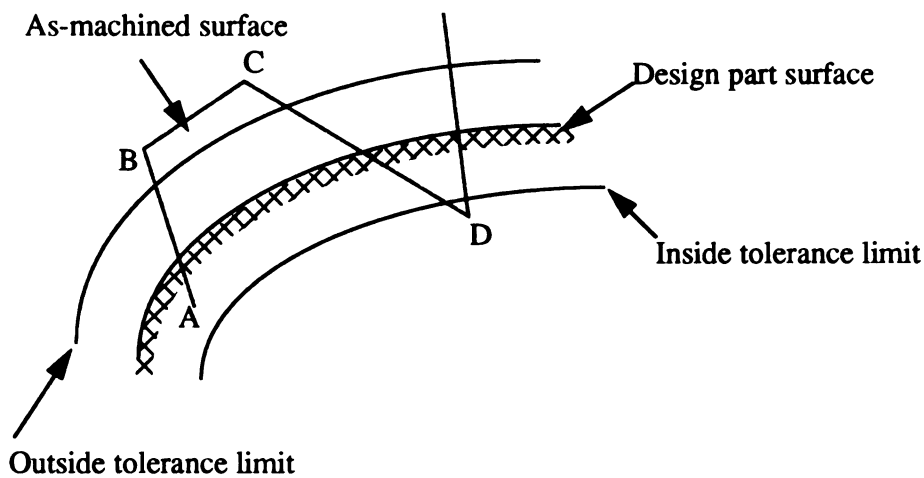


FIGURE 1.3 Tolerances for the NC Machined Surface

Traditional NC program proofing has normally been done by milling materials softer than the actual ones, or scaling down the size of the machined part to reduce the cost of the process. Verifying NC programs by the proofing runs on an actual NC machine is time consuming and costly. Most important of all, a successful dry run only indicates that the NC program is apparently correct. It is often impossible to measure all possible deviations between the whole design part model and the workpiece part, especially for sculptured surface models. Computer-aided NC verification constitutes a new tool for this problem. It enables calculation of the errors from the desired part, excluding any of the

unmodeled effects of tool and part deflection and tool wear. It can easily trace gouges to specific tool motions. It is a feasible and economical way to check NC programs, even if a reduced number of actual proofing runs is still required before final milling. Given the growing emphases on quality, flexibility, and agility in high technology manufacturing such as NC machining, there is strong demand for more accurate, more efficient, and more automatic computer-based manufacturing processes. Because development of the programs to guide manufacturing equipment is often time consuming, and because the quality of these programs often translates directly into product quality, manufacturing process simulation and verification can play an important role in the enterprise. It can contribute to quality, flexibility and agility through speedup of process development, optimization of cycle time, improvement of quality, minimization of waste, and consequently, reduction of manufacturing cost.

1.3 Motivation

Parallel or distributed computation are important to this effort because for many manufacturing process simulation and verification tasks, a single PC or workstation does not have sufficient computing power to carry out the tasks fast enough to permit automatic generation of manufacturing process programs, such as NC toolpaths. Furthermore, a uniprocessor system is often too slow to allow interactive design of manufacturing programs without significantly increasing the time required for new process development. To speed up simulation and verification processes without using expensive supercomputers, one can use parallel or distributed processing. However, application of parallel or distributed architectures to such simulation and verification tasks, such as NC machining, etc., is non-triv-

ial and still rare. Such tasks are not "embarrassingly parallelizable," and decomposability of a verification algorithm can be aided significantly if it is designed with this in mind.

CHAPTER II

LITERATURE REVIEW

Software for manufacturing process simulation and verification, including the NC machining process, typically computes either a graphical representation of the "as milled" workpiece or the differences between the design part model and workpiece part model (machined part model). Among the surface types machined by NC programs, sculptured surfaces which are made up of arbitrary, nonanalytic contours comprise one of the most challenging areas. Computer-assisted verification systems provide a process by which the errors between the desired part with the specified tolerances and the part as milled can be calculated and traced to specific tool motions. This process, using graphical computer models to replace at least some expensive physical prototypes, is a feasible and economical way to check the correctness of the CL (cutter location) data file. However, verifying an NC program, even for only three-axis machining, is still tedious work. In the last two decades, many researchers have worked on NC verification and simulation to reduce proofing time, through various approaches such as Constructive Solid Geometry (CSG), image-space methods, object-space methods, etc. However, the parallel processing is rarely considered, except for that provided by specialized graphics processors.

NC toolpath verification began with direct solid modeling approaches. It is based on the principles of set theory, and assumes that a simulated machined part can be represented as the result of a series of Boolean subtractions of the swept volumes of the tool

motions from the workpiece. The result of these operation is indeed a simulated machined part, but not the solution to the verification problem, which seeks to show the discrepancies between the simulated machined part and the nominal (design) part. Therefore, these approaches need to perform another Boolean subtraction to come up with the difference between the relatively simple design part model and the generally very complex simulated machined part in order to solve the verification problem. Of course, this difference is not necessarily in the form in which the user desires to see the discrepancies -- more likely, an identification of which toolpath segment caused a gouge, for example, is desired. Obviously, solid modeling based NC verification requires some other computations than those generally performed by solid modeling systems. Hence, it has been and still is very difficult to solve verification problems with high complexity because such problems tend to require very intensive computations. Therefore, other techniques designed more specifically for solving the same problem have been sought. Based on the new available hardware technology, there are two major categories of NC simulation and verification approaches which have been developed over the years. One aims at a new and more efficient model to represent the NC verification processes; the other aims at use of advanced multiprocessor technology. Image space approaches are examples of the former, while parallel or distributed NC verification approaches are one of the later. Image space methods employ the methods developed for surface shading to solve the verification problem. The swept volume of the tool motion is converted by a scan line rendering processor into screen pixels which are comparable with the screen pixels of the workpiece and fixtures along a sight line. The Boolean subtraction is performed in the image space and is view-specific. The resolution for detecting errors is limited by the resolution of the view, the Z-buffer as well as the view itself. The final verification produces the same type of result as a

solid model approach, but only for the chosen view. *VeriCut* is a typical example of this approach, and is a leader in the current market. The parallel or distributed NC verification approach focuses on exploiting advanced multiprocessor computer technology. It divides one NC verification process into sub-processes and distributes each of them onto more than one processor to increase the verification efficiency and reduce elapsed clock time. However, this method is still relatively new and more researches are needed to make it commercially successful.

Methods of NC milling simulation and verification are generally distinct from techniques used to model milling phenomena and formulate milling problems. One way to categorize these methods is to break them into four approaches: solid modeling, discrete vector intersection, spatial partitioning representation and parallel processing. Since the spatial partitioning representation approach is, in fact, a combination of others, and the discrete vector intersection approach can be further divided, we will view the methods in the following four categories: solid modeling method, image space method, object space method and parallel processing method. The first three approaches have been applied to five-axis NC verification with varying ranges of applicability and degrees of success. The parallel processing based approach is relatively new and is still under study.

2.1 Solid Modeling Techniques

Solid geometry modeling based NC simulation and verification began in the early 1980's. Solid geometry modeling systems[1]-[5] offer the possibility of doing both simulation and verification. Simulation is achieved by subtracting models of the swept volumes of tool motions from the model of the workpiece. A type of verification can be achieved

by Boolean subtraction of the model of the final workpiece from the desired part model. Constructive Solid Geometry (CSG) is one solid geometric modeling scheme. It provides a constructive representation of an object of complicated shape, based on a set of primitive solids such as blocks, cylinders, spheres, cones and other completely surface-bounded solids, and a set of Boolean operations among the primitive blocks. A complete solid can be obtained through simple Boolean operations, such as union, difference, intersection and assembly, of primitive solids. However, the CSG model is an implicit representation in the sense that the active regions bounding a complex solid are not represented explicitly in the data structure, which must be computed by means of the definitions of the primitives and the effects of the Boolean operations stored in the tree structure. A tree structure with Boolean operators at the non-terminal nodes and primitives at the terminal nodes can easily be used to define a CSG solid, where the root node represents the complete solid.

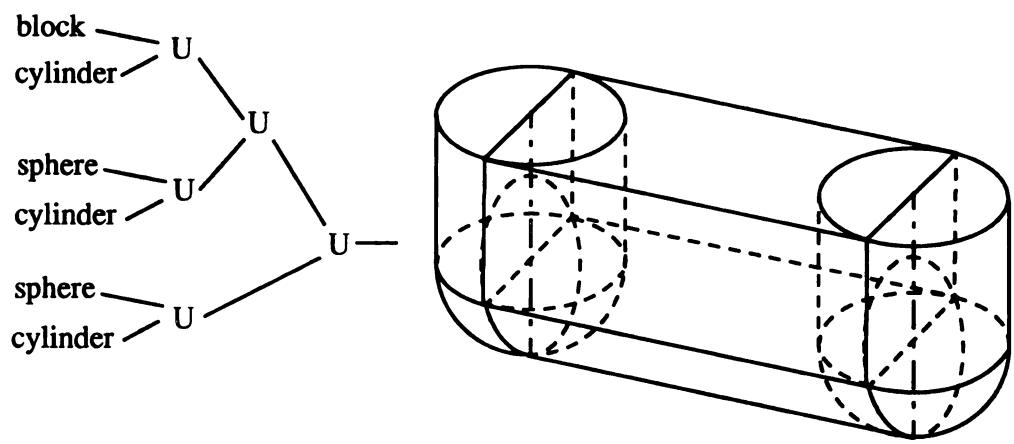


FIGURE 2.1 Swept Volume of Ball Cutter in CSG Model

As an example shown in Figure 2.1, a swept volume[5][6] of a toolpath with a ball cutter can be represented as a union of three cylinders, two spheres, one block. The simu-

lation of this kind of machining can be obtained by sequentially subtracting the swept volume models of tool motions from the workpiece model. The verification can be obtained by Boolean subtraction of the workpiece model from desired part model. The task is not as easy for multi-axis machining, where the same primitive solids are not enough to represent a twisting tool motion. One needs either to generate more complex primitive solids or to approximate them as unions of many more primitive blocks. An exploratory study was done by Voelcker and Hunt[1][2] on the feasibility of using the PADL CSG modeling system to simulate NC programs. Fridshal[3] modified GDTIPS to do NC simulation. A problem of the CSG approach is that it requires a large amount of computational expense, especially for multi-axis NC simulation and verification, where mathematical primitives are extremely complex and Boolean operations on them are very compute-intensive.

2.2 Image Space Techniques

The image-space method for NC simulation and verification is another category, where Boolean operations are done during image rendering. The 3-D machining process can be reduced to a 2-D problem by considering the intersection of rays from each image pixel. Two typical image space approaches are proposed by Wang[5][6][7] and Van Hook[8]. Almost all of the image based methods take advantage of the concept of Z-buffer. Z-buffer contains a real number Z or a depth value associated with each (X, Y) screen pixel. It always keeps the closest Z value and ignores all the other Z values for each pixel along the sight line. In other words, it always keeps all the visible Z values. Thus it is a widely used for hidden surface removal for display of an interactive shaded solid as well as some other area such as NC machining. For example, Carl Jepson at Ford Motor Company success-

fully developed a Z-buffer-based software package called CHIPS[42], which is able to generate multi-axis gouge-free toolpaths for sculptured surfaces.

Wang[5][6][7] developed a direct NC geometric verification technique for five-axis milling application. This is a novel view-dependent method for five-axis NC verification. The beauty of Wang's algorithm is that he successfully incorporated the idea of a Z-buffer, which is widely used in hidden surface removal processes, into NC verification. The geometric model of the boundary surface of the swept volume or the composition of the envelope surface is clearly described by its parametric form equation. The envelope consists of two categories of surface, the subset of the boundary of the generator at the initial position and the final position, and the new surface created by the generator during the motion. The algorithm uses a standard Z-buffer and converts CSG part data into pixel data stored in the Z-buffer for subsequent Boolean operations with other models. The swept volume is also converted into pixel data, which will be compared with those of the workpiece and fixtures. The Boolean subtraction removes the material from the workpiece. The interference between the tool swept volume and the workpiece can be shown using different colors to highlight various error areas.

Another Z-buffer-based approach was developed by Van Hook[8]. He developed an extended Z-buffer structure called a *dexel*. In contrast to a Z-buffer, a dexel contains not a single Z value, but several entries for each (X, Y) elements, such as a pointer, color, Z value of the furthest surface and the Z value of the nearest surface. The dexel structure is directly displayed just like a normal frame buffer since the color of a dexel at (X, Y) screen coordinate is the visible surface color at that pixel location. Hook's method differs from Wang's in that instead of intersecting scan lines with swept volumes, he precomputes

a pixel image of the cutting tool and performs Boolean subtractions of the cutter from the workpiece along the toolpath. After the tool is subtracted from the workpiece, the far surface of the tool typically becomes the new near surface of the block, and the inversely shaded tool color is the properly shaded new workpiece color. The calculation of all the distance of the verification is along the view-dependent sight lines. The selected view of the shaded image after verifying the milling path is easily displayed, but cannot be redisplayed from another view point without recomputing the entire problem. The view of the final part at the completion of the milling is an image-based model that does not provide toleranced verification or mass properties. His method is limited to three-axis toolpath verification.

Similar to Wang's method, Saito and Takahashi[9][10] developed the G-buffer, another extension of the idea of a Z-buffer, and applied it to NC toolpath generation and verification using graphics or image processing hardware. The image space methods developed by Wang, Van Hook, Saito et al. are all view dependent, which allows errors to be undetected because of the chosen viewing direction, and causes discrepancies to be calculated along the Z direction, rather than along part surface normals. Thus, they are not checking according to the actual specification of positional tolerances. Displaying another view of the part requires running the entire simulation again. In addition, the Z-buffer approaches are inherently limited in accuracy because of the resolution of the Z-buffer.

Based on Hook's[8] dexel method, Oliver[11], in 1994, developed a system based on a so-called spatial partitioning technique which incorporates incremental proximity calculations between milled and design surfaces. He derived a dexel representation from the dexel data structure of Hook to approximate free-form solid geometry as sets of rectan-

gular solid elements. The dexel representation of a solid is constructed in a dexel coordinate system via ray intersection and is manipulated using dexel-based Boolean set operations. The major distinction between this approach and the original dexel data structure is that the construction of dexels in not limited by the viewing vector. The independent dexel coordinate system is used to support dynamic viewing transformations. The verification process is to calculate the error between each dexel and design surfaces. The advantage of this algorithm is that the discrepancy between the dexel-based milled surfaces and actual design surfaces can be calculated during the simulation. To improve the efficiency, the calculation is performed only on dexels that are updated during simulation. Although the verification results can be displayed from other views without rerunning the simulation and verification, it in essence is still a view dependent verification algorithm, because the accuracy of the verification depends on the setup of the dexel coordinate system. Two methods were provided for specification of the dexel coordinate system, one is via interactive selection, and the other is via calculating the dexel coordinate system orientation that produces the maximum projected area of the design surfaces on the dexel plane. This means that the algorithm has the same drawbacks as those of image-based algorithms, and certain errors could not be detected, especially for multi-axis milling.

2.3 Object Space Techniques

A "point-vector" technique, which uses vectors to represent excess material removed by NC milling, was proposed by Chappel[4]. The part surface in his scheme is approximated by a set of points. Direction vectors are created parallel to the surface normal at each surface point. A vector extends until it reaches the boundary of the original

stock or intersects with another surface of the part. To simulate the cutting, the intersection of each vector with each tool movement's envelope is calculated. The length of a vector is reduced if it intersects the envelope. This method is possibly more efficient than typical solid modeling approaches, since the intermediate simulation step is simplified considerably. Chappel gives a detailed algorithm for computing the intersection between a vector and randomly oriented cylinder that represents the cutting tool. However, this algorithm is not very general because it treats only the side surfaces of cylindrical cutters. Thus, this algorithm is not widely used.

Oliver and Goodman[12]-[15], Jerard[17][18], and Chang and Goodman[16] developed various object-space approaches, in which the verification is accomplished by calculating the intersections of direction vectors with tool movement envelopes. These methods can work for any part consisting of a set of surfaces for which surface points and their corresponding normal vectors can be defined. The commonality of all the systems in this category is their view independence. That is, they can generate another view of a part without rerunning the simulation and verification. Color-coded graphics can display the machined part with its error areas and the areas within the given tolerance. In Oliver and Goodman's algorithm, the part surface is discretized into surface points and their corresponding normals; i.e., the entire part is represented by a collection of surface points and their normals. The surface discretization is based on pixel-level resolution for the view selected. To solve the problem of intersection of surface normal vectors with milling tool swept volumes for each tool motion, it creates a parallelipiped which bounds the tool swept volume for the primitive test of each surface point. Passing the primitive test means the surface point most likely intersects the swept volume. If so, the parallelipiped is refined by adding cylindrical

and/or spherical surfaces. The calculation of the intersection of the surface point and the refined swept volume is then performed. This algorithm is a viable solution to the problem of accurate and efficient geometric NC program verification. However, it is only useful for three-axis machining. Jerard developed another surface-based NC simulation and verification system. To gain efficiency, the surface curvature and cutter shape and size are used to discretize the part surface, which guarantees that a given user-defined level of simulation accuracy is achieved, at least in the Z direction. The surface points are then mapped into 2D buckets. Tool motion is projected onto the buckets. Only those surface points inside the buckets need to be examined for the intersection checking. One of the major problems for this system is that it requires a huge amount of memory because the system internally reserves a huge static array for the buckets, even though it might not need that many. Similar to Oliver and Goodman's method, this scheme was also developed for three-axis NC verification. In 1992, Chang and Goodman proposed a new approach for five-axis NC simulation and verification. Based on Goodman's previous research, the algorithm discretizes the nominal sculptured part surface and directly computes the possible interference between these surface points and the moving tool without explicitly creating the bounding surface of the tool swept volume. The geometric model of this method is based on the ruled surface defined by the axis of the cutter along each toolpath. The system is also view independent and utilizes positional tolerances for the desired part surface. The simulation time for all of the above systems grows linearly with the number of tool motions and the number of surface points discretized.

2.4 Parallel Processing Based Techniques

To achieve the maximum usefulness of today's computing resources, Menon *et al.*[22] and Yung[23] both proposed parallel NC verification methods. Menon *et al.* proposed a method of NC verification using ray representations (ray rep) in combination with the RayCasting Engine (RCE), a new highly parallel computer for processing ray representations. The combination of ray representation and RCE removes many of the current limitations in spatial verification, and extends the range of verifiable phenomena to include part tolerance assessment and machining dynamics. In Menon's scheme, the solids for the part and the cutter swept volume are both represented as collections of rays. Similarly, ray reps can be conceived for solids in 3D. The part solid can be directly sent to the RCE, while a ray rep for a swept volume is generated incrementally through 'discrete union'. The Boolean combination of the solids is straight forward. Menon described his NC verification in terms of a solids-based computation, which is implemented via ray reps. Yung's algorithm is also solid-modeling-based NC verification. He investigated the boundary surfaces of swept volumes for general rotational cutting tools undergoing multi-axis NC machining motions, and developed a massively parallel algorithm for generating geometric representations of boundary surfaces of tool swept volumes and the machined workpiece. The Jacobian determinant in the parametric space, indicating the direction of the motion relative to the surface normal of the tool boundary, is used to establish the necessary conditions for the boundary surfaces of the tool swept volumes. Yung concentrated on the construction of triangular surface representations that approximate the boundary of the tool swept volume, within specified tolerances. The parallel algorithm Yung presented, which is on a SIMD parallel computer, generates boundary representations of tool swept

volumes, performs Boolean subtractions between boundary representations of the tool swept volume and the workpiece, and renders raster images of the machined objects. A detailed description of the algorithm is in Yung[23]. Both Menon and Yung's parallel algorithms for NC verification are solid-modeling-based approaches. The algorithms takes advantage of machine power for solving the complex NC verification problem. However, the five-axis swept volumes are composed of mathematical primitives which are extremely complex, and Boolean operations on them are very compute-intensive, thus reducing the cost-effectiveness of the techniques. In addition, both of them did not fully consider the load balancing problem of the parallel computations. They show the correctness of their algorithms, but did not show the speedup and efficiency of their algorithms.

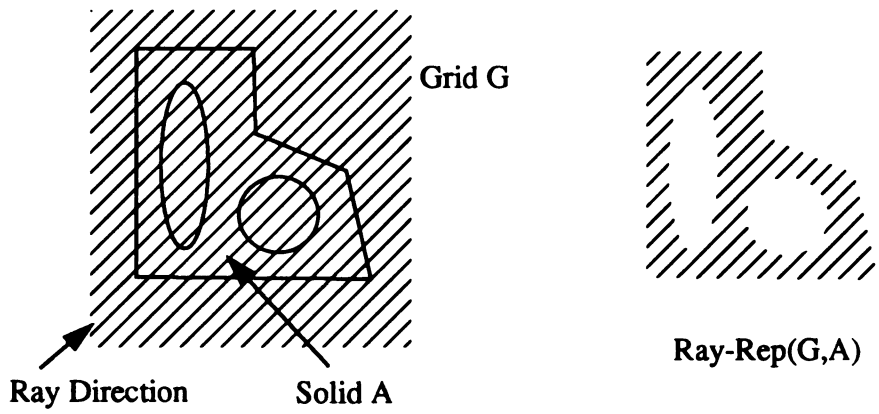


FIGURE 2.2 A Simple 2D Ray-Rep in Menon's Scheme

CHAPTER III

MATHEMATICAL REPRESENTATION OF COMPONENTS OF THE NC MILLING PROCESS

3.1 Overview of the NC Verification Model

NC verification entails two main tasks -- modeling of machining phenomena, and assessment of modeled phenomena to determine program correctness. Generally speaking, NC simulators model the machining process, but rely on human observation to detect machining errors. In contrast, verifiers detect errors by testing mathematical conditions. Good NC verifiers not only provide accurate analysis results, but also give a graphical representation of the verification results, which makes it much easier for the user to find areas where any tolerance violations occur.

This chapter introduces a model for a single tool motion and other concepts used in NC milling and verification. The parallel NC verification algorithm will be left for the next chapter. We will first give a short discussion on the characteristics of NC machining, and then describe the tool motion model which will be used in our parallel NC verification system. Most of the current multi-axis NC verification systems operate by continually creating the boundary or envelope of the tool motion through time. The verification problem is handled only as a sort of postprocessing which begins after a full simulation of the NC

process is completed. The tool motion model we employ does not need to calculate the boundary or envelope of each tool motion boundary and perform a full temporal simulation of the milling process. It detects interferences between sculpture surfaces, namely between part surfaces and tool swept volumes, on different processors (or computers) without using any Boolean operations. The mathematical model for each swept volume will be briefly described Section 3.3.

3.2 The Characterization of NC Machining

Two major components of the modeling work for NC verification are part surface modeling and cutter motion modeling. Various approaches, such as CSG methods, image space methods and object space methods, differ in this aspect. In order to apply parallel NC verification, let's examine some of the characteristics and assumptions made for the NC machining process.

For the purpose of the verification, the cutter is represented as a (symmetrical) surface of revolution, which means that the cross section at any specific position along the cutter axis has a given radius. Of course, this is not actually true of milling cutters, because they have teeth. In order to make the assumption tenable, the speed of revolution of the cutter relative to its feed rate must be large, so that the effects of the individual teeth are not large relative to the tolerances being checked. This is done purposely to simplify the modeling of the cutter for the verification. Note that, for example, if a soft material were cut with a high feed rate, this assumption could be violated.

A quite general model for cutter geometry (envelope) will be used here -- the APT 7-parameter cutter. This definition is contained in the APT[33] standard (APT is an acronym for Automatically Programmed Tool, and is an NC programming language still in wide use in the automotive and aerospace industry). The envelope of the cutter to be treated here can be expressed as a function of position (height) along the cutter axis. This is true of any 7-parameter APT cutter except one with a flat bottom. A flat bottom has a radius, but the cutter envelope is actually the plane bounded by the circle of radius R(0). Our current verification procedure handles such a flat-end cutter as a special case.

There are two types of cutter motions commonly used in manufacturing, point-to-point motions and contouring motions. Point-to-point motion is a tool motion from a start point to a destination point without specific requirements on the toolpath, and a roughly linear motion over a fairly short segment is generally assumed, but not tightly specified. A contouring motion is one that keeps the cutter moving along the toolpath with designated orientations as a function of location along the path in the case of multi-axis motions. Such complex tool motions are often subdivided into a sequence of simpler, point-to-point motions, maintaining the deviation of the tool from the path within a specified tolerance of the given contour. For instance, each step of APT contour motion can be defined by a drive surface, a part surface and a check surface. As shown in Figure 3.1, the part surface is one of two surfaces with which the cutter is in continual contact (within tolerance) during a given machining motion. The part surface is usually the surface that controls the depth of cut. The drive surface is the second surface with which the cutter is in continual contact (within tolerance) during a given machining motion. The drive surface guides the cutter through space, while a given relationship is maintained between the cutter and the part

surface. The check surface is the limiting surface for a given motion statement. The cutter maintains a specified relationship with the part surface and the drive surface until it reaches a given condition with regard to the check surface. When this occurs, a new motion statement can be specified. The surfaces to be machined are defined exactly by the control processor. However, most NC machines are capable of moving only in a straight line. Therefore, a series of straight lines that approximate the desired contour within tolerance must be generated. Many post-processing software packages in the market are capable of accomplishing this task. Although some researchers have used these three surfaces to implement an NC verification process, such an approach is quite intractable for multi-axis tool motion. There are various ways to model both types of tool motion in an NC verification process. The key to our approach is to use the centerline of the tool, as well as the envelope of the tool, to define the swept volume of the tool motion. A brief description of the algorithm will be given in next chapter.

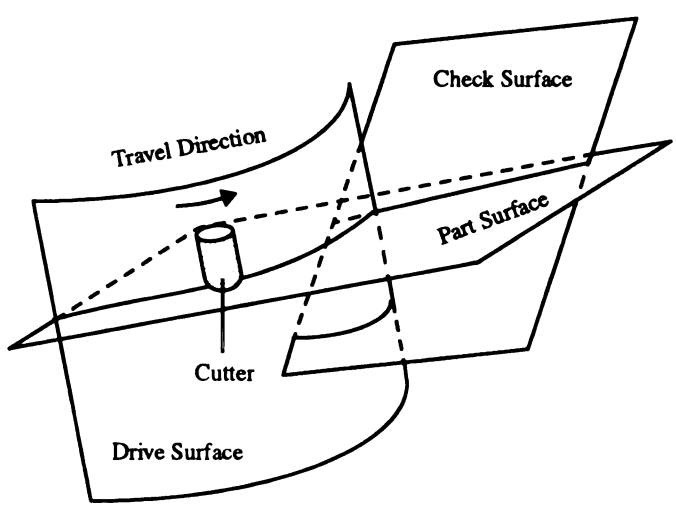


FIGURE 3.1 Part Surface, Drive Surface and Check Surface

Another modeling task of NC verification is surface modeling. In Voelcker *et al.*, [1][2] Fridshal[3], and Menon's approaches, solids are used to represent desired parts. Goodman's[15][16] and Jerard's[18] methods discretize the sculptured surfaces representing the parts to be machined into a set of surface points and normal vectors at those points. The level of discretization of the workpiece and associated holding fixtures depends directly on the size and curvature of the surfaces. The generated surface point data structure is then used as an approximation to the desired part surfaces.

In the multi-axis simulation and verification system[16] developed at Michigan State University, the surface evaluator can provide surface points and can calculate unique normal vectors for any surfaces except at slope discontinuities -- i.e., along edges and at vertices. Along edges, two normals may be defined, and at vertices, three or more normals may be defined. Surfaces need not comprise a closed solid model. The discretized surface points are further organized into a triangular grid of points, in which the resolution depends on user-specified values for maximum chordal deviation in each parametric direction.[21]

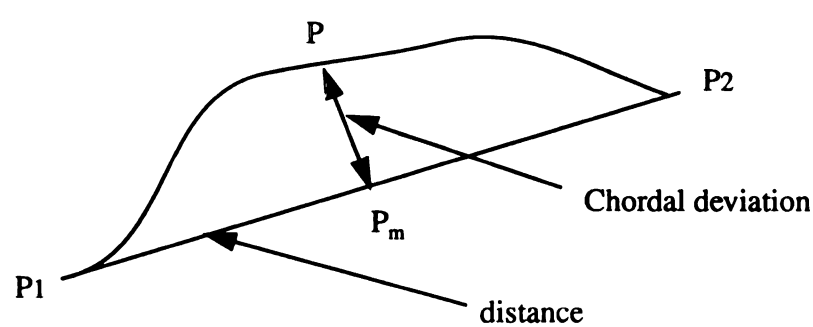


FIGURE 3.2 Chordal Deviation of Two Surface Points

The chordal deviation is defined to be the distance between the geometric midpoints P_m of two points and the point P as shown in Figure 3.2. An approximation to the chordal deviation is used as an approximation to the maximum error between a curve and a line segment joining its two endpoints. A user-specified maximum chordal deviation is used as a criterion for subdivision of parametric curves in surfaces, by introducing a new point at the parametric midpoint of any segment of a curve being examined, whenever the chordal deviation conditions are violated. At any particular time, the points are evaluated either in the u or the v parametric direction. In other words, one of the parameters is constant. If the distance between two points or the chordal deviation of two points P_1 and P_2 exceeds the user-entered limits for discretization, the curve needs to be subdivided. Recursive subdivision is conducted until all the surface points along the changing parameter (u or v) satisfy the condition.

An advantage of this approach is that surface evaluation can be done separately along each parametric flow line -- i.e., in parallel. However, this advantage also becomes the algorithm's main drawback, because the satisfaction of chordal deviation along both u and v parametric directions cannot guarantee the accuracy of surface approximation of a set of triangles formed with those edges and diagonal lines joining opposite vertices of the quadrilateral mesh of edges. We will present a new surface discretization algorithm in the next chapter, which guarantees that the surface tessellation does not violation the user-specified geometric tolerance.

We observe that tool motion, including multi-axis motion, may be modeled as a function of tool location and orientation only. For point-to-point tool motions, the swept volume can be uniquely defined given only the two end points and their (common) normal.

The NC verification process will not make any change in the tool location or orientation data. Hence, the CL data information is an ideal candidate for residence in shared memory, because it guarantees that dirty memory[28] situations will not occur. In contrast, the cut depths associated with the discretized surface points will be affected by the NC verification process. In other words, the simulation will change the location of the deepest-cut surface points. The geometric change for each surface point depends on all the tool motions that pass close enough to the point. Because of the changes, memory access, mainly memory write, will occur frequently. If the discretized surface points reside in shared memory, dirty memory is inevitable, since at a particular time, multiple processors which hold various tool motions could tend to access the same surface point. Hence memory contention happens that not only slows down the NC verification process, but also leads to network contention and dirty memory problems. The conclusion is that it is best to assign the surface data to different local memories.

3.3 Tool Motion Model

The tool definition, which specifies the shape and size of the cutter envelope when the cutter is rotating in place, the control information, such as feed rate, spindle speed, coolant switch, and the tool paths, can all be included in a CL data file. A CL data file has an ISO standard format. It is independent of the particular milling machine on which the cutting will be done. It furnishes sufficient milling information to allow NC post-processors to generate machine-specific codes for different types of NC milling machines. An NC toolpath typically consists of a list of tool positions and orientations which control a milling machine driving along the part surface. In 3-axis NC machining, the tool keeps the

same orientation for all the tool motions in a particular machine setup. For a multi-axis NC machine tool, the tool motion becomes very complicated because there is at least one rotary axis. Figure 3.3 shows the toolpath and cutter orientations for 5-axis slab milling.

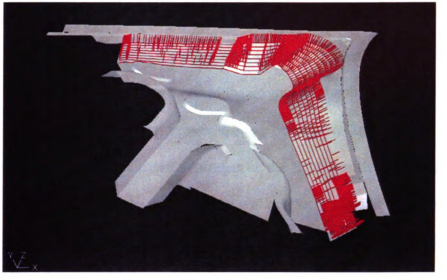


FIGURE 3.3 Example of 5-Axis Slab Cut Toolpath

As mentioned earlier, the key concept of our tool motion model is to define the translation and rotation of the tool in 3D space by using a ruled surface determined by the axis of the cutting tool. The verification computes the possible interference between discretized surface points and the moving tool without explicitly creating the bounding surface of the tool motion (the tool swept volume). Mathematically, the motion of a tool can be defined by the control point $\mathbf{C}(t)$ and the orientation of the tool axis $\mathbf{N}(t)$. The trajectory of the control point usually is rendered by linear interpolation between the previous position and the current position[5]. A series of linear motions is also frequently used to approximate more complex trajectories, such as circular motion between CL points, by using a chordal tolerance to determine the step sizes for a series of short linear segments.

There are various ways to interpolate the orientation of the cutter axis for each toolpath segment. The method employed in our system is that the previous tool axis and current tool axis are treated as two vectors on the same unit sphere[5], and interpolation is conducted along the great circle joining them. In other words, the tool vector function **N**(t) is defined by an arc on the great circle of a unit sphere passing through the two vectors, with motion at a constant velocity as shown in Figure 3.4. The unit sphere is determined by normalizing the vectors and translating one of the vectors so that the two have a common origin at the center of the unit sphere. However, on a particular milling machine, the actual interpolation for the motion of the tool axis relies on the NC controller and the post-processor used. If the function for the motion of the tool axis were a known function different from the great circle function used here, then **N**(t) could instead be specified without using the great circle approximation.

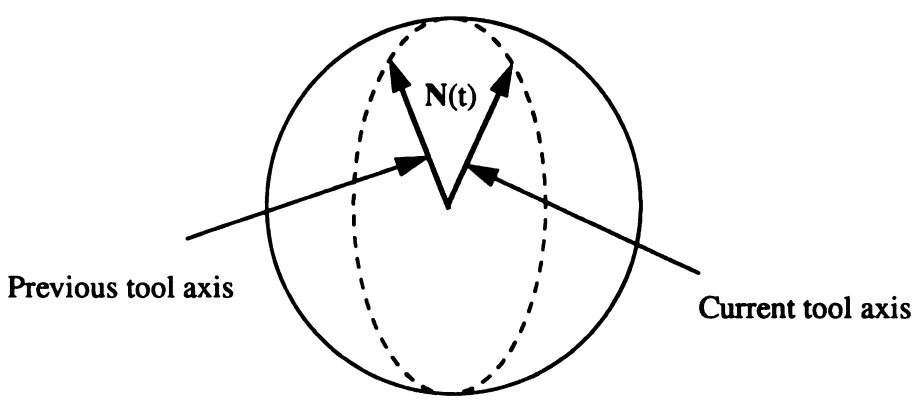


FIGURE 3.4 Orientation of Tool Axis N(t)

The locus of the tool centerline moving with one degree of freedom during a multi-axis tool move constitutes a ruled surface [29][30]. For a given $\boldsymbol{C}(t) = \{x(t), y(t), z(t)\}$

and $N(t) = \{x(t), y(t), z(t)\}$ in Cartesian coordinates, the center-line ruled drive surface for a tool motion can be parameterized as:

$$\begin{aligned} \boldsymbol{r}(t, h) &= \boldsymbol{C}(t) + h\boldsymbol{N}(t) \\ &= [x(t) + hN_x(t)]\boldsymbol{i} + [y(t) + hN_y(t)]\boldsymbol{j} + [z(t) + hN_z(t)]\boldsymbol{k} \end{aligned} \quad \text{(EQ 3.1)}$$

where $0 \le h \le L$, $0 \le t \le 1$ and L is the cutter height from the control point of the cutter as shown in Figure 3.5. For a general APT tool, the profile at any specific position along the cutter axis has a given radius and it can be expressed as a function $\boldsymbol{R}(h)$ of position h along the cutter axis. The function $R(h)$ for a general APT cutter can have up to three different functional forms for subintervals of $0 \le h \le L$. It should point out that the parameter t plays double roles in our model. On one hand, it is the parameter to define the ruled surface; on the other hand, it defines different tool positions at particular time instances t. For any given moment t, a unique line l_t on the ruled surface is determined by $\boldsymbol{r}(t, h)$, as shown in Figure 3.5, because of the generating property of the ruled surface. Now given any 3D surface point $\boldsymbol{P_i} = (x_i, y_i, z_i)$, it is straightforward to find a distance vector (the shortest vector between l_t and $\boldsymbol{P_i}$) $\boldsymbol{u}_i(t, h) = \boldsymbol{P}_i - \boldsymbol{r}(t, h_i(t))$. By applying the property of orthogonality -- *i.e.* $\boldsymbol{u}_i(t, h) \cdot \boldsymbol{N}(t) = 0$ -- to $\boldsymbol{u}_i(t, h)$, $h_i(t)$ can be determined to be the following:

$$h_i(t) = [x_i - x(t)]N_x(t) + [y_i - y(t)]N_y(t) + [z_i - z(t)]N_z(t) \quad \text{(EQ 3.2)}$$

where $\boldsymbol{P}(t) = (x(t), y(t), z(t))$ is a ruled surface point (control point) and $\boldsymbol{N}(t)$ is the tool vector at t.

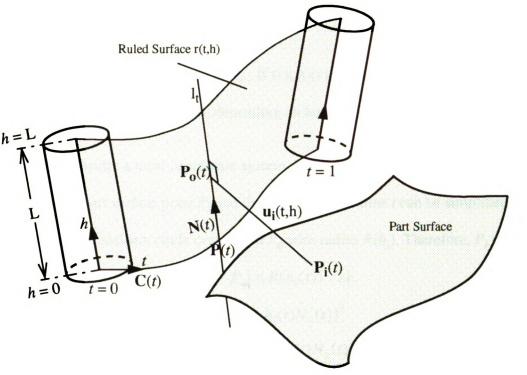


FIGURE 3.5 Tool Swept Ruled Surface

Having set up the tool motion model, we are able to determine whether or not a given surface point $\boldsymbol{P_i}$ is inside the envelop of a tool motion defined by $\boldsymbol{C}(t)$ and $\boldsymbol{N}(t)$ at particular time instant t. For any surface point $\boldsymbol{P_i}$, there is a corresponding $\boldsymbol{P_o}$ which is the closest point on the cutter axis. $\boldsymbol{P_o}$ can be defined by $h_i(t)$. Since $h_i(t)$ may or may not be within the bound of cutter [0, L], $\boldsymbol{P_o}$ is not necessarily the closest point on the ruled surface to $\boldsymbol{P_i}$, but is the closest point on the cutter axis. To explain it clearly, let's look at the problem from another view. At any given time t, we can determine a unique tool center point (control point) and its axis vector by $\boldsymbol{C}(t)$ and $\boldsymbol{N}(t)$. The center point and the vector in turn determine the line l_t on the ruled surface. For any given discretized surface point $\boldsymbol{P_i}$, there is only one point $\boldsymbol{P_o}$ on l_t which is closest to $\boldsymbol{P_i}$. $\boldsymbol{P_o}$ has two characteristics.

- The distance vector $\boldsymbol{u}_i(t, h) = \boldsymbol{P}_i - \boldsymbol{r}(t, h_i(t))$ passes through $\boldsymbol{P}_o$. In other words, $\boldsymbol{P}_o$ is the closest point on l_t to $\boldsymbol{P}_i$.

- The distance from $\boldsymbol{P}_o$ to $\boldsymbol{P}(t)$ is $h_i(t)$. If $0 \le h_i(t) \le L$ and $0 \le t \le 1$, then $\boldsymbol{P}_i$ could be affected by the tool, depending on how far it is from tool axis.

Now let's consider a local coordinate system which originates at $\boldsymbol{P}_o$. The interference check between part surface point $\boldsymbol{P}_i$ and the tool motion at time t can be simplified to checking whether $\boldsymbol{P}_i$ is inside a circle centered at $\boldsymbol{P}_o$ with radius $R(h_i)$. Therefore, $\boldsymbol{P}_i$ is affected by the tool at t if and only if $|\boldsymbol{P}_i - \boldsymbol{P}_o| \le R(h_i(t))$, i.e.

$$[x_i - x(t) - h_i(t)N_x(t)]^2 + [y_i - y(t) - h_i(t)N_y(t)]^2 + [z_i - z(t) - h_i(t)N_z(t)]^2 \le R^2(h_i(t))$$

By applying equation 3.2 and the unit $N(t)$, we can have

$$f(t) = [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2(h_i(t)) \le 0 \qquad 0 \le t \le 1 \quad \textit{and} \quad 0 \le h_i(t) \le L \tag{EQ 3.3}$$

Equation 3.3 states the condition under which the part surface point is inside or on the envelope of the tool motion at t or the part surface point is inside the swept volume of the tool motion at t.

Figure 3.6 shows a simple example -- a ball-end cutter. The tool control point at the center of the spherical part of the cutter, $f(t)$ has the following form for a given tool motion $\{\boldsymbol{C}(t), \boldsymbol{N}(t)\}$ and point $\boldsymbol{P}_i$:

$$f(t) = \begin{cases} [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - h_i^2(t) - R^2(h_i(t)) \\ \qquad 0 \le t \le 1 \quad and \quad 0 \le h_i(t) \le L \\ \\ [x_i - x(t)]^2 + [y_i - y(t)]^2 + [z_i - z(t)]^2 - R^2(h_i(t)) \\ \qquad 0 \le t \le 1 \quad and \quad -R \le h_i(t) \le 0 \end{cases} \qquad \text{(EQ 3.4)}$$

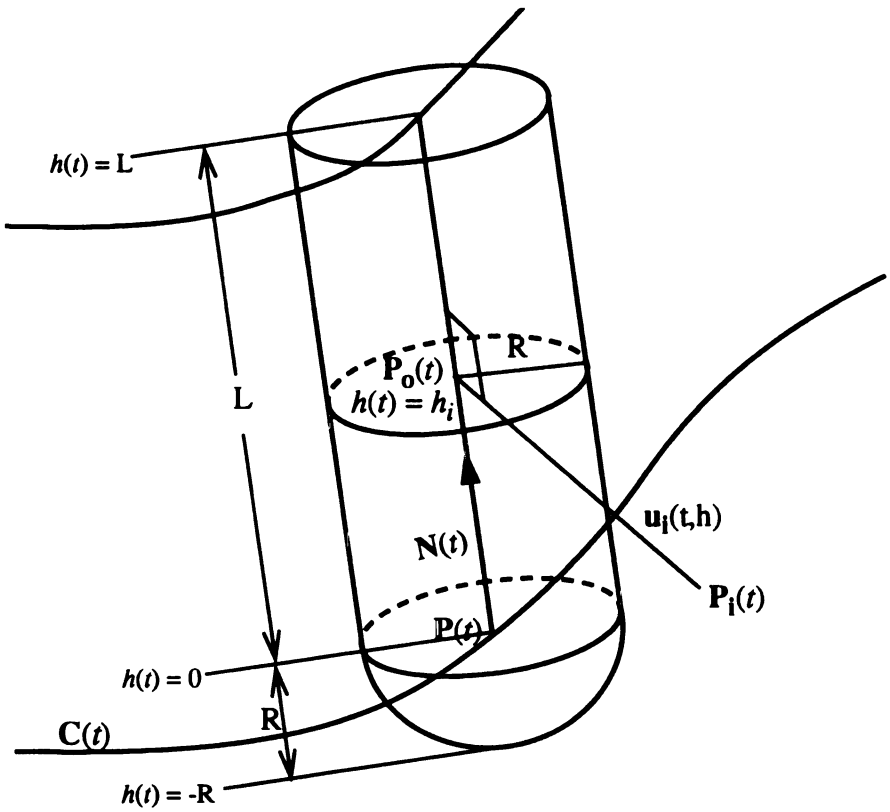


FIGURE 3.6 Interference Detection for Ball-end Cutter

The interference detection based on this model includes two steps.

- If $-R \le h_i(t) \le L$, interference could occur. Further calculation is required.
- If $f(t) \le 0$, then the part surface point P_i is inside or on the envelope of the tool motion.

The implementation of this tool motion model leads to our parallel NC verification system which will be described in a later chapter.

CHAPTER IV

SURFACE MODEL FOR PARALLEL NC VERIFICATION

Two major efforts have been developed over the years in NC simulation and verification area. One aims at a new and more efficient model to represent the NC verification process, and the other aims at methods based on advanced computing hardware. Image space approaches represent the former, while the parallel or distributed NC verification approach is of the latter type. The parallel or distributed NC verification methods distribute one NC verification process onto more than one processor to increase the verification efficiency and reduce process time. However, such methods are relatively new. The systems developed by Menon et al.[22] and by Yung[23] took advantage of the concept of parallel processing, but neither of them addressed essential issues, such as job distribution, network contention, load balance, system performance (speedup), etc., in the parallel processing. The proposed surface model will address these issues and provide reasonable load estimation for job distribution and load balance, based on geometric information. It will also guarantees that the discretized surface proximity to the original part surfaces is within the user-specified tolerance.

The NC toolpath verification process is computationally intensive and time consuming. Each of the various approaches has its own characteristics. Examination of the NC toolpath simulation and verification system[16] developed by Case Center researchers reveals the following six characteristics:

- Among all jobs of the NC verification process, the toolpath verification generally dominates system performance. It is the most time-consuming process, compared with other tasks, like loading data, surface discretization, etc. Surface discretization may also take a non-trivial amount of time especially for small value of the tessellation tolerance.
- The "as-milled" surface points receive changing cut values, changing their positions during verification.
- The toolpaths are the entities to be verified, and verification does not alter them.
- The calculations at any surface point are independent of calculations at any other surface point.
- The calculations at each surface point depend only on those toolpath segments which may pass within a certain radius r of the point, where r is the maximum radius of the cutter used.
- Toolpath segments are linear segments and irregular.

These six proposed characteristics of the NC verification process establish the basis for a parallel NC verification model. They explain why we should place CL data in shared memory and surface data in local processors' memories. To achieve high parallelism, we have decided to make each processor not only conduct a verification task, but also perform surface discretization, since it also represents a significant load on the system.

Having presented the tool motion model in the last chapter, we will introduce a new surface model in this chapter. We will first give an overview of B-spline and NURB surfaces. Then the novel mathematical surface model and surface tessellation algorithm will be introduced. Finally, the parallel NC verification algorithm will be presented. The implementation and the discussion of the algorithm will be left to the next chapter.

4.1 B-Spline Curves and NURB Surfaces

To help the reader understand the surface model used in our parallel NC verification scheme, we first provide a brief overview of B-spline and NURB surfaces -- their definitions and some useful properties.

A B-spline is a parametric spline or curve which consists of one or more segments. Each segment represents the curve over a range of the parameter t. The curve is partially defined by a set of control points $[\mathbf{P_0}, \mathbf{P_1}, \mathbf{P_2}, \ldots \mathbf{P_n}]$ which produce an open sided polygon when connected sequentially by straight lines. The general shape of the B-spline follows the shape of this polygon, and each segment of the curve is defined by a subset of the control points. Figure 4.1 shows a control point polygon with its three-segment B-spline curve. Segment ends are denoted with an $\times$. Noted beside each segment are the control points that are used to compute that particular segment.

Each control point $\mathbf{P_i}$ has a blending function $N_{i,k}(t)$ associated with it, where the subscript k refers to the order of the blending function and B-spline. The blending function $N_{i,k}(t)$ associated with control point $\mathbf{P_i}$ will change from segment to segment. The general form of a B-spline segment is defined by

$$\boldsymbol{P}(t) = \sum_{i=0}^{n} \boldsymbol{P}_i N_{i,k}(t) \qquad \text{(EQ 4.1)}$$

The summation goes from $i = 0$ to $i = n$ which means that there are $n+1$ control points. The order k of the B-spline is its degree plus 1. The fact that only a subset of the

total set of control points is used for each segment implies that the blending functions associated with other unused control points are all zero.

The blending functions are determined by a set of values called a knot vector -- for example, knot vector = $[x_0\, x_1\, x_2\, \ldots\ldots\, x_l]$. The relationship between the knot vector and the blending functions is defined as follows:

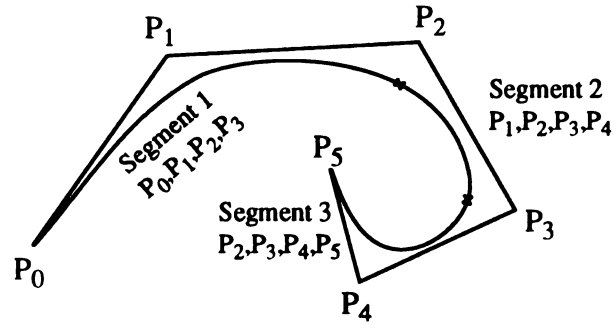


FIGURE 4.1 2D B-spline Curve

$$\begin{cases} N_{i,1}(t) = \begin{cases} 1 & x_i \le t < x_{i+1} \\ 0 & otherwise \end{cases} \\ \\ N_{i,k}(t) = \dfrac{(t - x_i)N_{i,k-1}(t)}{x_{i+k-1} - x_i} + \dfrac{(x_{i+k} - t)N_{i+1,k-1}(t)}{x_{i+k} - x_{i+1}} \end{cases} \qquad \text{(EQ 4.2)}$$

where x_i is the ith knot vector element. If $x_i = x_{i+1}$, then $N_{i,1}(t) = 0$. This notation also uses the convention that 0/0 = 0 in the blending function evaluation. There is only one $N_{i,1}(t)$ that is nonzero for each segment. This nonzero function cascades down to the final blending functions associated with that segment as shown in Figure 4.2. Notice that the number of nonzero blending functions per segment is equal to the order

of the B-spline curve. There are two main properties of blending functions which should be mentioned as well.

- $\sum_{i=0}^{n} N_{i,k}(t) = 1$ for each segment.
- $N_{i,k}(t) \geq 0$ for all t.

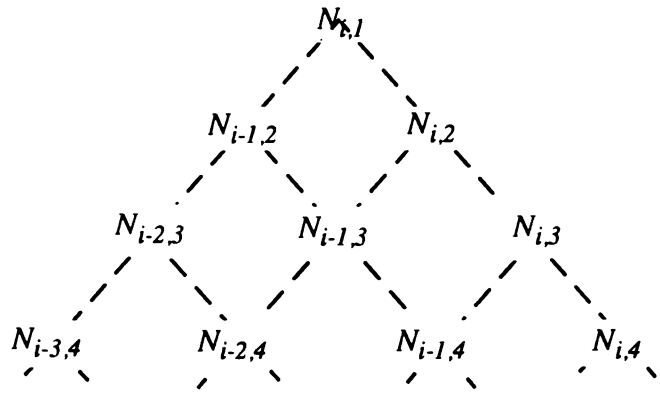


FIGURE 4.2 Blending Function Cascade

In most cases, it is required that the B-spline pass through the endpoints of the control point polygon. However, to make the curve pass through the endpoints, the first and last entries of the knot vector must repeat k times, i.e. $x_0 = x_1 = ... = x_{k-1}$ and $x_{l-(k-1)} = ... = x_{l-1} = x_l$. The length of the knot vector ($L = l + 1$) is a function of the number of control points ($n + 1$) and the order (k) of the B-spline. It equals the number of control points plus the order of the B-spline, *i.e.*, $L = n + k + 1$. The order of the curve must be less than or equal to the number of control points. The number of segments of the B-spline equals the number of control points minus the degree of the curve, i.e. $n - k + 2$. The parameter range associated with each segment can be determined by examining the knot vector.

Control points help to control the shape of the B-spline, and the blending functions, which are determined by the knot vector, control how much each control point contributes to each B-spline segment. When the blending functions for a segment of a B-spline are computed, only a subset of the knot vector is used. These values form what is called the effective knot vector. The length of the effective knot vector is a function of the order k. The relationship is: $l_e = 2(k-1)$.

Internal knot vector elements can also be repeated. Repeating an internal knot produces a zero-length segment and introduces a derivative discontinuity at the segment joint corresponding to that knot. In order words, repeating a knot once causes the curve to lose the highest order derivative continuity at the segment joint corresponding that knot. Each repetition of a knot decreases the number of nonzero segments by one. Thus for a uniform knot vector, each repetition causes the end knot or maximum parameter value to decrease by one. A knot may be repeated at most k times, where k is the order of the curve.

EQ. 4.1 shows that a B-spline curve depends on its control points and blending functions, which in turn depends on the knot vector. Different forms of control point and knot vector representations lead to the classification of B-spline curves. The representation of control points defines whether the curve is *rational* or *nonrational*, while the knot vector defines whether the curve is *uniform* or *nonuniform*.

If the B-spline control points are expressed in homogeneous coordinates, it is denoted a *rational* B-spline. There are an infinite number of homogeneous representations of a single coordinate point -- for example, the three dimensional Cartesian coordinate point (1, 2, 3) can be represented by (1, 2, 3, 1) = (2, 4, 6, 2) = (5, 10, 15, 5). The fourth

position or coordinate is the homogeneous variable. If the homogeneous variable is equal to 1, then the other variables represent the actual Cartesian coordinates. If the homogeneous variable is not equal to 1, say *h*, then to transform back to Cartesian coordinates, each term must be divided by *h*. For rational B-splines, the value of the homogeneous variable is called the weight. Each control point has its own weight. If all of the weights are 1.0, then the curve will be the same as the *nonrational* curve -- *i.e.*, the control point coordinates expressed in ordinary Cartesian coordinate form. Expressing the control points in homogeneous coordinates has no effect on the blending functions, which depend only on the knot vector.

The *uniform* or *nonuniform* B-spline curve depends on the knot vector being *uniform* or *nonuniform*. *Uniform* knot vectors have integer values as knots and the interior knots are consecutive integers. Thus each segment of the *uniform* B-spline has a "uniform" unit change in the parameter value over its length. The use of real (non-integer) values as knots, or the occurrence of repeated interior knots, produce *nonuniform* knot vectors. The term *nonuniform* is predominantly used to signify real knot values which produce unequal parameter changes per segment, such as [0 0 0 .3 .8 1 1 1]. An integer knot vector with repeated knots, and thus segments with zero parameter change, such as [0 0 0 1 1 2 2 2], is sometimes referred to as an "enhanced" *uniform* knot vector instead of being called simply *nonuniform*[20]. The ability of the *nonuniform* B-spline to have each segment cover a different parameter range is the main reason for their use.

B-spline surfaces are an extension of B-spline curves. The use of B-splines to generate surfaces follows easily from an understanding of the curve. The most commonly used surface representation is the four-sided patch as shown in Figure 4.3. Each point on

the patch or surface is a function of two parameters, u and v. Figure 4.3 shows the mapping from parametric space (u, v) to 3D geometric space. One side of the patch and its opposite side are chosen as curves that depend only on u. The other two edges are functions only of v. Both u-varying sides share the same knot vector, as do both v-varying sides. The u knot vector does not have to be identical to the v knot vector; that is, they can be of different orders and have different numbers of control points. Each edge has a control point polygon. Additional control points determine the interior shape of the patch. If the u edges have m+1 control points and the v edges have n+1, then there will be a total of $(m + 1) \times (n + 1)$ control points. The control points joined together by straight lines form what is called a control point net.

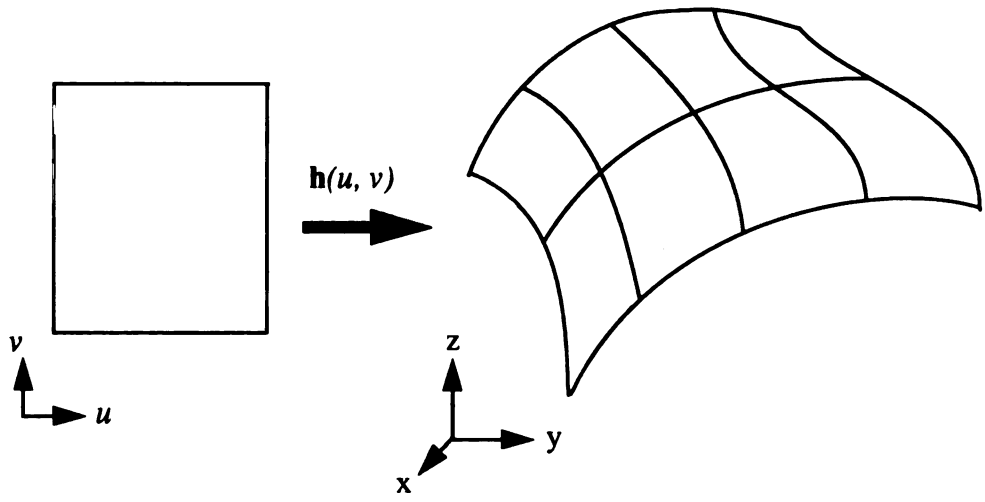


FIGURE 4.3 Surface Mapping Between Parametric Space and 3D Space

Just as the B-spline curve consisted of segments, the B-spline surface consists of segments known as subpatches. From earlier discussion, we see there are (m-k1+2) u direction segments if the u edge B-splines have m+1 control points and are of order k1,

and that there will be (n-k2+2) v direction segments if the *v* edge B-splines have n+1 control points and are of order k2. Therefore the total number of subpatches is (m-k1+2)*(n-k2+2). The general formula for a B-spline surface is:

$$S(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} P_{i,j} w_{i,j} M_{i,k1}(u) N_{j,k2}(v)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{i,j} M_{i,k1}(u) N_{j,k2}(v)} \qquad \text{(EQ 4.3)}$$

where M and N are two blending functions and $w_{i,j}$ is the weight of control point $\mathbf{P}_{i,j}$.

Likewise, B-spline surfaces can be classified into *rational/nonrational*, *uniform/nonuniform* surfaces. The definitions of these terms are exactly parallel to those described previously for space curves. The terms *NURB surface* or *NURBS surface*, which are widely used in industry, stand for NonUniform Rational B-Spline surface.

4.2 Chordal Deviation and the Surface Model

Two major components of the modeling work for NC verification are part surface modeling and cutter motion modeling. Various approaches, such as CSG methods, image space methods and object space methods, differ in this area. The previous chapter discussed tool motion modeling. Now we are ready to present a new efficient surface model for a parallel (or distributed) NC verification system. In Voelcker *et al.*, [1][2] Fridshal[3], and Menon's approaches, solids are used to represent desired parts. Goodman's[15][16] and Jerard's[18] methods discretize the sculptured surface, which is widely used to represent parts to be machined, into a set of surface points, with the location and normal for each

surface point. The level of discretization of workpiece and associated holding fixtures depends directly on the size and curvature of the surfaces. The generated surface point data structure is then used as an approximation to the desired (nominal) part surfaces.

In the multi-axis simulation and verification system[16], the surface evaluator can provide surface points and can calculate unique normal vectors for any surfaces except at slope discontinuities -- *i.e.*, along edges and at vertices. Along edges, two normals may be defined, and at vertices, three or more normals may be defined. Surfaces need not comprise a closed solid model. The discretized surface points are further organized into a triangular grid of points, in which the resolution depends on user-specified values for maximum chordal deviation in each parametric direction.[21]

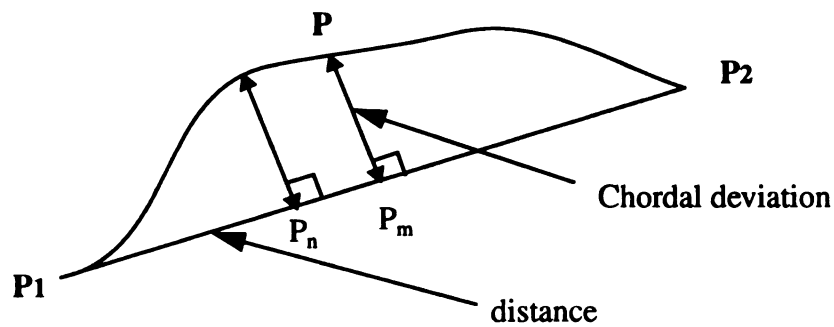


FIGURE 4.4 Chordal Deviation of Two Surface Points

The chordal deviation plays very important roles in surface discretization. It is defined to be the distance between the geometric midpoint $\mathbf{P_m}$ of two points, $\mathbf{P_1}$ and $\mathbf{P_2}$ and the point **P** as shown in Figure 4.4. An approximation to the chordal deviation is used as an approximation to the maximum error between a curve and a line segment joining its two endpoints (it would be exact if the parameterization were uniform and the curve were

an arc of a circle). A user-specified maximum chordal deviation is used as a criterion for subdivision of parametric curves in surfaces. Whenever the chordal deviation conditions are violated, a new point at the parametric midpoint of the curve segment being examined is introduced. At any particular time, the points are evaluated either in the *u* or the *v* parametric direction. In other words, one of the parameters is constant. If the distance between two points or the chordal deviation of two points $\mathbf{P_1}$ and $\mathbf{P_2}$ exceeds the user-entered limits for discretization, the curve needs to be subdivided. Recursive subdivision is conducted until all the surface points along the changing parameter (*u* or v) satisfy the condition. Note that the chordal deviation at $\mathbf{P_m}$, the geometric midpoint of $\mathbf{P_1}$ and $\mathbf{P_2}$, is not necessarily the maximum chordal deviation of the B-spline considered. As shown in Figure 4.4, the maximum chordal deviation is at $\mathbf{P_n}$ instead of $\mathbf{P_m}$. Although there is a straightforward way to solve for the maximum chordal deviation by recursively evaluating $\mathbf{P_m}$, it would require a huge amount of computational time. To reduce computational costs, we instead sample several points and evaluate the chordal deviation at these points. The number of sampled points is determined by the order of the given B-spline.

As was the case in the previous scheme just described, the parallel NC verification system will still use discretized surface points along with surface triangles as the surface model. An advantage of discretizing a part surface along its parametric flow lines is that surface evaluation can be done *separately* along each line -- *i.e.*, in parallel. However, this advantage also becomes the algorithm's main drawback, because the satisfaction of chordal deviation along both *u* and *v* parametric directions cannot guarantee the accuracy of surface approximation of a set of triangles formed with those edges and diagonal lines joining opposite vertices of the quadrilateral mesh of edges. In other words, the satisfac-

tion of the chordal tolerance condition in curve evaluation along both u and v directions does not imply its satisfaction by a set of triangular planes (a faceting of the surface). We will present a new surface discretization algorithm in next section which guarantees that the surface tessellation does not violate the user-specified geometric tolerance. Meanwhile, it gives the estimation of the work load for processing each particular surface. This algorithm can be further used in some other area such as FEM, mass analysis, etc.

We have observed that tool motion, including multi-axis motion, may be modeled as a function of tool location and orientation only. For point-to-point tool motions, the swept volume can be uniquely defined given only the two end points and their (common) normal. The NC verification process basically needs to compute the minimum distance between the designed surface and its corresponding machined surface along the surface normal vector to the boundary of the tool motion. This minimum signed distance can be defined as the so-called *cut value* or *cut depth* of the surface point. The NC verification process will not make any change in the tool location or orientation data. Hence, the CL data information is an ideal candidate for residence in shared memory, because it guarantees that dirty memory[28] situations will not occur. In contrast, the cut depths associated with the discretized surface points will be affected by the NC verification process. In other words, the simulation will change the location of the deepest-cut surface points. The geometric change for each surface point depends on all the tool motions that pass sufficiently close (a dynamic condition) to the point. Because of the changes, memory access, mainly memory write, will occur frequently. If the discretized surface points reside in shared memory, memory access contention is inevitable, since at a particular time, multiple processors which hold various tool motions could intend to access the same surface point. To

handle this problem, atomic process function on the surface point is required. All the involved processors except the one who is performing the calculation for the surface point have to be either waiting or revisiting the surface points. This not only slows down the NC verification process, but also leads to network contention and dirty memory problems. The conclusion is that it is best to assign the surface data to different local memories.

4.3 Surface Discretization Algorithm for Parallel Processing

Surface discretization or subdivision is a technique for tessellating parametric surfaces, which generates approximating polyhedrons to represent original designed surfaces for applications such as NC part surface modeling, mesh generation, etc. One surface discretization approach is to generate a set of small triangles or quadrilaterals in parametric space and then map them to the Euclidean space through mapping the vertices of the parametric triangles or quadrilaterals to the desired surface to form the approximating surface. The approximating surface actually consists of a set of flat polyhedrons or triangles. In other words, flat subpatches are used to construct the approximating surface. To guarantee the goodness of the approximation of the surface, all the subpatches of the approximating surface should be within a given tolerance ε. Finer approximating surfaces require more subdivision and mapping steps.

The two main purposes of the parallel surface discretization algorithm presented in this section are to improve the quality of surface approximation, and to exploit parallelism in sculptured surface based manufacturing processes. The algorithm guarantees that the approximating surface, consisting of a set of polyhedrons, is within the required accuracy

or tolerance. It is also able to provide the necessary information to solve the potential load balance problem for parallel processing or distributed processing.

There are two major types of subdivision techniques. One is called adaptive subdivision, which can generate an approximation surface that is within the required tolerance. This method is very time consuming because of all the flatness testing required during the subdivision process. The other technique is to estimate a subdivision depth ahead of the subdivision process. This subdivision depth guarantees the required flatness of the subpatches after the subdivision. Therefore, flatness testing is not required during the subdivision process. Filip *et. al*[25] and Cheng[24] proposed widely adopted methods for estimating the subdivision depths for parametric surfaces. Filip *et. al.*'s algorithm requires direct estimation of the *sup*'s of the second partial derivatives over the domain of the surface. This is straightforward for polynomial B-spline surfaces, but is tedious and painful work for rational B-spline surfaces because the estimation of *sup*'s of the second derivatives requires computing the third partial derivatives, finding their roots, and evaluating the second partial derivatives at these roots. Cheng extended Filip *et al.*'s algorithm by transferring rational B-spline surfaces to polynomial B-spline surfaces under the standard perspective projection. Cheng's algorithm successfully avoids direct evaluation of the *sup*'s of the second partial derivatives for the given rational surfaces. However, his algorithm fails to give the expected results due to the sensitivity of the weights of the rational B-spline surface because of the perspective transformation. Based on Filip *et al.*'s and Cheng's work, we present a new algorithm which improves both surface discretization and parallelization. It guarantees the proximity of the approximating surface and the desired surface. It is also helpful to pre-estimate the computational load of NC verification and give a

worst-case performance evaluation for parallel processing. Furthermore, it can guide the job distribution on a parallel machine or distributed system. We first give the definitions of some problem-related terminology, as follows:

Given an E^n (Euclidean n-space, $n \geq 3$) parametric surface S(u,v) defined over a 2-D quadrilateral Q.

Let ε be a user-specified surface discretization limit ($\varepsilon > 0$) on S(u,v). It means that the distance between any surface point $\boldsymbol{P}$ and its corresponding point $\tilde{\boldsymbol{P}}$ on the approximating polyhedral surface must be smaller than ε; i.e., $|P - \tilde{P}| < \varepsilon$.

Let $\Delta Umax$ and $\Delta Vmax$ be the maximum allowed Δ values for u and v. These parameters determine the maximum steps in parametric space. They implicitly indicate a lower bound on the number of discretized surface points required so that the approximated surface is close to the desired surface within limit ε. This is especially important for flat surfaces, since flat surfaces could have arbitrarily large Δ values. In most cases on surfaces with high curvature, the required Δ values are normally less than these limits, so they do not come into effect there.

Let $\Delta Uest$ and $\Delta Vest$ be the estimated allowable Δ values for u and v. These parameters are obtained by the algorithm that will be presented later in this section. To discretize a given surface, our objective is to find $\Delta Umin = min(\Delta Uest, \Delta Umax)$ and $\Delta Vmin = min(\Delta Vest, \Delta Vmax)$ in order to get a good approximation of the surface and good estimate of workload for parallel or distributing processing.

A midpoint subdivision of quadrilateral Q is the process of subdividing Q into four subquadrilaterals along the midpoints of its edges. The number of recursions of the midpoint subdivision process is called the subdivision depth n. Through recursion n times, the parametric surface can be split into a fine, locally quite uniform mesh.

[24]Let T be a subtriangle of Q with vertices $\mathbf{v}_i(u_i, v_i)$, i=0,1,2, and let $\mathbf{h}(u, v)$: $T \rightarrow E^n$ be a linearly parametrized triangle in n-space with vertices $\mathbf{S}(\mathbf{v}_i)$, i=0,1,2. A triangular subpatch of S is said to be within ε if

$$\sup_{(u,v) \in T} \inf_{(u',v') \in T} \|S(u, v) - h(u', v')\| \leq \varepsilon$$

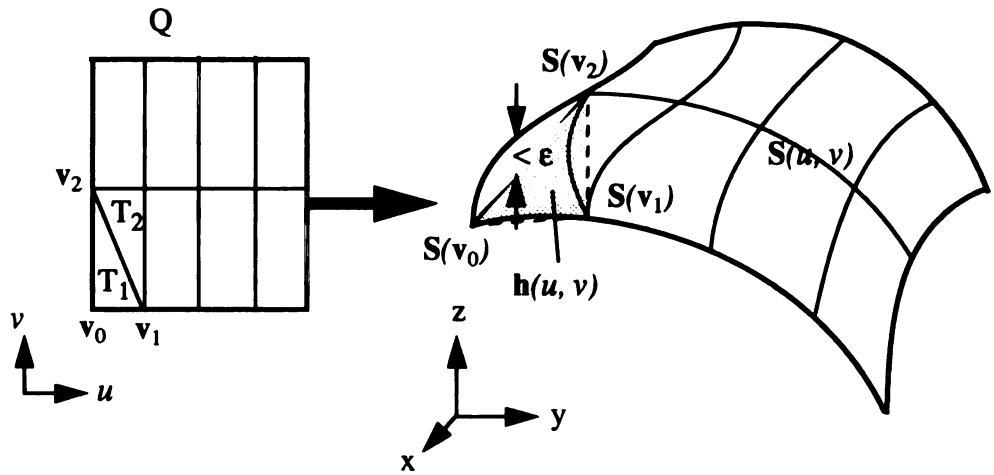


FIGURE 4.5 Transformation from Parametric Space to E^3

Filip *et. al*[25] proved that if T is a right triangle in parametric space with $\mathbf{v}_1 = \mathbf{v}_0 + (\Delta u \;\; 0)$ and $\mathbf{v}_2 = \mathbf{v}_0 + (0 \;\; \Delta v)$, which is the case in most surface applications, and if $\mathbf{S}(u,v)$ and $\mathbf{h}(u, v)$ are defined as above, then the following equation is true.

$$\sup_{(u,v)\in T} \|S(u,v) - h(u,v)\| \le \frac{1}{8}(\Delta u^2 M_1 + 2\Delta u \Delta v M_2 + \Delta v^2 M_3) \qquad \text{(EQ 4.4)}$$

where

$$M_1 = \sup_{(u,v)\in [0,1]\times[0,1]} \left\| \frac{\partial^2}{\partial u^2} S(u,v) \right\|$$

$$M_2 = \sup_{(u,v)\in [0,1]\times[0,1]} \left\| \frac{\partial^2}{\partial u \partial v} S(u,v) \right\|$$

$$M_3 = \sup_{(u,v)\in [0,1]\times[0,1]} \left\| \frac{\partial^2}{\partial v^2} S(u,v) \right\|$$

[24]Let Q' be a subquadrilateral of Q after n levels of recursive midpoint subdivision. $\mathbf{S}(Q')$ is said to be within ε if the triangular subpatches $\mathbf{S}(T_1)$ and $\mathbf{S}(T_2)$ are both within ε, where T_1 and T_2 are subtriangles generated by splitting Q' along one of its diagonals. Furthermore, a subdivision depth n guarantees the ε-closeness of $\mathbf{S}$ if $\mathbf{S}(Q')$ is within ε tolerance for all the subquadrilaterals Q' of after n levels of recursive midpoint subdivision.

The polyhedron based approximation surface can be obtained by recursively subdividing Q into small subquadrilaterals and replacing each triangular subpatch with corresponding triangle $\mathbf{h}(u, v)$: $T \rightarrow E^n$. The closeness of the approximating surface to the desired surface $\mathbf{S}$ is controlled by the subdivision depth n. The proximity is within tolerance ε if the subdivision depth guarantees the ε-closeness of $\mathbf{S}$. Clearly, the subdivision depth should be just large enough to provide the required closeness. Otherwise, the surface

might be over-subdivided. However, in the case of a very flat surface, we also impose conditions $\Delta u = \Delta Umax$ and $\Delta v = \Delta Vmax$ in order to place an upper limit on the triangle size even for flat surfaces.

Let $S(u, v)$ be a rational B-spline surface with degree $m \times n$ ($m = k1 - 1$ and $n = k2 - 1$, where $k1$ and $k2$ are the order of surface in u and v direction separately), with control points $P_{i,j}$, weights $w_{i,j}$ and blending functions $M_{i,k1}(u)$ and $N_{j,k2}(v)$

$$S(u, v) = \frac{\sum_{i=0}^{m} \sum_{j=0}^{n} P_{i,j} w_{i,j} M_{i,k1}(u) N_{j,k2}(v)}{\sum_{i=0}^{m} \sum_{j=0}^{n} w_{i,j} M_{i,k1}(u) N_{j,k2}(v)}$$

Consider $S(u, v)$ to be the image, under the standard perspective projection **F**: $E^4 \rightarrow E^3$, of the E^4 polynomial surface $\hat{S}(u, v)$. We have

$$\hat{S}(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \hat{P}_{i,j} M_{i,k1}(u) N_{j,k2}(v) \qquad \text{(EQ 4.5)}$$

where $\hat{P}_{i,j} = \begin{bmatrix} P_{i,j} w_{i,j} \\ w_{i,j} \end{bmatrix}$

As we described earlier, once a rational B-spline surface is given, homogeneous representation of control points as well as the knot vector are then determined. Expressing the control points in homogeneous coordinates has no effect on the blending functions, which depend only on the knot vector. Instead, it adds one more degree of freedom per control point to the surface and thereby allows the representation of surfaces that are impossible to represent exactly with nonrational B-splines. The homogeneous representa-

tion of control points also provides a key for surface estimation. Two useful values, minimum weights w_{min} and maximum norm of control points p_{max}, can be derived from it. The definitions are given as follows:

$$w_{min} = min\{w_{i,j} \mid 0 \le i \le m, 0 \le j \le n\} \text{ and} \quad \text{(EQ 4.6)}$$

$$p_{max} = max\{\|P_{i,j}\| \mid 0 \le i \le m, 0 \le j \le n\} \quad \text{(EQ 4.7)}$$

Similarly, we can define w_{max} and p_{min}. Since $w_{min} \le w \le w_{max}$ and $p_{min} \le \sqrt{\left(\frac{x}{w}\right)^2 + \left(\frac{y}{w}\right)^2 + \left(\frac{z}{w}\right)^2} \le p_{max}$, it is possible to find a surface boundary domain D as follows:

$$D \equiv \left\{ P = (x, y, z, w)^t \,\middle|\, w_{min} \le w \le w_{max}, p_{min} \le \frac{1}{w}\sqrt{x^2 + y^2 + z^2} \le p_{max} \right\}$$

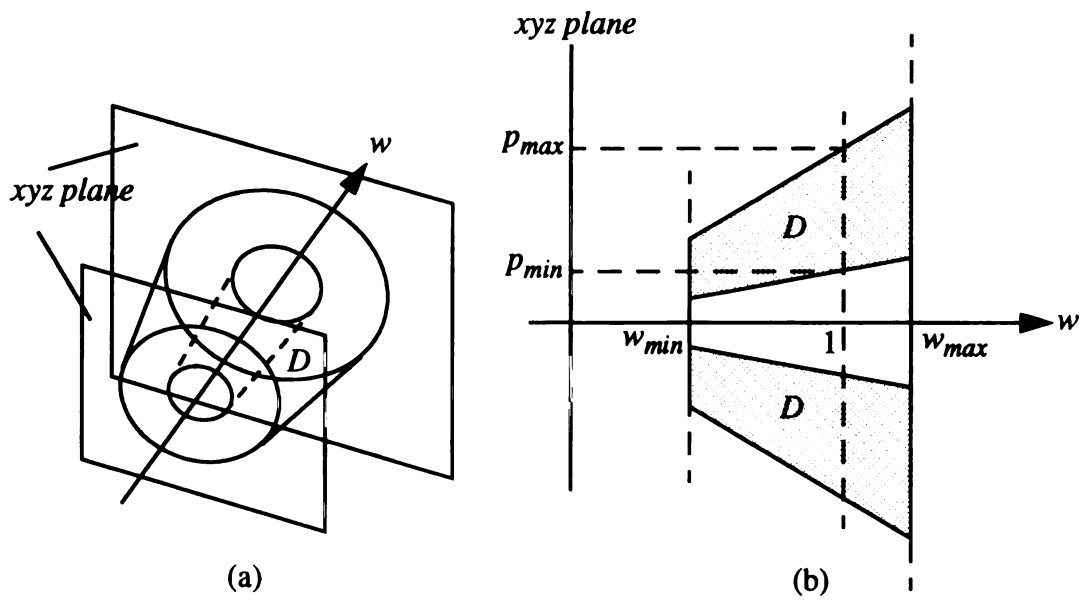


FIGURE 4.6 Surface Bounds and Weights

Examining this situation, it is obvious that D is an E^4 truncated hypercone which determines surface upper and lower bounds because rational B-spline surfaces satisfy the convex hull property. To understand this discussion, picture a 3D graph with one w axis and an xyz plane containing surface bounds for a particular w, as shown in Figure 4.6. Figure 4.6 (a) is a perspective view, and (b) is the side view. Based on the definition of D, we know that $wp_{min} \le \sqrt{x^2 + y^2 + z^2} \le wp_{max}$, which determines the truncated hypercone. At each particular value of w, two hypercircles are obtained. These two hypercircles define the boundary of the surface points for the given weights. In fact, $\sqrt{x^2 + y^2 + z^2}$ is the distance of a surface point from the w axis. When $w = 1$, the surface bounds equal the control point bounds. In other words, the whole surface is within the control point bounds. When w decreases, the surface bounds shrink and surface estimation becomes more difficult.

Cheng[24] gave two properties based on polynomial surfaces $\hat{S}(u, v)$. He proved that for any given $\varepsilon > 0$, there exists a $\delta > 0$ which is the length of the shortest line segment P_1P_2 which lies in D and is projected onto a segment $S(P_1)S(P_2)$ of length ε. Such a δ can be computed by

$$\delta = \begin{cases} \dfrac{w_{min}\varepsilon}{(1 + (p_{max} - \varepsilon)^2)^{1/2}} & 0 < \varepsilon \le p_{max} \\ w_{min}\varepsilon & p_{max} < \varepsilon \le 2p_{max} \\ +\infty & \varepsilon \le 2p_{max} \end{cases} \qquad \text{(EQ 4.8)}$$

Cheng also stated an important property -- that a subdivision depth n which guarantees the δ-closeness of $\hat{S}(u, v)$ would guarantee the ε-closeness of $S(u, v)$, where δ is defined as above and ε is a user-specified surface discretization tolerance. We will

directly apply this properties as well as EQ. 4.8 in our algorithm because $\hat{S}(u, v)$ is polynomial surface.

Unlike what Cheng did in his approach, we want to consider the subdivision depths in the u direction and v direction separately, instead of computing a single common subdivision depth. This is because the necessary iterations of midpoint subdivision processes in the u direction and v direction are generally different. More unnecessary marching steps along either the u or v direction implies much more unnecessary computational work is needed.

Since $\hat{S}(u, v)$ is polynomial surface, we can then directly use M_1, M_2 and M_3 defined by Filip *et al.*[25] to make our parametric space surface approximation based on second derivatives.

$$M_1 = \sup_{(u,v) \in [0,1] \times [0,1]} \left\| \frac{\partial^2}{\partial u^2} \hat{S}(u, v) \right\|$$

$$M_2 = \sup_{(u,v) \in [0,1] \times [0,1]} \left\| \frac{\partial^2}{\partial u \partial v} \hat{S}(u, v) \right\|$$

$$M_3 = \sup_{(u,v) \in [0,1] \times [0,1]} \left\| \frac{\partial^2}{\partial v^2} \hat{S}(u, v) \right\|$$

As given in de Boor[26], the n-th derivative of a B-spline curve

$$C(u) = \sum_{i=0}^{m} P_i M_{i,k}(t)$$

can be expressed as

$$\frac{d^n}{du^n} C(u) = (k-n)...(k-1) \sum_{i=n}^{m} P_i^{(n)} M_{i,k-n}(t)$$

where

$$P_i^{(n)} = \frac{P_i^{(n-1)} - P_{i-1}^{(n-1)}}{u_{i+k-n} - u_i}$$

By applying the above equation and the convex hull property of B-spline surfaces, the following relations can be obtained:

$$M_1 \le (k1-2)(k1-1) \max_{2 \le i \le m} \max_{0 \le j \le n} \left| \frac{a_u \hat{P}_{i,j} - (a_u + b_u)\hat{P}_{i-1,j} + b_u \hat{P}_{i-2,j}}{a_u b_u c_u} \right| \quad \text{(EQ 4.9)}$$

$$M_2 \le (k1-1)(k2-1) \max_{1 \le i \le m} \max_{1 \le j \le n} \left| \frac{\hat{P}_{i,j} - \hat{P}_{i-1,j} - \hat{P}_{i,j-1} + \hat{P}_{i-1,j-1}}{b_u b_v} \right| \quad \text{(EQ 4.10)}$$

$$M_3 \le (k2-2)(k2-1) \max_{2 \le j \le n} \max_{0 \le i \le m} \left| \frac{a_v \hat{P}_{i,j} - (a_v + b_v)\hat{P}_{i,j-1} + b_v \hat{P}_{i,j-2}}{a_v b_v c_v} \right| \quad \text{(EQ 4.11)}$$

where

$$\begin{aligned} a_u &= u_{i+k1-2} - u_{i-1} & a_v &= v_{j+k2-2} - v_{j-1} \\ b_u &= u_{i+k1-1} - u_i & b_v &= v_{j+k2-1} - v_j \\ c_u &= u_{i+k1-2} - u_i & c_v &= v_{j+k2-2} - v_j \end{aligned} \quad \text{(EQ 4.12)}$$

Refer to Appendix B for the proof of EQ. 4.9, EQ. 4.10 and EQ. 4.10

The physical meaning of M_1, M_2 and M_3 are the maximum curvatures of surface $\hat{S}(u, v)$. They also provides an estimation of necessary stepovers in the u and v directions for surface tessellation. If $M_1 = 0$, it means that the surface is *flat* in the u direction -- *i.e.*, the u direction is linear. Likewise, we say the surface is *flat* in the v direction if $M_3 =$

0. If the *u* direction is *flat* and the *v* direction is not, such as occurs in a cylinder, then the necessary stepover size, or number of recursive subdivisions, in the *u* direction during surface evaluation depends on $\Delta Umax$. Similarly, if the v direction is *flat* and the *u* direction is not, the necessary stepover size in the *v* direction during surface evaluation depends on $\Delta Vmax$. If both directions are *flat*, then the stepover sizes in both directions should be the same. Otherwise, the proportional contribution of the partial term M_2 to stepover sizes in the *u* and *v* directions will take into consideration the ratio of M_1 and M_3. As a result, we have four different cases, $M_1 > 0$ and $M_3 > 0$, $M_1 > 0$ and $M_3 = 0$, $M_1 = 0$ and $M_3 > 0$, and $M_1 = M_3 = 0$. We will discuss each of them later in this chapter.

Based on the above equations, a linear surface approximation which is different from the traditional surface approximation can be presented. This approximation is especially important for applications where the original surface definition is difficult to work with, such as in display of the surface and calculation of surface properties. It is important that a bound on the error of the approximation is known so that any analysis done on the approximation can take it into consideration. The traditional way to approximate a given NURB surface is to recursively subdivide the B-splines in *u* and *v* directions, and to perform a tolerance test. These methods have the advantages over non-adaptive methods that the approximating pieces can be placed adaptively over the surface -- that is, more surface points or meshes can be placed in areas of greater curvature, which minimizes the total number of surface points. However, in the manufacturing area, high degree surfaces are rarely used, because they are not very stable in practice. Those places with high curvature on manufacturing parts are often be broken down into small surface pieces so that relatively low degree surfaces can be used. The algorithm presented here is several times

faster than the usual adaptive algorithms. In addition, the number of surface points generated by this scheme is not much more than necessary, since in the worst case, the surface can first be split into its patches, with each patch usually having roughly similar curvature. Then each patch can be approximated independently.

Unlike Cheng's algorithm, we divide surfaces in the u and v directions separately. In other words, we have two different subdivision depths, n_u and n_v. Thus the approximated surface consists of $(n_u + 1)(n_v + 1)$ discretized surface points, or $2n_u n_v$ triangles constructed from these points. The subdivision depths n_u and n_v also give the stepover sizes, $1/n_u$ and $1/n_v$, in parametric space.

It is important to point out that n_u and n_v will be very useful for us to estimate a so-called min-max bounding box. The bounding box is defined by two points $\boldsymbol{P}_{min} = (x_{min}, y_{min}, z_{min})$ and $\boldsymbol{P}_{max} = (x_{max}, y_{max}, z_{max})$. Any surface point P satisfies $\boldsymbol{P}_{min} \leq \boldsymbol{P} \leq \boldsymbol{P}_{max}$. The bounding box will help us to eliminate those toolpath segments which will not affect the selected surface, so that most unnecessary computation for toolpath verification can be eliminated.

Here is the parallel NC verification algorithm for a given rational B-spline surface.

1. Compute $\hat{\boldsymbol{S}}(u, v)$

2. Compute the values of w_{min}, p_{max} and δ by using EQ. 4.6, EQ. 4.7 and EQ. 4.8.

3. Compute the values of M_1, M_2 and M_3 based on EQ. 4.9 - EQ. 4.12.

4. Compute subdivision depth n_u and n_v or parametric stepover size $1/n_u$ and $1/n_v$, using one of the four cases below, based on the values of M_1, M_2 and M_3. Notice that $1/n_u$ and $1/n_v$ are actually equal to $\Delta Uest$ and $\Delta Vest$ defined earlier.

As we discussed earlier in this chapter, M_1, M_2 and M_3 gave us a clue to determine the necessary stepover sizes in the u and v directions. M_1 carries the curvature information for the u direction, while M_3 has it for the v direction. Thus we can define the ratio of M_1 and M_3 as k -- *i.e.* $k = M_1/M_3$ in general, and treat each of the special cases separately.

From another view of the problem, since $1/n_u$ and $1/n_v$ have the same meaning as Δu and Δv in EQ. 4.4, we can substitute them to obtain the following equation:

$$\frac{1}{8}(\Delta u^2 M_1 + 2\Delta u \Delta v M_2 + \Delta v^2 M_3) = \frac{1}{8}\left(\frac{1}{n_u^2}M_1 + \frac{2}{n_u n_v}M_2 + \frac{1}{n_v^2}M_3\right) = \delta \quad \text{(EQ 4.13)}$$

Based on M_1, M_2 and M_3, which were found in step 3, we can now solve for n_u and n_v using one of the following four cases:

Case 1: $M_1 > 0$ and $M_3 > 0$

This is the general case. Since n_u and n_v are the necessary subdivision depths in the u and v directions, we can set the ratio of them equal to k -- *i.e.*, $k = n_u/n_v = M_1/M_3$. By substituting k into EQ. 4.13, we obtain:

$$\frac{1}{8n_u^2}(M_1 + 2kM_2 + k^2 M_3) = \delta$$

The solutions for n_u and n_v for this step are then given in EQ. 4.14

$$\begin{cases} n_u = \sqrt{\frac{1}{8\delta}(M_1 + 2kM_2 + k^2 M_3)} \\ \qquad\qquad \text{where } k = M_1/M_3 \\ n_v = \frac{n_u}{k} \end{cases} \tag{EQ 4.14}$$

Case 2: $M_1 > 0$ and $M_3 = 0$, which means that the surface is *flat* in the v direction or the v direction is linear. Thus the necessary stepover sizes in the v direction can be set to 1, i.e. $n_v = 1$. EQ. 4.15 gives the solution in this case.

$$\begin{cases} n_u = \frac{1}{8\delta}(M_2 + \sqrt{M_2^{\,2} + 8\delta M_1}) \\ n_v = 1 \end{cases} \tag{EQ 4.15}$$

Case 3: $M_1 = 0$ and $M_3 > 0$, which means that the surface is *flat* in the u direction or the u direction is linear. Similarly to Case 2, we have

$$\begin{cases} n_u = 1 \\ n_v = \frac{1}{8\delta}(M_2 + \sqrt{M_2^{\,2} + 8\delta M_3}) \end{cases} \tag{EQ 4.16}$$

Case 4: $M_1 = 0$ and $M_3 = 0$, which means that the surface is *flat* in both the u and v directions, or both the u direction and the v direction are linear. In this case, the ratio of n_u and n_v is equal to 1. Thus we have

$$n_u = n_v = \frac{1}{2}\sqrt{\frac{M_2}{\delta}} \tag{EQ 4.17}$$

5. Determine the actual stepovers in the u and v directions by

$$\begin{cases} \Delta u = min(1/n_u, \Delta Umax) = min(\Delta Uest, \Delta Umax) \\ \Delta v = min(1/n_v, \Delta Vmax) = min(\Delta Vest, \Delta Vmax) \end{cases} \quad \text{(EQ 4.18)}$$

6. Compute the total estimated number of discretized surface points for all the surfaces, assuming there are a total of m surfaces

$$Np = \sum_{i=1}^{m} \left\{ \left(\frac{1}{\Delta u_i} + 1 \right) \times \left(\frac{1}{\Delta v_i} + 1 \right) \right\} \quad \text{(EQ 4.19)}$$

7. Have the master processor distribute surface patches along with Δu and Δv to each processor for discretization. The total number of discretized surface points on each processor should be approximately N_p/p in order to maintain load balance.

8. Each processor evaluates its assigned surface patches independently.

9. For each surface patch, determine the min-max bounding box by first finding $\mathbf{P_{min}}$ and $\mathbf{P_{max}}$ from the discretized surface points. Notice this can be done during the surface evaluation process. Because our algorithm guarantee δ-closeness, the actual min-max bounding box can be easily determined by $\mathbf{P_{min}} - (\delta, \delta, \delta)$ and $\mathbf{P_{max}} + (\delta, \delta, \delta)$.

10. Eliminate unnecessary toolpath segments and perform NC verification on those surface patches in local processor memory.

11. Postprocessing, including extraction of the NC verification results from each processor and graphical display of the results.

Suppose we have p processors on our parallel machine, and we distribute the mathematical representation of a subset of the surfaces to each processor and allow it to generate approximately N_p/p surface points independently. The sets of discretized surface points on each processor are totally independent of each other. Therefore each set is placed in its processor's local memory. The toolpath data can be stored in shared memory, or each processor may make a local copy of those toolpath segments which it might possibly need, depending on the size of local memory on each processor. In this manner, fully parallel operation can be achieved.

The estimated processing time on a sequential machine, T_s, and the parallel processing time, T_p, can be expressed as:

$$T_s = T_{sinit} + T_c Np \text{ and}$$

$$T_p = \left\lceil T_{pinit} + T_c \frac{Np}{p} + T_n(p) \right\rceil$$

where T_{sinit} and T_{pinit} are the average process initiation or setup times on a sequential machine and a p-processor parallel machine, respectively. T_c is the average computation time for NC verification on each surface point. N_p is the total number of discretized part surface points. $T_n(p)$ is the communication time spent among the processors, including data transfer, shared memory access, etc. $T_n(p)$ is a function of number of processors p.

Finally, the estimated process speedup S_p of our algorithm is given in EQ. 4.20

$$S_p = \frac{T_s}{T_p} = \frac{T_{sinit} + T_c Np}{\left\lceil T_{pinit} + T_c \frac{Np}{p} + T_n(p) \right\rceil} \qquad \text{(EQ 4.20)}$$

When problem size -- *i.e.*, the total number of surface points to be verified, N_p -- tends to infinity, we can achieve the ideal case, which is, $\lim_{Np \to \infty} S_p = p$. However, when p increases without N_p dramatically increasing, $S_p \to 0$ because $T_n(p)$ also increases, mainly due to network contention.

We will present experiment results in a later chapter to demonstrate the proposed surface discretization algorithm for parallel processing.

CHAPTER V

PARALLEL NC VERIFICATION SYSTEM

Based on the previous discussion and the proposed parallel NC verification scheme, we construct the system defined in Figure 5.1. This system reads part surface data files, CL data files and APT cutter statements as inputs. It also allows users to specify the surface discretization tolerance and NC verification tolerance including *intol*, *outol* and range of interest. If a cut value is within the range specified by *intol* and *outol*, then the cut can be considered within machining tolerance. Range of interest is a user-selected parameter to provide a maximum offset around the surface, outside of which the algorithm need not calculate cut distance, and the display of those areas can be uniformly colored. The system pre-evaluates the part surface data and distributes the surface data in parametric form to each of the p processors. The initial load balance on each processor is guaranteed because each processor will conduct the NC verification process on an approximately equal number of surface points. The surface discretization is performed locally on each processor. The selection of the set of possibly relevant toolpaths is also done on each processor (note that these sets are typically neither disjoint nor identical across processors). A local copy of the toolpath set can reduce network contention and increase computational efficiency as well as system speedup, if local memory permits. Then the most time-consuming computations of the NC verification, minimum cut value calculation and surface discretization, proceed totally independently on the p processors. Verification results are

displayed graphically to depict any tolerance violations, and the data structure permits arbitrary views without redoing any verification calculations.

In this chapter, we introduce the parallel NC verification system. The four main tasks of the system are discussed first. Each of the major components of the system will be presented. The algorithm for each component is depicted in pseudo code. We then introduce the overall NC verification procedure and the necessary input and output data for the system.

5.1 Main Parallel NC Verification Tasks

There are several steps in our parallel NC verification system. First, one must be able to load surface data, cutter description data and CL data. Then surface data must be evaluated and initial load balance must be estimated before distribution of parametric surface data to each processor. On each processor node, the discretized surface point nets must be generated. Those toolpath segments which possibly affect part surfaces must be selected and discretized if necessary. Finally, maximum cut depth on each surface point must be computed to provide NC verification results. Thus, this procedure consists of four main tasks: surface pre-evaluation, object space surface discretization, toolpath processing and verification, and display of the verification results as a pseudo-coloring of the part surfaces. We now provide pseudocode to describe each process.

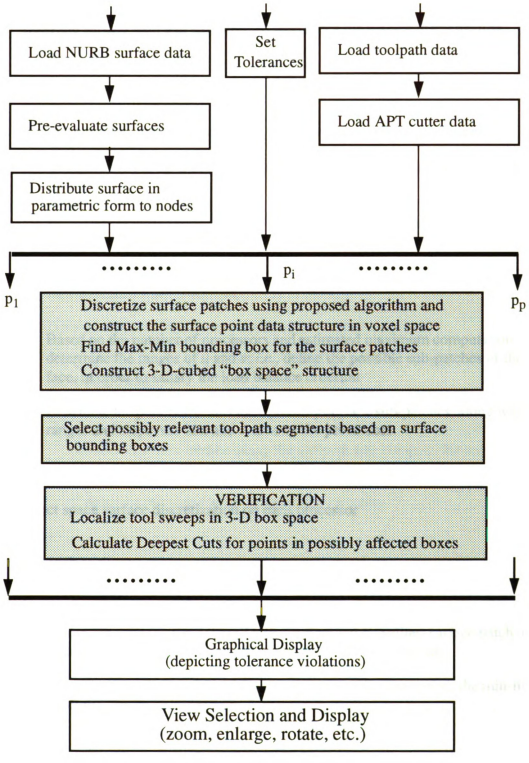


FIGURE 5.1 Overview of Parallel NC Tool Path Verification System

• Surface pre-evaluation

Load the surface data and user-defined tolerances for surface discretization

For each surface:

{

Pre-evaluate surface and compute the stepovers in u and v directions

Compute maximum number of discretized surface points possibly required

Calculate the min-max box

}

For each surface:

{

Based on the number of processors and estimated maximum computation load, determine the ranges of u and v; *i.e.*, define the possible sub-patches of the surface, in order to satisfy the load balance criterion

Distribute the parametric surfaces, possibly surface sub-patches, along with the ranges and stepovers in u and v, among the processors

}

• Object space surface discretization on each processor

For each surface:

{

Discretize each NURBS (Non-Uniform Rational B-Spline) surface patch and store surface points and normals in a POINT list data structure

Compute the min-max box of the processor's work space -- *i.e.*, the min-max of all the processor's surfaces

}

Construct a triangle list based on the POINT list

Classify each surface point into a uniform discretization of 3D-space (voxels), and establish a linked list for each voxel including all of its points.

- Toolpath Verification on each processor

  Load the cutter data and CL file

  Link all CL points to create a toolpath segment list

  For each toolpath segment:

  {

    Check whether it is possibly within the min-max box of the processor's work-space

    Calculate the parameters for the tool axis motion, implicitly defining a ruled surface

    Further discretize the toolpath segment if necessary, when the tool axis orientation changes within a segment

    Determine which voxels may contain this toolpath segment

    For every surface point within those voxels which possibly interfere with this toolpath segment

    {

      Simulate the tool motion along the toolpath and compute the cut value for the surface point

      Update the cut value in the POINT data structure

    }

  }

- Display the verification results as a pseudo-colored raster image of the part surfaces

  For each triangle in the triangle list:

  {

    Retrieve the cut values from the vertices of the triangle

    Compute the color value at each pixel inside the triangle based on the magnitudes of the three cut values

Determine the boundaries between regions of varying hues between the discretized surface points by linear interpolation of cut values. Determine intensity based on angle between sight line and surface normal

}

5.2 Command File and Command Driven Mode

In a previous NC simulation and verification system[16], the system ran in an interactive mode; that is, the user had to specify all the parameters via an X-Window-based graphical user interface (GUI) and wait for the result while the verification computation was processed. Since the amount of computation required is often large, the user might have to wait a long time for results. Many CAD/CAM systems in the market now provide a so-called batch or command-driven mode to alleviate this type of problem. While the software is running under batch mode, the user is free to use the computer (and CAD/CAM system) to do other tasks. We incorporated such a batch mode in our new NC verification system. The two immediate benefits obtained are: 1. system efficiency is improved and enforced idle waiting time is eliminated; 2. the system is more adaptable and portable. For example, there is no X-Window environment on the BBN GP1000 parallel computer, so the earlier system could not be run on that machine; the new system has run successfully under this command-driven mode.

The new parallel NC verification system can run either with or without an X-Window environment. If one has an X-Window environment, the program will run as before. However, if one does not have or does not want to run under X-Windows, the program can be run through a command file. There are some reserved keywords for command files, such as *rd_srf*, *rd_path* and *rd_tool*, for reading surface file, toolpath file and cutter defini-

tion file, respectively. There is also *srf_data* for reading surface discretization parameters, // stands for comment line, etc. The following example shows a typical verification procedure defined by a command file.

```
// read surface file, toolpath file and tool definition file
rd_srf apmes.iges
rd_path apmes.cl
rd_tool apmes_apt_cutter
// surface tessellation parameters: maximum chordal deviation and polygon length
srf_data 0.25 0.5
ld_srf
ld_tool
pre_process
// conduct verification with INTTOL, OUTTOL and range of interest
verify 0.5 0.5 1.0
```

5.3 File Loader

To process the NC verification, the system needs to know input data, such as part surface information, cutter definition and CL data, etc. The file loader is designed for this purpose.

5.3.1 Surface File Loader

The desired part surfaces used in our system consist of NURB surfaces, which are defined by a designer and typically are directly produced by a CAD package. NURBS, trimmed NURBS, and bounded NURBS may all be stored in standard IGES formats. Trim

and boundary curves may be either NURBS curves, composite curves or copious data entities in IGES format. The core NURBS surface data is the type 128 entity of IGES. This entity can exactly represent various analytical surfaces of general interest.

The IGES 128 entity includes two major parts, the directory entry and the parameter data. The directory entry provides general surface information for both the surface generating system and the surface manipulating system. The surface *form number* in this entry defines surface type, which could be NURBS surface type, ruled surface type, plane surface type, revolution surface type, etc. The parameter data entity gives the entire definition of the surface. It is in the following form:

Index	Name	Type	Description
1	K1	Integer	Upper index of first sum of S(u,v)
2	K2	Integer	Upper index of second sum of S(u,v)
3	M1	Integer	Degree of first set of blending function
4	M2	Integer	Degree of second set of blending function
5	PROP1	Integer	1 = Closed in first parametric variable direction 0 = Not closed
6	PROP2	Integer	1 = Closed in second parametric variable direction 0 = Not closed
7	PROP3	Integer	0 = rational surface 1 = polynomial surface
8	PROP4	Integer	0 = Nonperiodic in first parametric variable direction 1 = Periodic in first parametric variable direction
9	PROP5	Integer	0 = Nonperiodic in second parametric variable direction 1 = Periodic in second parametric variable direction

Now let N1 = 1+K1-M1, N2 = 1+K2-M2, A = N1+2*M1, B = N2+2*M2, C = (1+K1)*(1+K2), then

10	U(-M1)	Real	First value of first knot vector

...			
10+A	U(N1+M1)	Real	Last value of first knot vector
11+A	V(-M2)	Real	First value of the second knot vector
...			
11+A+B	V(N2+M2)	Real	Last value of the second knot vector
12+A+B	W(0,0)	Real	First weight
13+A+B	W(1,0)	Real	
...			
11+A+B+C	W(K1,K2)	Real	Last weight
12+A+B+C	X(0,0)	Real	First control point
13+A+B+C	Y(0,0)	Real	First control point
14+A+B+C	Z(0,0)	Real	First control point
15+A+B+C	X(1,0)	Real	
16+A+B+C	Y(1,0)	Real	
17+A+B+C	Z(1,0)	Real	
...			
9+A+B+4*C	X(K1,K2)	Real	Last control point
10+A+B+4*C	Y(K1,K2)	Real	Last control point
11+A+B+4*C	Z(K1,K2)	Real	Last control point
12+A+B+4*C	U(0)	Real	Starting value for first parametric direction
13+A+B+4*C	U(1)	Real	Ending value for first parametric direction
14+A+B+4*C	V(0)	Real	Starting value for second parametric direction
15+A+B+4*C	V(1)	Real	Ending value for second parametric direction

Besides the untrimmed surface entity described above, the system is able to handle rational B-spline curves, composite data, copious data, curve on a parametric surface and trimmed parametric surface data as well, to form a complete surface definition. The surface file loader reads IGES surface data files into our project data structure. Then the surface evaluator computes the parametric surfaces and trim curves as points in 3D space, and constructs the lists of surface points and triangles which constitute our internal representation of the trimmed surfaces.

The first step in loading surfaces is the reading of all the entries in the IGES-format file directory into an internal directory. From the internal directory, we determine which parameter data to read from the file and the order of reading. IGES 144 entity parameter

data are read first, as these entities control the collection of trim curves with their corresponding surfaces. As directed by the 144's and their associated 142's, an internal surface data structure is created and the definition parameter data for the surface and trim curves is read from the IGES file and loaded into IGES128, IGES126 and IGES106 data structures, which are pointed to by the surface data structure as surface definition data and trim data. Any IGES 128 entities remaining after the 144's have been loaded will be untrimmed surfaces. These untrimmed surface are loaded into the surface data structure with the pointer to trim data left *NULL*.

The internal surface data structure is defined as follows:

```
#define SURFACE struct surface
{
    SURFACE     *next;              /* pointer to the next surface structure */
    POINT       *point_list;        /* pointer to this surface point list */
    TRIANGLE    *triangle_list;     /* pointer to triangle list which display
                                       the surface */
    IGES128     *surf_data;         /* pointer to surface definition */
    B_DATA      *bounds_data;       /* pointer to surface boundary data */
    int         bounds_count;       /* number of boundary curves */
    float       corners[4][3];      /* corner points of the surface in 3D
                                       space */
    float       surface_minimum[3]; /* surface bounding box */
    float       surface_maximum[3]; /* surface bounding box */
    int         norms_checked;      /* for automatic surface normal check */
    int         steps_u, steps_v;   /* number of steps in u and v direction */
}
```

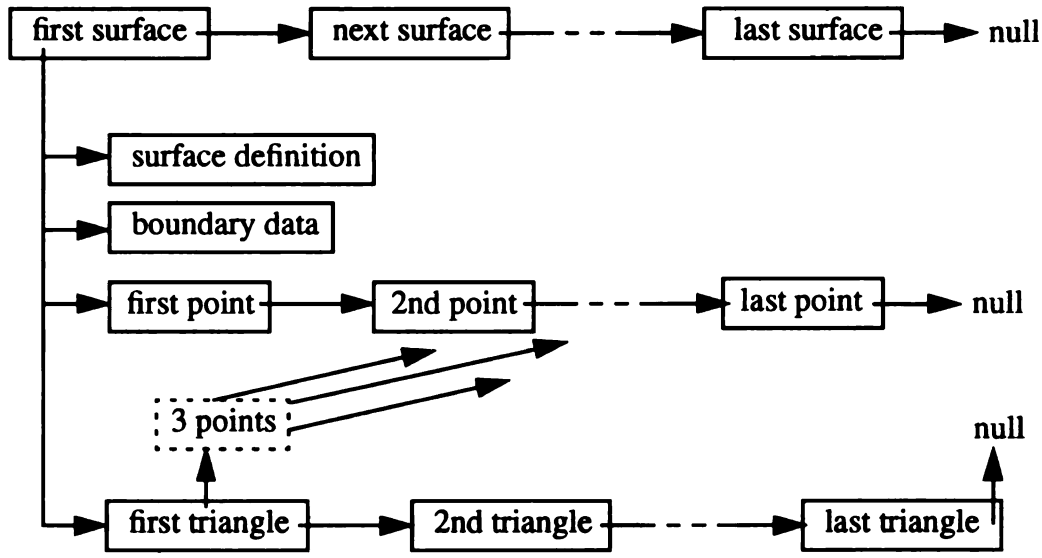


FIGURE 5.2 Surface Data Structure and Links

The surface structure list can be depicted in Figure 5.2. As shown above, at the first level, the surfaces are organized as a linked list of SURFACE data structures, each containing all the information from one IGES128, the underlying surface definition. These SURFACE structures include pointers to the definition data as well as the list of surface POINT and TRIANGLE data structures by which each surface is represented internally. As the SURFACE list is stepped through, each surface is analyzed into lists of 3D xyz points and triangles as required to meet the discretization parameters. The surface point lists of all surfaces make up the entire part to be machined, as it will be used in NC toolpath verification, while the surface triangle lists represents the part for display purposes. If a surface is trimmed, points found to be outside trim boundaries are removed from the point lists. Points are created on the trim boundaries as required for good verification output.

5.3.2 Toolpath (CL) File Loader

The toolpath is considered to define a linear motion of the tool's control point between adjacent CL (cutter location) points. In other words, the sequence of toolpath segments can be represented by a sequence of cutter locations, (x, y, z), and their orientations, $\cos\alpha,\cos\beta,\cos\gamma$ where α, β and γ are the angles of the cutter axis with the x, y and z axes, respectively, at that cutter location. $\cos\alpha,\cos\beta,\cos\gamma$ can also be written as i, j and k, the normalized cutter axis vector.

The data structure of the toolpath is also built as a linked list. It includes eight data members: a pointer to the next cutter position, the current cutter location (x, y, z), tool vector (i, j, k) and toolpath number. Because of the toolpath list structure, each toolpath segment can be defined by cutter position from start point to end point. The start point of the next toolpath segment is the end point of the previous toolpath segment, and so on. Dynamic memory allocations are used while constructing the cutter paths.

5.3.3 Tool Definition File Loader

The tool is defined in this system as a general APT cutter. Seven parameters are used, in accordance with the definition in Appendix A. The data structure for the tool contains cutter type, which can be calculated from the APT parameters, distance from the cutter control point to top of cutter, the seven APT parameters, three h values, $f - r \times \cos(a)$, $f - r \times \sin(b)$ and L that bound the three cutter regions, two precomputed coefficients of radius-dependent functions (to avoid the need to recalculate them for each tool move), and the range of interest. The three most commonly used cutter types are ball-end cutter, flat-end cutter and bull-nose cutter. In all cases except ball-end cutter, the

control point is considered to be at the tip of the cutter. The APT definitions of these cutter are as follows:

Ball-end cutter: CUTTER/d, d/2, 0, d/2, 0, 0, L

The ball-end cutter has radius d/2 and total length L. Notice that the distance from the control point to the top of the cutter is L - d/2.

Flat-end cutter: CUTTER/d, 0, d/2, 0, 0, 0, L

The flat-end cutter has radius d/2 and length L.

Bull-nose cutter: CUTTER/d, r, d/2-r, r, 0, 0, L

The bull-nose cutter has radius d/2, corner circle radius r and length L.

5.4 Spatial Subdivision

To speed up the NC verification process, we subdivide the 3D workspace into a set of cubic 3D boxes, or voxels, as shown in Figure 5.3. This makes it easier to search for the 3D area where the cutter may contact part surfaces and to obtain the surface points possibly affected by a given tool motion. The spatial subdivision method selected for use creates a set of uniform cubic 3D voxels. A voxel is defined as a rectangular solid element in 3D space. A voxel is the primary volume element in our 3D workspace. As an example, a 1 m^3 volume could be considered to contain 10^6 individual 1 cm^3 cells, or voxels.

The voxel-based 3D computational space is established by a translation and scaling transformation of the 3D workspace. The translation aims to move the minimum point

of the workspace bounding box to the origin of the computational space, while the scaling transformation, which is determined by the density of overall discretized surface points, is mainly to reduce surface point searching time during toolpath verification, as described below. The transformation from workspace to computational space moves all surface points into a positive 3D space of such voxel size that the integer parts of each surface point's coordinates can serve as the index of the voxel containing the point. In other words, after the transformation, the indices of the voxel that contains the surface point are the truncated real parts of its translated (x, y, z) coordinates. Each voxel has a linked list of all the translated surface points it contains.

The voxel size plays important roles in the toolpath verification process. If the size is too large, each possibly affected voxel may contain a large number of surface points. This implies that the verification of each toolpath must consider that large number of surface points and perform a computation for each surface point. As result, the overall verification performance will be greatly reduced. The extreme case is one voxel containing all surface points -- in this case, the spatial subdivision loses all its advantages. In the other extreme, if the voxel size is too small, most voxels will be empty. This will increase the burden for searching those voxels which contain the surface points that may be cut by the given toolpath segment. The advantage of workspace subdivision is lost in this case, as well. On the other hand, if the voxel size is appropriate, most (or at least a large number, depending on the nature of the part) voxels will contain a small number of surface points. The computational costs for each toolpath segment will be reduced and system performance will be increased. It is suggested that one choose the size of the voxel to be on the

order of the length of the median toolpath segment. Therefore, typical tool motions may move into some voxels and out of others.

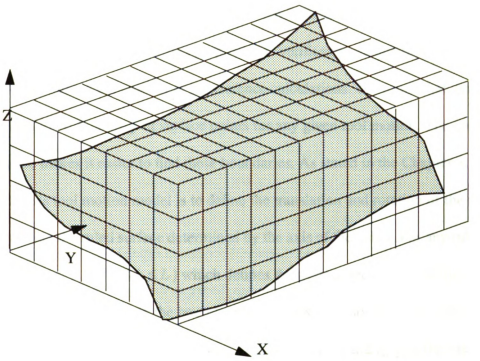


FIGURE 5.3 Spatial Subdivision of a Discretized Surface Patch

The process of the transformation from 3D workspace to 3D computational space can be depicted in the following pseudocode:

```
while (surface != NULL)
{
    Translate all surface points as well as surface corner points, surface maximum
    and surface minimum points, into the positive quadrant of the "computational
    space" coordinate system based on the part bounding box, with the minimum
    corner at the origin.

    Locate each surface point in its corresponding voxel.
}
```

A sculptured surface is typically discretized into thousands of points, depending on the desired accuracy, size, and curvature of the surface. Since a given tool motion will interfere with only a small percentage of the surface points, it is desirable to eliminate all the surface points which cannot possibly cause any interference. The voxel-based 3D computational space is designed for this purpose. NC verification calculation time is essentially proportional to the number of interference calculations. To reduce that, one needs to determine a loose boundary of surface points for any given tool motion. The computational space makes it easier to find these boundaries. As stated in the Chapter 3, the key to the multi-axis tool motion model is to define the translation and rotation of the tool in 3D space by using the ruled surface determined by the axis of the cutter. For any ruled surface $\boldsymbol{r}(t, h)$ $(0 \le t \le 1$ and $0 \le h \le L)$ which defines the corresponding tool motion, one can easily use EQ. 3.1 to evaluate the bounding box of this tool motion -- *i.e.*, determine the maximum and minimum values x_{max}, y_{max}, z_{max}, x_{min}, y_{min} and z_{min} of the ruled surface. For a 3-axis tool motion, it is straightforward to compute the bounding box by evaluating the four corner points because the ruled surface is a rectangle. For multi-axis tool motion, it is not guaranteed that the ruled surface is monotonic in any of its coordinates. The number of local extreme along any coordinate axis may be greater than one. Therefore, the evaluation of the bounding box of the ruled surface becomes much more complicated. In fact, all these problems are introduced by the increase of tool axis freedom. To simplify the problem, we shall, when necessary, subdivide each tool motion based on the angle between the tool axis at the start position and end position to guarantee the monotonicity of the surface. Hence we are still able to estimate the bounding box for each sub-motion by using the four corner points of the sub-toolpath segment (see details in the next section). Now suppose the maximum radius of the tool is R_{tool} and the range of interest is

R_{int}. R_{int} is a user-defined parameter which indicates a maximum offset around the part surfaces in which the user is concerned about verifying cut depths. In other words, the space further than R_{int} units away from all part surfaces will not be included in the NC verification process. Then any surface point **P** which might be affected by a given tool motion satisfies the following equation:

$$x_{min} - R_{tool} - R_{int} \le x \le x_{max} + R_{tool} + R_{int}$$
$$y_{min} - R_{tool} - R_{int} \le y \le y_{max} + R_{tool} + R_{int}$$
$$z_{min} - R_{tool} - R_{int} \le z \le z_{max} + R_{tool} + R_{int}$$

The pseudocode for the algorithm for searching the voxels containing points possibly affected by a given tool path segment can be expressed as follows:

Evaluate the maximum and minimum values of corner points for the ruled surface of the tool path segment

maximum coordinate of the bounding box for any effect from this tool motion = maximum coordinate of the ruled surface + R_{tool} + R_{int}

minimum coordinate of the bounding box for any effect from this tool motion = minimum coordinate of the ruled surface - R_{tool} - R_{int}

for ($i=x_{min}-R_{tool}-R_{int}$; $i<=x_{max}+R_{tool}+R_{int}$; i++)

  for ($j=y_{min}-R_{tool}-R_{int}$; $j<=y_{max}+R_{tool}+R_{int}$; j++)

    for ($k=z_{min}-R_{tool}-R_{int}$; $k<=z_{max}+R_{tool}+R_{int}$; k++)

    {

      Use i, j, k as index of voxel in which to calculate cut values of surface points.

      Set surface_pointer = first surface point in the voxel

      while (surface_pointer != NULL)

      {

```
        Compute the cut value for this surface point.
        Set surface_pointer = surface_point->neighbor
      }
    }
```

5.5 Toolpath Discretization for Multi-Axis NC Verification

As mentioned earlier, the orientation changes of the tool axis from the start point to end point of a tool motion introduce much more complexity into multi-axis NC verification, as well as into multi-axis toolpath generation. The ruled surface defined by the tool motion can become much more complex. In 3-axis point-to-point tool motions, the number of minimum distances between the ruled surface and a given part surface point is exactly one, because tool axis does not change, so the ruled surface is planar. This is no longer true in 4- or 5-axis tool motion. To solve this problem, an algorithm for dynamic toolpath discretization is implemented using a user-specified angle tolerance. The number of additional toolpath segments to be produced is determined by the angle, θ, between the tool orientations at the start point and end point and by the maximum angle tolerance. θ can be decided easily using the cross product of the orientations of the tool axis at the starting and end points of the tool motion. The procedure for toolpath discretization is depicted in Figure 5.4.

$$\theta = \text{acos}(N(0) \bullet N(1))$$

```
if (θ < angle tolerance)
  number of segments = 1
else
  number of segments = ⌈θ/(angle tolerance)⌉
```

```
store current toolpath data
for (i = 0; i <= number of segments; i++)
  {
    evaluate cut values of surface points based on this sub-toolpath segment
  }
restore toopath data
```

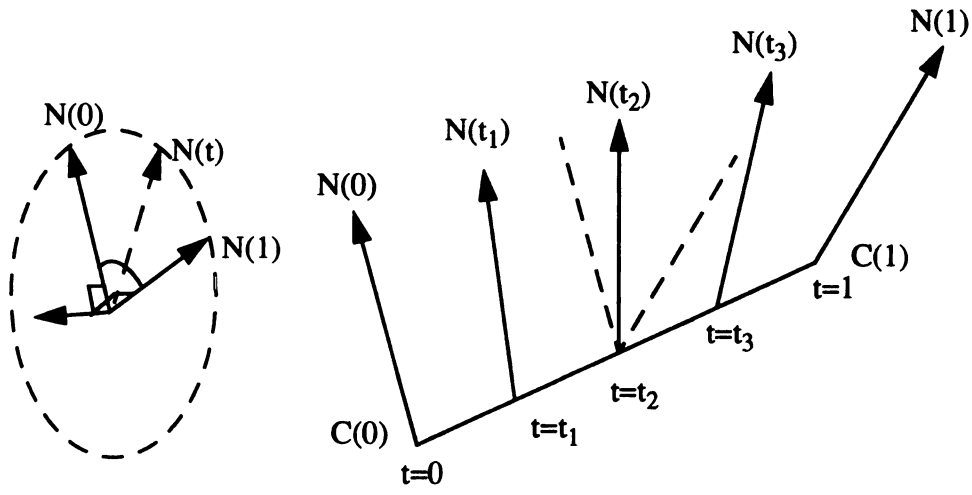


FIGURE 5.4 Tool Path Discretization

As we described in Chapter 3, given a surface point P_i and a time instant t, the closest ruled surface point can be determined by the tool control point C(t), tool vector N(t) and equation 3.2 $h_i(t) = [x_i - x(t)]N_x(t) + [y_i - y(t)]N_y(t) + [z_i - z(t)]N_z(t)$. As shown in Figure 5.4, C(t) = C(0) + [C(1)-C(0)]t. N(t) is linearly interpolated along a great circle. It is a function of t as well as θ. To derive the expression for N(t) in terms of θ and t, one needs to solve the following three equations:

$$N_0 \bullet N = |N_0||N|\cos(\theta t) = \cos(\theta t)$$

$$N \bullet N_1 = |N||N_1|\cos(\theta(1-t)) = \cos(\theta(1-t))$$

$$N \bullet (N_0 \times N_1) = 0$$

where $N_0 = N(0)$ and $N_1 = N(1)$

The above equations can be rewritten as

$$n_{0x}n_x + n_{0y}n_y + n_{0z}n_z = \cos(\theta t)$$
$$n_{1x}n_x + n_{1y}n_y + n_{1z}n_z = \cos(\theta(1-t))$$
$$r_x n_x + r_y n_y + r_z n_z = 0$$

where $r_x = n_{0y}n_{1z} - n_{0z}n_{1y}$, $r_y = n_{0z}n_{1x} - n_{0x}n_{1z}$ and $r_z = n_{0x}n_{1y} - n_{0y}n_{1x}$.

The final solution for N(t) can be expressed as

$$N_x(t) = \frac{(n_{1y}r_z - n_{1z}r_y)\cos(\theta t) + (n_{0z}r_y - n_{0y}r_z)\cos(\theta(1-t))}{|N_0 \times N_1|^2}$$
$$N_y(t) = \frac{(n_{1z}r_x - n_{1x}r_z)\cos(\theta t) + (n_{0x}r_z - n_{0z}r_x)\cos(\theta(1-t))}{|N_0 \times N_1|^2}$$
$$N_z(t) = \frac{(n_{1x}r_y - n_{1y}r_x)\cos(\theta t) + (n_{0y}r_x - n_{0x}r_y)\cos(\theta(1-t))}{|N_0 \times N_1|^2}$$

By substituting C(t) and N(t) into Equations 3.2 and 3.1, we can determine the closest ruled surface point r(t,h) = C(t) + hN(t).

5.6 NC Verification with Inward and Outward Tolerances

The algorithm presented earlier allows us to quickly evaluate the minimum distance along each surface normal vector to the boundary of the tool motion. Once all of the distances have been evaluated, our system is able to display the color coded verification results from arbitrary viewpoints based on user-specified inward and outward tolerances, as long as both values are less than the range of interest.

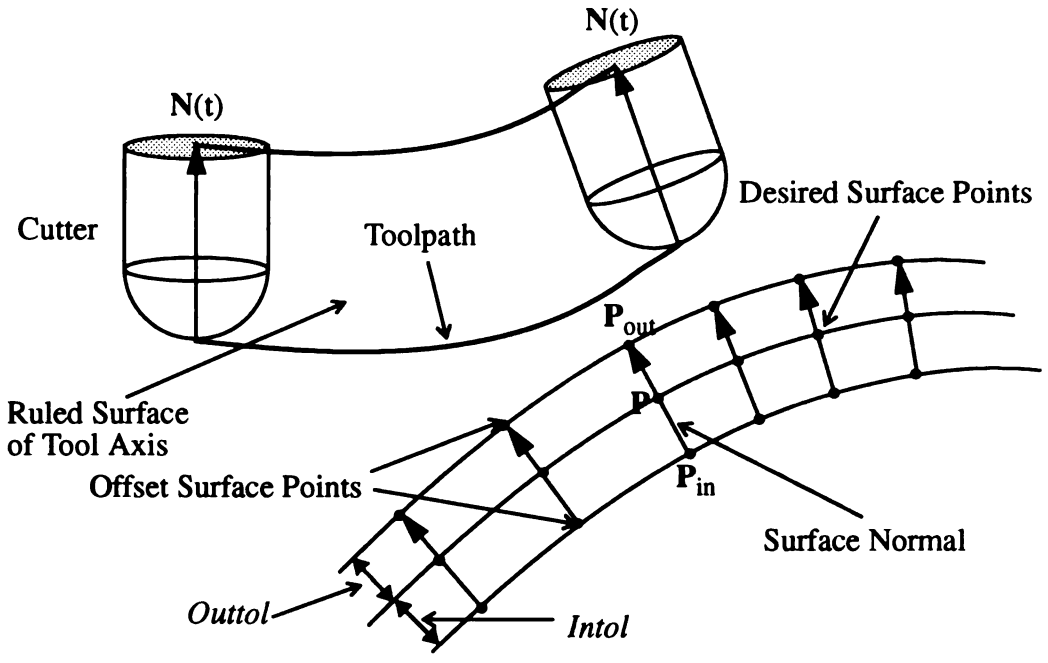


FIGURE 5.5 Offset Point P_{out} and P_{in} for Tolerance Specification

To compare the final machined part surfaces with the desired (nominal) part surfaces, one normally use the tolerances *INTOL* and *OUTTOL* defined by APT. As long as the error of the machined part surface is between *INTOL* and *OUTTOL*, the machined part surface can be considered to meet the tolerance specification. Otherwise, it violates the tolerance specification. Color coded results help the user identify the areas within tolerance and out of tolerance. Shades of green represent the areas cut within tolerance. Gouges deeper than *Intol* vary from red to yellow for distances from *Intol* to *-Rint* (a user-specified maximum range of interest). Undercuts greater than *Outtol* are shown in hues from dark blue to light blue for distances from *Outtol* to *Rint*. For intelligibility of the output, intensities of all hues are varied according to the angle between the eye point and the part surface normal.

To reduce the cost of the minimum distance calculation, two discretized offset surfaces are created from each part surface. Rather than solving for the equations of the offset surfaces, the normals of the discretized part surface points are simply used to generate $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$ offset points for each surface point in the data structure. As shown in Figure 5.5, $\mathbf{P}_{in}$ is offset along the negative of the normal vector by Intol, and $\mathbf{P}_{out}$ is offset along the positive normal vector by Outtol. By definition, these offset points are on the corresponding offset surfaces[31]. To detect whether or not the cutter cuts the part surface within the tolerance limits, the signed distances from the corresponding tool envelope to $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$ for each surface point must be calculated using EQ. 3.3. If $\mathbf{P}_{out}$ is inside the envelope of the tool motion and $\mathbf{P}_{in}$ is not, as shown in Figure 5.6 (a) and (b), then the corresponding surface point is within the tolerance specification. This is equivalent to saying that the cut is within tolerance at the given surface point if the depth of cut is anywhere between $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$, even though the level of deepest cut is only approximated.

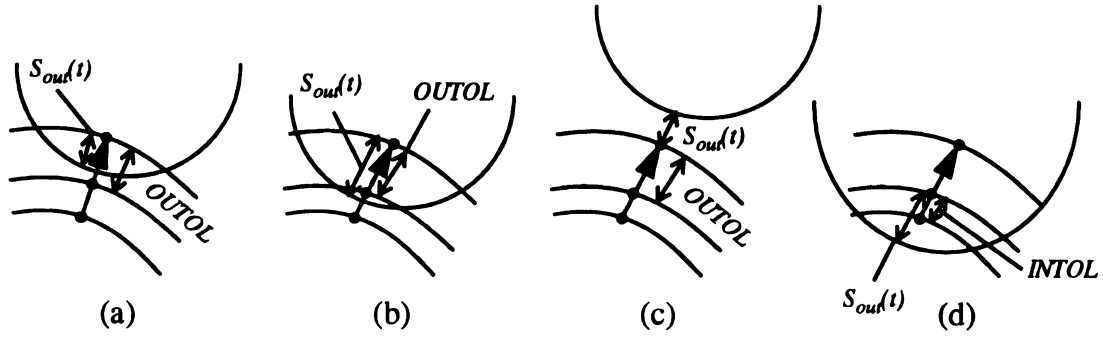


FIGURE 5.6 Relationship Between Tool Envelope and $\mathbf{P}_{out}$ and $\mathbf{P}_{in}$

The main objective of NC verification is to provide verification results to allow subsequent adjustment of the toolpath to eliminate undercut or overcut (gouge) regions.

Therefore, it is not only important to identify those regions where the tolerance specification is satisfied, but also necessary to estimate how deep the gouge is or how great the undercut is for any surface point outside the tolerance range. The approximated distance between a surface point and the ruled surface of the tool motion can be evaluated as follows.

First, evaluate the approximate cut values $S_{in}(t)$ and $S_{out}(t)$ for the $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$ offset points, respectively. The cut values for $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$ can be easily derived from EQ. 3.3, which states the condition under which the part surface point is inside or on the envelope of the tool motion at time instant *t*. As shown in Figure 3.5, suppose C(t) = (x, y, z) represents the cutter's control point and $\mathbf{P}_0$ represents the closest point on the cutter axis ruled surface to a given part surface point **P**. Since the distance between **P** and $\mathbf{P}_0$ can be determined as $|\boldsymbol{P} - \boldsymbol{P}_o|$, the distance from **P** to the cutter boundary is then equal to $|\boldsymbol{P} - \boldsymbol{P}_o| - R(h_i(t))$. If we define this distance as the cut value of **P**, we can obtain its equation, applying EQ. 3.2, as:

$$S(t) = \sqrt{[P_x - x(t)]^2 + [P_y - y(t)]^2 + [P_z - z(t)]^2 - h_i^2(t)} - R(h_i(t))$$

To calculate the cut value for $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$, we can simply substitute $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$ into the above equation.

$$S_{in}(t) = \sqrt{[x_{in} - x(t)]^2 + [y_{in} - y(t)]^2 + [z_{in} - z(t)]^2 - h_i^2(t)} - R(h_i(t))$$

$$S_{out}(t) = \sqrt{[x_{out} - x(t)]^2 + [y_{out} - y(t)]^2 + [z_{out} - z(t)]^2 - h_i^2(t)} - R(h_i(t))$$

It should be pointed out that it is not actually necessary to evaluate the cut value S(t) once we obtain $S_{in}(t)$ and $S_{out}(t)$. S(t) can be approximated from $S_{in}(t)$ and $S_{out}(t)$.

If $S_{in}(t)$ and $S_{out}(t)$ are both positive, which means both $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$ do not interfere with the envelope of the tool motion, then the approximate cut value is $S(t) = S_{out}(t) +$ *OUTOL*, as shown in Figure 5.6 (c).

If $S_{in}(t)$ is positive and $S_{out}(t)$ is negative, which means $\mathbf{P}_{in}$ does not interfere with the envelope of the tool motion and $\mathbf{P}_{out}$ does, as shown in Figure 5.6 (a) and (b), then the approximate cut value is still $S(t) = S_{out}(t) +$ *OUTOL*. Notice $S_{out}(t)$ is negative now.

If $S_{in}(t)$ and $S_{out}(t)$ are both negative, which means both $\mathbf{P}_{in}$ and $\mathbf{P}_{out}$ interfere with the envelope of the tool motion, as shown in Figure 5.6 (d), then the approximate cut value is $S(t) = S_{in}(t) -$ *INTOL*.

If $S_{in}(t)$ is negative, but $S_{out}(t)$ is positive, then we need to evaluate the surface point itself, since the surface normal and the tool axis are pointed in radically different directions. If the cut value of this surface point is also negative, then the approximate cut value is S(t). Otherwise, it means that the cutter may be approaching the surface from behind the outward-directed surface, and this case can be considered either as an error -- say, a deep gouge -- or as not cutting the surface point from its normal direction.

CHAPTER VI

PARALLEL ARCHITECTURE AND IMPLEMENTATION

6.1 Introduction

Many important problems in the areas of simulation, modeling, and manufacturing require tremendous amounts of computational power for more accurate solution. Although computing technology has advanced dramatically, many of the so-called grand challenge problems, including real-time simulation, prediction of weather, determination of molecular, atomic and nuclear structure, *etc.*, are still challenging and await new tools to provide precise solutions, because even the most powerful and advanced computers do not provide enough computational power. In order to address such problems, the goal has been to obtain computer systems capable of computing at teraflops (10^{12} floating point operations per second) rates. However, due to technological limitations, it has not been possible to boost the performance of single processors to that rate. Parallel computers are not bound by the speed of an individual circuit, and are opening the door to teraflops computing performance to meet the increasing demand for computational power. Parallel computers with hundreds or thousands of processors have proven to be the most promising technology to achieve such dramatic increases in computational power.

This chapter focuses on our parallel NC simulation and verification environment and implementation. It has two parts. The first provides the background on the parallel

computing environment and data parallel algorithms. In particular, the system configuration of the model BBN GP1000 machine and its characteristic regarding software development are introduced.

The second part discusses issues and methods related to implementation of parallel NC simulation and verification on the above system.

6.2 Background Information on Parallel Processing

The concept of parallel computation has been studied since the early days of computer development. But only recently has parallelism become an attractive and viable approach to the attainment of very high computational speeds, because of the decline in cost and size of computer components and the increasingly expensive gains in developing ever faster circuitry.

Parallelism in computation takes many forms with regard to hardware configuration and software design. The spectrum of parallel computers runs from systems with dual or quadruple processors to those utilizing hundreds or even thousands of processors. Each individual processor may be programmed to perform identical tasks or completely different tasks from those of the other processors. Data communication among processors can be through a memory common to all processors, or by means of networks interconnecting the processors in various fashions.

6.2.1 Classification of Parallel Computers

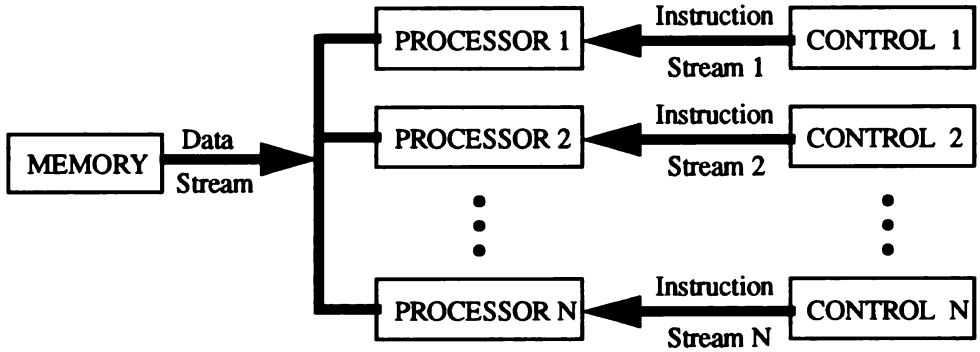
A single processor operates by executing a stream of instructions on a stream of data. It follows a single thread of control to fetch, decode, and execute a sequence of

instructions. Multiple processors in the same computer system may execute different streams of instructions on the same steam of data, the same steam of instructions on different steams of data, or different streams of instructions on different steams of data.[28][35] Depending on their data and instructions streams, machines are classified into four categories.

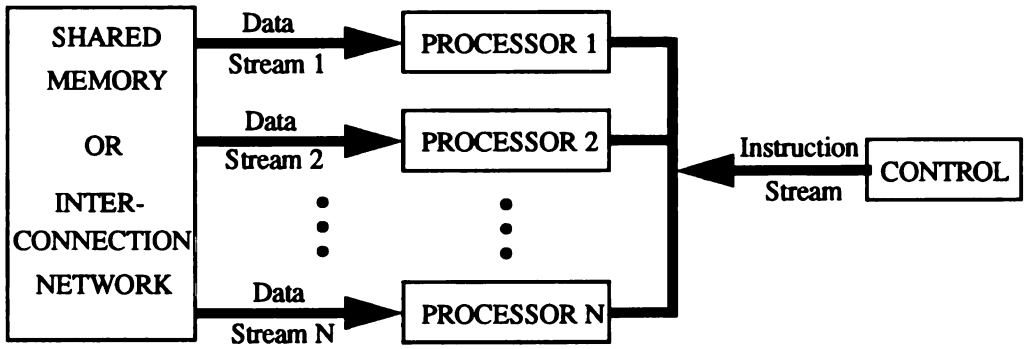
- SISD: Single Instructions Single Data
- SIMD: Single Instruction Multiple Data
- MISD: Multiple Instruction Single Data
- MIMD: Multiple Instruction Multiple Data

SISD computers are the traditional uniprocessor systems. The performance of SISD computers is enhanced by techniques that try to exploit more instruction-level parallelism.

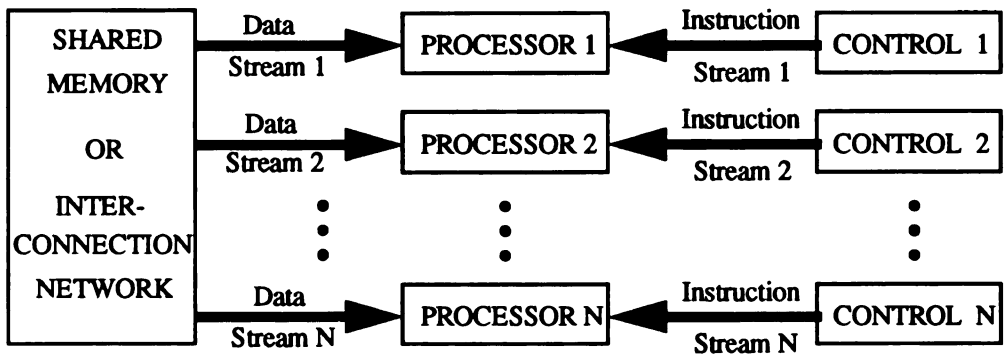
In MISD computers, as shown in Figure 6.1 (a), multiple computing functional units are organized in cascade to process a stream of data. It is a type of pipelined computing, but it is different from vector processors. The pipeline in MISD computers is visible to the programmer, while in the vector processor, a single instruction is applied *concurrently* across many data items. Because of the potentially large variation in execution time across the various computing functional units implied by the MISD organization, the overall execution rate of the MISD pipeline is determined by the stage with the poorest performance. Thus it is rare to see real MISD computers.



(a) MISD Computer



(b) SIMD Computer



(c) MIMD Computer

FIGURE 6.1 Three Classes of Parallel Computers

SIMD computers, as shown in Figure 6.1 (b), include array processors, pipelined array processors, and associative processors. All processors perform the same instruction,

which is broadcast from a single control unit. Through enable/disable or masking capabilities of individual processors, a number of applications can benefit from SIMD computers. Examples of SIMD computers include Thinking Machine's CM-2 and earlier Cray vector machines.

As shown in Figure 6.1 (c), MIMD computers can perform arbitrary operations -- *i.e.*, multiple instructions -- on different data at the same time. MIMD computers consist of a number of uniprocessors which are connected through a network. Depending on how processors and memories are connected, MIMD computers can be further classified. A popular approach tries to distinguish the systems in terms of the coupling among processors. A *tightly coupled system* usually refers to a multiprocessor system in which processors are connected to a set of memories through a switch and processors can access any location in the memories. A *loosely coupled system* consists of a number of processors, each with its own local memory, and processors communicate through explicit message passing.

Ni[28] has depicted a six-layer architecture to characterize parallel computer systems. The six layers, from bottom to top, include application, programming environment, language, communication model, address space and PMS (Processor, Memory, Switch) organization. Among these six layers, the first two are machine independent, while the last two are machine dependent. The address space of a processor depends on the memory organization, which is machine dependent. PMS organization, the bottom hardware layer, defines the configuration of processors, memories and interconnection switches, which differs from machine to machine. There are two fundamental clusters, as shown in Figure 6.2. In a Global Memory Cluster (GMC), the various processors and memory modules are

connected by a switch. In a GMC, all memory modules are equally accessible to all processors. In a Local Memory Cluster (LMC), each memory is associated with a particular processor, so that the processor and memory form a unit called a *node*. All nodes are interconnected by a switch. BBN's Butterfly machine and Thinking Machine's CM-5 are the examples of LMC. A mixed Memory Cluster can be formed as a combination of the two cluster models above.

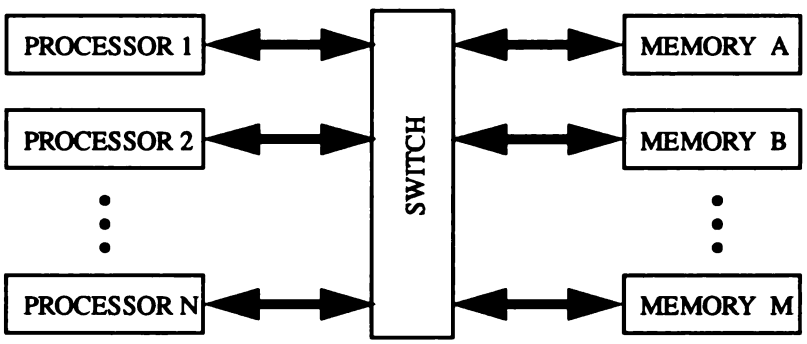
6.2.2 Data Communication and Memory Consistency

Data communication among multiple processors can be achieved through a memory common to all processors -- *i.e.*, shared memory. When processor *i* sends a piece of data to processor *j* through the shared memory, processor *i* must first write the piece of data in the memory at a location known to processor *j*, then processor *j* will read that piece of data from the location. Since any location in the memory is common to those processors sharing the memory, it is possible at any instant that more than one processor is at some stage of accessing the same memory location, and one or more of them is trying to alter the contents of the location. This is the so-called memory inconsistency problem, in which different processors may have different views of the same memory location. There are four general models for accessing shared memory.

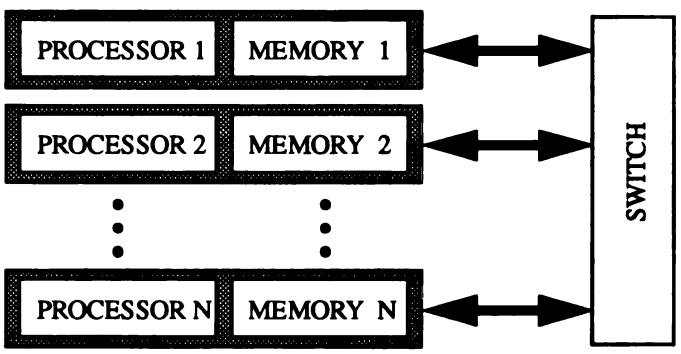
- Exclusive-Read, Exclusive-Write (EREW): No more than one processor is allowed to access the same memory location at any time.

- Concurrent-Read, Exclusive-Write (CREW): No more than one processor is allowed to write to the same memory location at any time, but there is no restriction for read access.

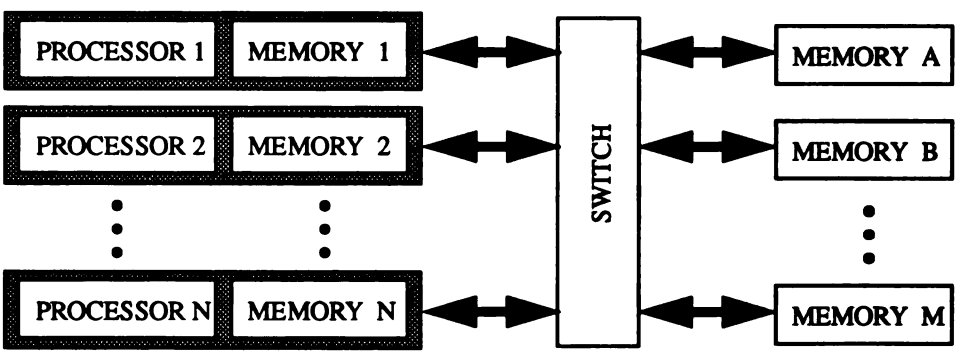
• Exclusive-Read, Concurrent-Write (ERCW): Multiple processors are allowed to write to the same memory location simultaneously, but read access remains exclusive.



(a) Global Memory CLuster (GMC)



(b) Local Memory CLuster (LMC)



(c) Mixed Memory Cluster

FIGURE 6.2 Three Parallel Computer PMS Configurations

• Concurrent-Read, Concurrent-Write (CRCW): There is no restriction for either read or write access.

The above models provide the framework for defining more subtle communication protocols to resolve memory conflicts in various computing environments.

6.2.3 Parallel Programming Models

A parallel program has two aspects: computation and communication. In computation, a processor fetches instructions and required data to perform a task. The processor may stall due to hardware contention, such as memory conflict or network contention. In communication, processes communicate or coordinate in order to preserve the required execution order. Communication overhead has a significant impact on the performance of parallel programs.

There are two important types of parallelism, *control parallelism* and *data parallelism*, in designing parallel algorithms and developing parallel programs. *Control parallelism* allows two or more operations to be performed simultaneously, while the same operation is performed on many data elements by multiple processors simultaneously in *data parallelism*. Four popular parallel computation models -- the asynchronous model, SPMD model, master/slave model and systolic model -- are frequently used.

The asynchronous model is the most general parallel computation model. There are no constraints on the interactions of the processes. Each process may run its own program and may communicate with any other processes, depending on the application. The Single Program Multiple Data (SPMD) model runs the same program code on all processors, with each processor working on its own portion of the data. The SPMD model

exploits data parallelism and is well suited for a multiprocessor with a large number of processors. Our parallel NC simulation and verification system is mainly based on the SPMD model. In the master/slave model, a dedicated master processor collects global information from slave processors or instructs slave processors to perform some dedicated function. We apply this model at the first stage of our parallel NC simulation and verification program, when the master pre-evaluates the surface data and sends the data in parametric form to each of the "slave" nodes to perform surface discretization, toolpath simulation and verification. At the second stage, the SPMD model is applied, in which each node works on its own task independently. The systolic model is another type of parallel computing model, in which each processor operates on input data from some neighboring processors and produces output data for some neighboring processors. It allows overlapped computation and communication, and has been very useful in applications such as signal processing, image processing, neural nets, etc.

6.3 BBN GP1000 and Mach 1000 System

Due to limited availability of resources, the parallel NC verification and simulation system has to date been developed and implemented only on a BBN GP1000 machine. The implementation of the system in a distributed environment through CORBA is under study. This section will focus on the BBN GP 1000 implementation.

6.3.1 BBN GP1000 Overview

The Butterfly GP1000 parallel machine is a powerful, modular computer system which consists of multiple processors and memory models connected by a high-performance internal network called the Butterfly switch. Each processor, along with an associ-

ated memory module, occupies one circuit card called a processor node (BPNE). All BPNEs in a GP1000 machine are identical; all connect to the Butterfly switch in the same way and can work together interchangeably to run an application program. This tightly-coupled processor architecture allows for efficient and extensive interprocessor communications. It gives each processor equal access to the global memory. GP1000 is a MIMD machine in which each processor can execute an independent program on local or shared memory data. It can be configured with up to 256 processor nodes.

Collectively, the memory modules of all the processor nodes form the shared memory of the machine. Although each memory module is local to one particular BPNE, any processor can access the local memory of any other processor by using the Butterfly switch to make remote memory references. Typical memory reference instructions that access local memory take about one microsecond, while accessing remote memory takes about five microseconds -- shared memory access is five times slower than local memory access. Whether a memory location is remote or local memory is determined by its address. If the high-order address bits match the processor number, the access is local and can be made without involving the switch. If these address bits differ from the processor number, the access is remote and must be made through the switch.

The distributed, shared memory architecture of the GP1000, together with the firmware and software of the Mach 1000 operation system, provides a program execution environment in which tasks can be distributed among processors regardless of where the task data are located. Although the GP 1000 is a MIMD machine, it can be programmed in several different ways -- for example, it is primarily configured to perform as a SIMD machine in our parallel NC simulation and verification system. It can be also used as a

pool of interchangeable computing resources that are allocated to tasks dynamically. Inter-processor communication can occur through shared memory, with one processor writing data for another processor to read, or through message passing. The Mach 1000 operating system and its associated set of programming languages, such as 'C' and Fortran, tools, and utilities, supports various styles of parallel processing.

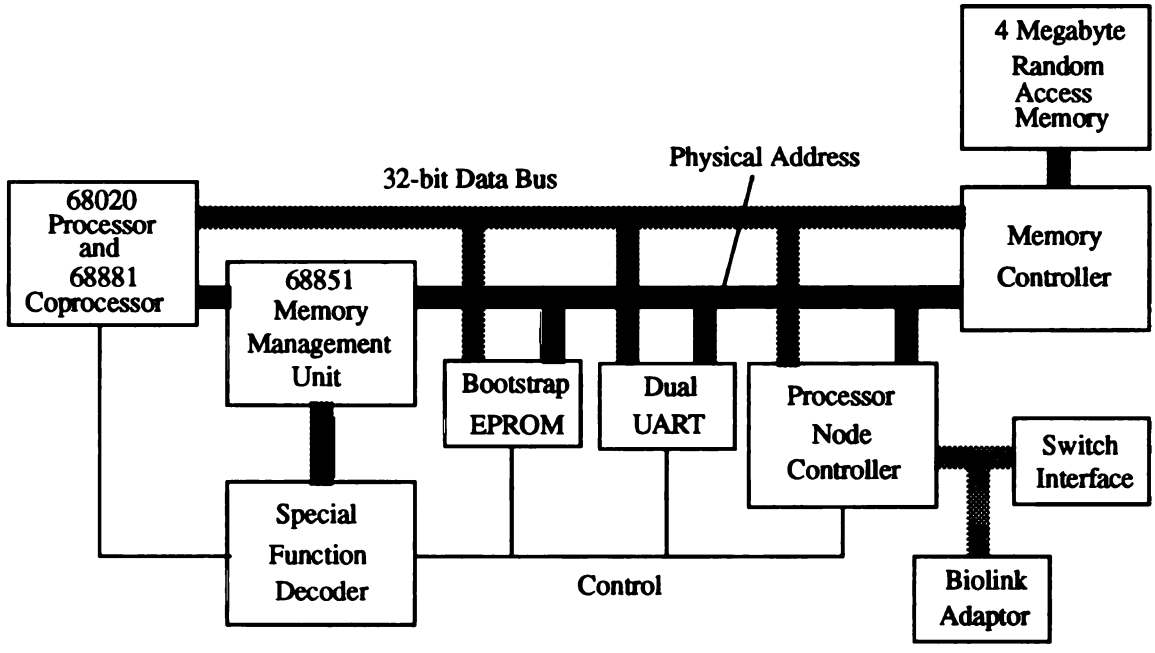


FIGURE 6.3 Block Diagram of Processor Node

The heart of the GP1000 is the identical nodes, called BPNEs. Each processor node contains the following components: an MC68020 microprocessor with MC68881 floating point coprocessor and MC68851 paged memory management unit; 4 megabytes of dynamic random access memory (DRAM); 128K programmable read only memory (EPROM); address decoding logic; processor node controller (PNC); DUART that drives the serial lines to the console terminal; I/O bus adapter; interface to the Butterfly switch;

switch power supply. Figure 6.3 is a block diagram of a processor node. The functionality of each block in Figure 6.3 is addressed in detail in Chapter 2 of [37].

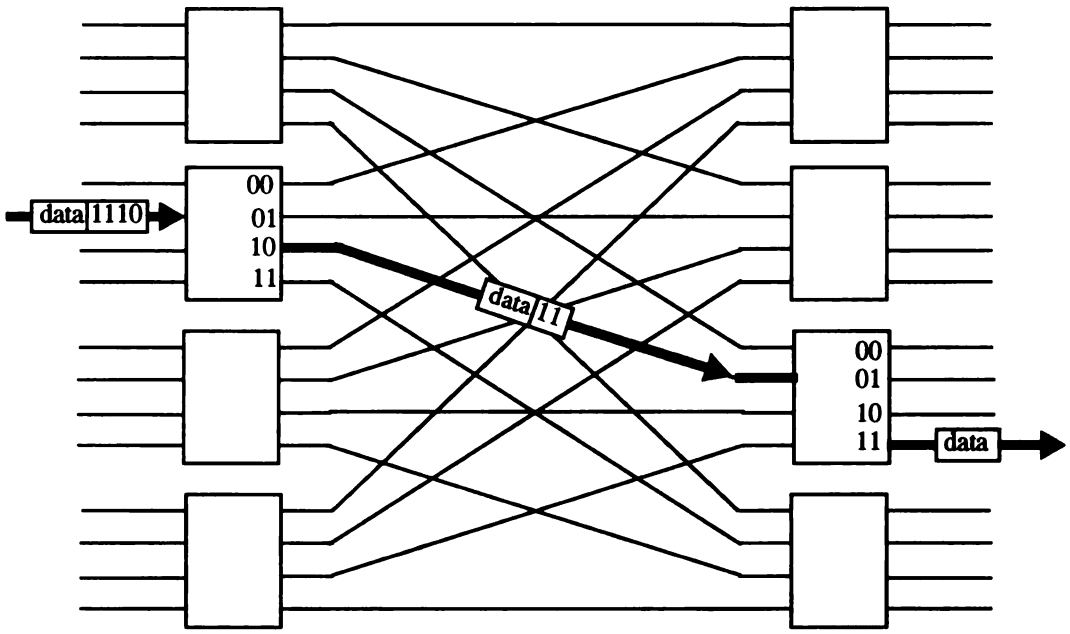


FIGURE 6.4 Eight-Node Switch and Packet Move

Interprocessor communication is implemented through butterfly switches. A Butterfly switch is a collection of switching nodes, configured as a serial decision network, which interconnects processor nodes and gives each processor equal access to the shared global memory. It supports reading and writing memory on remote processor nodes, block transfer of memory data between processor node memories, special atomic functions of the operating system, and processor node reset. Each Butterfly switching node is a 4-input, 4-output switching element chip. Eight of these chips are logically arranged in two columns to produce a 16-input, 16 output switch module as shown in Figure 6.4. At least one path through the switch module connects each processor node to every other processor

node. Switch operation is similar to that of a packet switching network. For example, if node 5 wants to send a message to node 14, node 5 must build a packet containing the 4-bit address of node 14, i.e. 1110, followed by the message data, which it then sends to the switch. The first switching node strips the two least significant address bits, i.e. 10, from the packet and uses these two bits to route the remainder of the packet out of its output port 10. The next switching node strips off the next two address bits, i.e. 11, from the packet and sends the remainder of the data to its output port 11. Port 11 of the second switch actually connects to node 14. The structure of the switch network ensures that packets with binary address 1110 will always be routed, with the same number of steps, to node 14, regardless of which processor node sent them.

6.3.2 Mach 1000 and Uniform System Approach

The GP 1000 hardware and Mach 1000 operating system[38] are a foundation on which a variety of parallel application software structures may be built. Based on this foundation and wide range of parallel software development experiences, BBN proposed a parallel programming style or method called the *uniform system approach.* This approach has proved to be particularly effective for applications containing a few frequently repeated tasks. The "uniform system" provides a library of functions or subroutines which can be used in either 'C' or Fortran.

There are two key considerations during parallel programming on GP1000: memory management and processor management. The goal of memory management is to use the full memory bandwidth of the machine -- *i.e.*, to attempt to prevent many processors from accessing a single memory module while other memory modules are idle. The goal

of processor management is to utilize the full processor bandwidth of the machine -- *i.e.*, to keep all the processors equally busy or the keep the load balance among processors.

The Butterfly switch in the GP1000 provides high memory bandwidth for all the processors. The uniform system follows two principles for memory management. First, it tries to use a single large address space shared by all processes to simplify programming. Two or more processes can share a single large block of virtual memory if they have either a parent-child or sibling relationship. This frees the application programmer from the need to manipulate memory maps. Data that two or more processors must share is allocated without regard to which processors will use it. The stack and local variables, as well as code, are not fetched across the Butterfly switch. Second, the uniform system scatters application data across all memories of the machine to reduce possible memory contention. The large shared memory is composed of the memories of all the nodes collectively. If all the shared data used by an application happened to be located in a single physical memory, the memory contention -- that is, multiple processors accessing the same memory location -- would force the processors to proceed serially, thereby reducing the program performance. Since the aggregate memory bandwidth of the GP1000 is very large, memory contention can be reduced by scattering application data across the physical memories of the machine. When many processors access data that has been scattered, their references tend to be distributed across the memories and can make use of the full memory bandwidth.

The most novel aspect of programming on the GP1000 is processor management. It falls naturally into two parts: identification of the parallel structure inherent in the application to be parallelized, and controlling the processors to achieve the parallelism identi-

fied. In many applications, the parallel structure is both obvious and rich, such as in matrix computation. In others, the structure is less clear and requires more study, such as in the simulation and verification of robotic spraying or NC machining. Discovering and implementing parallelism is done application by application. Once the programmer has decided what processing will occur in parallel, he or she must then control the GP1000 to make this happen. The Mach 1000 kernel provides a rich collection of relatively low-level operations for starting processes on various processors and for communicating among them. The Uniform System provides a higher level abstraction for managing the processors. It treats processors as a group of identical workers, each able to do any task. To use the Uniform System, a programmer must divide his or her application into two parts -- a set of functions that perform various application tasks and one or more functions called *task generators* that identify the *next* task for execution. Notice that the *task generator* is normally embedded in the control structure in a serial program. The *task generator* can also be considered to be the master in a master/slave parallel programming model.

To understand the how the task generator works, we can think of the generator concept in terms of three procedures -- a generator activator procedure, a worker procedure and a task generation procedure -- and a task descriptor data structure. Suppose we have a "master" processor which calls the generator activator procedure to start the application software. The generator activator procedure first builds a task descriptor data structure that specifies the tasks to be generated in terms of the worker procedure, the data, and the task generation procedure. It then "activates" the generator by making the task descriptor available to other processors. The "master" processor, along with other available processors, then uses the task descriptor and the task generation procedure to make repeated calls on

the worker procedure, specifying subsets of the data to work upon. Each call of the worker procedure is a task. When the last task is done, the "master" processor continues execution of its program, while the other processors that worked on the tasks look for other work. To help understand the above concept, let's consider a matrix multiplication $A_{m \times k} \times B_{k \times n} = C_{m \times n}$ as an example. For each *i*th row of **A** and *j*th column of **B**, we can have a 'C' function, say DotProduct(i, j), which computes the dot product and then stores the results the $(i,j)$the element of **C**. In this example, DotProduct(i, j) is the worker procedure, and the operand and result matrices are the data. DotProduct(i,j) is called once for each combination of row and column index; these indices are stored in the task descriptor and are incremented atomically each time the task generation procedure is executed by a processor.

The Uniform System supports two kinds of task generators. Synchronous generators return to the caller after all of the generated tasks have been processed. Furthermore, the processor that calls a synchronous generator always works on the tasks that are generated. Asynchronous generators return to the caller as soon as the generator has been activated. This enables the calling process to do other work. The Uniform System matches available processors to the generated tasks and keeps track of active task generators. Whenever a processor has nothing to do, it obtains a task using the task generation procedure for one of the active generators. When a Uniform System program begins execution, all the processors, except the one used to start the program, are idle. As long as there are active generators with tasks to be done, there are no idle processors.

6.4 Implementation of Parallel NC Verification

As we mentioned earlier, the key issue in implementing applications in parallel is to explore the embedded parallel structure. The structure in NC simulation and verification is not obvious and straight forward. In order to develop the parallel NC simulation and verification system, we followed general parallel programming guidelines, starting with the best existing algorithms and system, namely, the early version of an NC simulation and verification system developed at the Case Center[16]. The block diagram for the overall system was studied and analyzed. An attempt was then made to do the same number and kinds of steps as those in the existing system, studying the steps and re-ordering them in order to achieve parallelism. Opportunities for parallel execution were sought at all levels and in all sizes. The final solution was developed after numerous exploratory studies.

Although the Uniform System provides both static and dynamic approaches for determining the desirable number of concurrent operations to have at any stage in the processing, we decided to use the dynamic approach, to attempt to maintain the load balance. That is, tasks are allocated dynamically to processors. As a processor finishes a task, it is assigned the next task ready for execution. This approach minimizes adverse end effects by having many more tasks than processors. The end effects refer to the processor idle time that occurs toward the end of computation when some processors have finished and others are still working. In this approach, some idle processor time occurs at the end of execution, but it is generally small relative to the total program execution time.

During implementation, we concentrated on the two key considerations -- memory management and processor management. We applied the master/slave parallel program-

ming model in our system. The master processor works on the program module which cannot be parallelized and assigns those jobs that can be parallelized to all processors available at a particular time. In fact, all the NC verification tasks are assigned to worker procedures which can be processed in parallel.

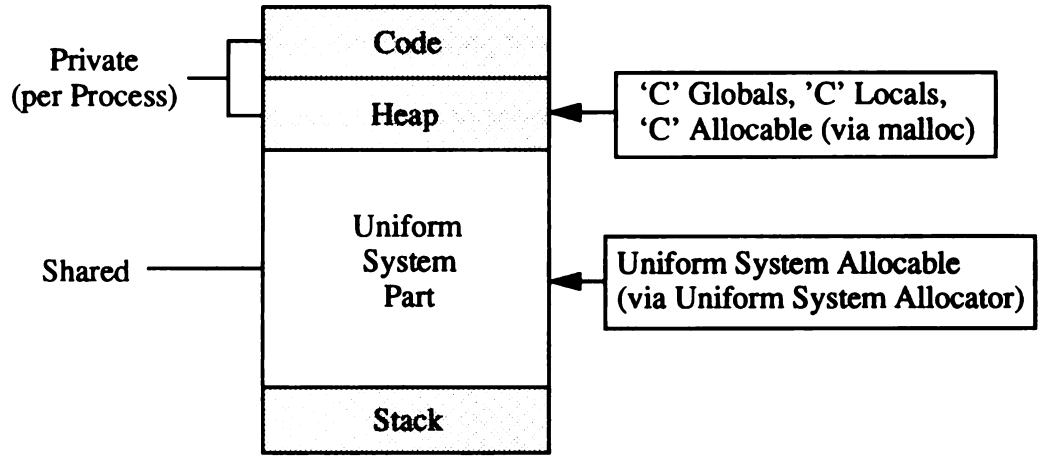


Figure 6.5 Address Space of 'C' Program

FIGURE 6.5 Address Space of 'C' Program

The parallel NC simulation and verification system is implemented in C language. Figure 6.5 shows the general address space map of the system, as seen by the 'C' program. 'C' local variables are process-private and are stored on the stack. A local variable is visible only within the function that declares it. 'C' global variables are also process-private. These variables are shared by functions within the same process, but are hidden from all other processes. 'C' dynamic variables, obtained by *malloc*, are also process private. There is one instance of an allocated variable per process. These variables can be accessed by functions within same process, providing the necessary pointers have been made available. Shared variables, obtained from the Uniform System allocator *UsAlloc*, are globally

shared. Pointers to shared variables are valid on all processors and can be passed freely among them. This is the only way to communicate between different processors and tasks, unless one uses the Mach 1000 mechanisms directly.

In order to distribute tasks equally and reduce memory conflict, we distribute surface data in parametric form among processors and keep toolpath data in shared memory. During surface pre-evaluation, the bounding box, or min-max box, of the surface can be obtained. These bounding boxes help us to select those toolpath segments which possibly affect the particular surface. Toolpath data block transfer from to shared memory to local memory can then occur if necessary. Figure 6.6 shows where the surface data and toolpath data are located in the memory.

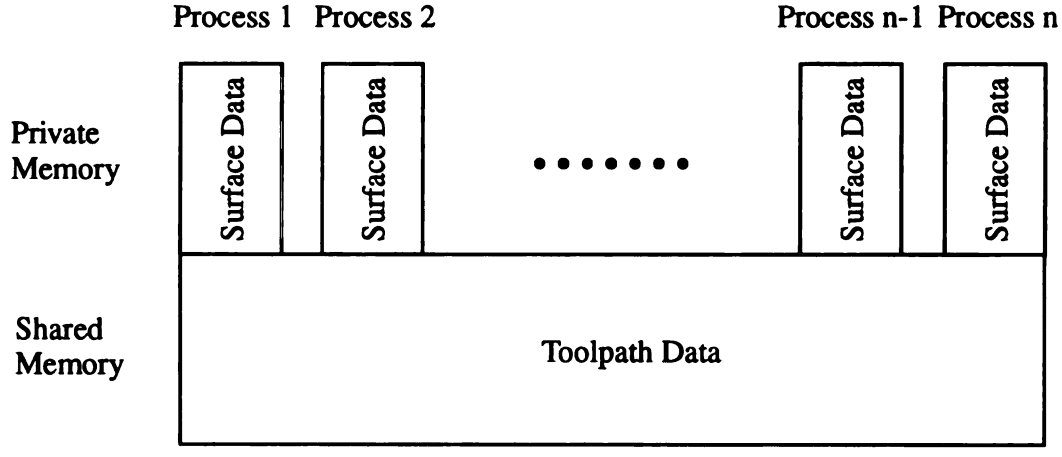


FIGURE 6.6 The Residences of Surface Data and Tool Path Data

As we mentioned earlier, if the huge amount of data is not distributed over all available memories, poor performance will occur. This is because clumping a lot of data in a single processor node's memory can easily result in contention for that memory by mul-

tiple processors. Fortunately, the Uniform System provides memory allocators which allow us to distribute data across the memories of the machine to reduce memory contention. In our system, as shown in Figure 6.7, we choose the UsAllocOnUsProc function to allocate memory on each node and store the parametric surface data at the first stage. Notice that there is no memory contention at this stage because all other processors, except the master processor, have not started to work yet.

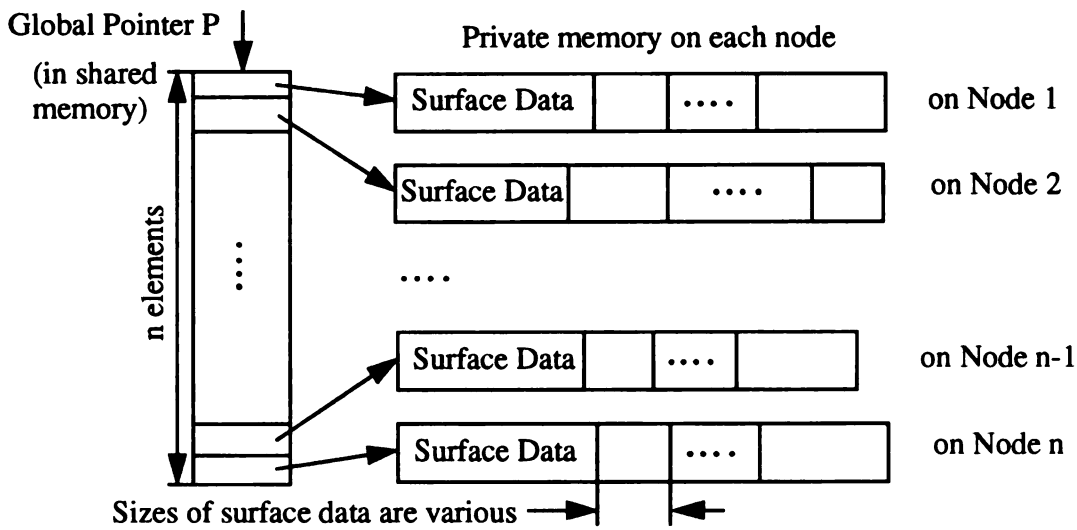


FIGURE 6.7 The Distributed Surface Data Created by UsAllocOnUsProc

It is often useful for each processor to have its own copy of certain frequently referenced variables declared as 'C' globals, such as user-specified surface discretization tolerances, NC verification tolerance, and working space minimum and maximum values. These copies eliminate the memory contention that might otherwise occur as multiple processors access shared copies of the variables. Recall that 'C' globals are in process private memory. There are several ways to copy these private data to each of the other processors. Two main methods are used in our system. The first way to make the values of these vari-

ables accessible to the other processors is to pass the values in the data structure argument to a task generator and have the generator "initialization" routine make copies on each processor. Another way we frequently used is to call the Uniform System function *Share* to propagate variables to processors. For example, assume that *i* is an integer declared as global or static; *i* is therefore process private. The effect of *Share(&i)*, as shown in Figure 6.8, is to copy the value of *i* into each processor that performs tasks generated by subsequent task generators. When processor P1 executes *Share(&i)*, it allocates a share block in shared memory to hold both the address of *i* and the current value of *i*. The share block is linked together with other share blocks and they can be all found when needed because they are in shared memory. When processor P_i begins working on a task generator for the first time, and before the init routine for the generator is called, it finds all of the share blocks that have been linked together. For each share block, P_i copies the value of *i* saved in the share block to the address of *i*, which also was saved in the share block. Allocating *i* in static or global process private memory ensures that the address of *i* is at the same location in all processors. It is important to understand that the value of *i* is propagated to other processors by the *Share* mechanism, but the variable *i* itself is not in shared memory. Therefore, should one processor change its copy of *i*, only that processor will see the changed value.

Processor management is accomplished using task generators. Although the Uniform System provides several generator families, such as the index family, array family and half array family, we simply used the index family generator GenOnIFull because it generates a task for each value (index) within the range when given an integer range. GenOnIFull has the following form:

code = GenOnIFull(Init, Verifier, Final, Arg, Range, Limited, Abortable);

Init(Arg) is called on a particular processor before the generator calls Verifier(Arg, Index) for the first time on that processor. Init(Arg) is used to copy frequently referenced constants from globally shared memory into process private memory or to initialize private temporaries. Verifier(Arg, Index) starts the NC verification process on that processor. After the last call of Verifier(Arg, Index) on each processor used to perform tasks for the generator, Final(Arg) is called once on each such processor used, to do post-processing associated with the tasks. the "Limited" parameter specifies the number of processors to which the generator is to be restricted. The "Abortable" parameter indicates whether or not the generator can be aborted.

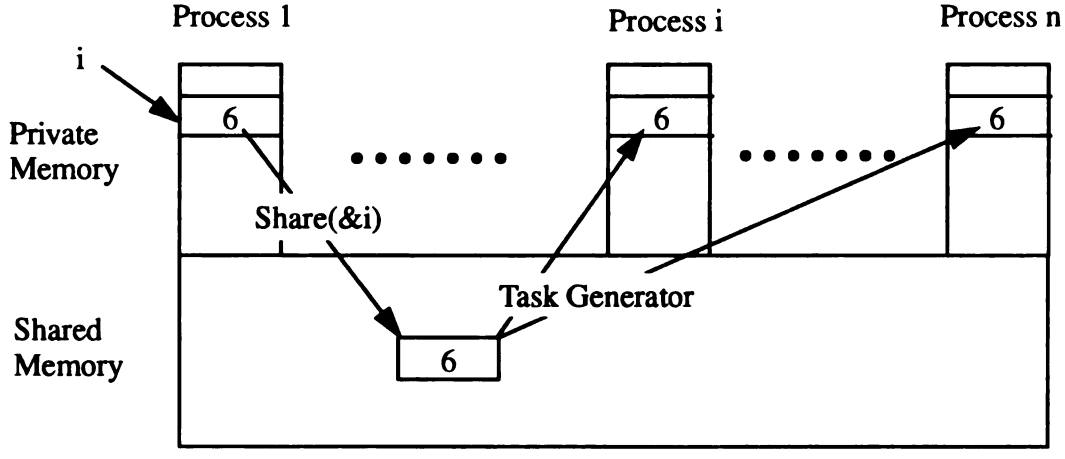


FIGURE 6.8 Share Passes Copies of Process Private Variable

CHAPTER VII

EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

The proposed parallel approach for NC toolpath verification is practical and realizable. In this chapter, a series of test cases is presented. These test cases are chosen to illustrate the complexity of the problem, as well as the correctness and efficiency of the algorithm. All the test cases are based on realistic models of manufactured sculptured surfaces. The verification results are obtained through a batch process, and can be displayed in X Window environment. The color-coded verification results make it easy for the user to identify undercut or overcut areas. The functionality of querying cut values provides the user a tool to determine exact cut values at points of interest on part surfaces. Views from arbitrary viewpoints based on the same results, without rerunning the verification process, are given to illustrate the view independence of the program.

7.1 Benchmark Application Models

This section presents two real industrial part surface models. The first example is a file provided by CIMLINC, a CAD/CAM software company, and is a model frequently used to benchmark CAD/CAM systems. The model consists of multiple NURB surfaces, including trimmed surfaces. The toolpath was also generated by CIMLINC. A real work-

piece was provided to facilitate comparison of the verifier's output with the part as actually cut by the toolpath being verified. The second example demonstrates five-axis NC machining. The surfaces were created in PDGS[41], a CAD/CAM software package developed by the CAD/CAM/PIM department of Ford Motor Company and widely used within Ford and its suppliers. The toolpath was generated by CHIPS[42], a robust cutter path generation package which is also developed and widely used by Ford Motor Company. Two different toolpaths -- one a gouge-protected cutter path and the other not protecting against gouging -- are used to demonstrate the verification results.

For the first example, we conducted three different experimental cases. In all three cases, the same toolpath file and cutter definition files were used. The verification parameters for inward tolerance (*intol*) and outward tolerance (*outtol*) were set to 0.025mm, while *range of interest* was set to 5.0mm in all three cases. In Case 1, all of the sculptured surfaces were discretized by letting *chordal deviation* equal 1.0mm and maximum parametric stepover equal 2.0mm. This generated 21,485 discretized surface points. It took about 2 minutes to complete the verification process on a SPARC20 workstation with 64 MBRAM, 60 Mhz CPU and SunOS 5.5. In order to test parallel speedup, however, the same run was repeated twice, using one and four processors, on a BBN GP1000 machine. While the processors on this machine are obsolete, it nonetheless constitutes a reliable platform on which to examine the parallel speedup of this algorithm. It took 4 hours, 47 minutes and 16 seconds to complete the verification process on one processor node, and 1 hour, 18 minutes and 19 seconds with 4 processor nodes. The speedup in this case is about 4. In Case 2, the *chordal deviation* was set to 0.02mm and maximum parametric stepover was set to 0.5mm, and 306,266 discretized surface points were obtained. Verifica-

tion took about 20 minutes on a 60MHz SPARC20. In Case 3, the above two parameters for surface discretization were set to 0.02mm and 0.25mm, respectively. Verification took 1 hour, 18 minutes and 36 seconds to complete on a 60MHz SPARC20.

The experimental results are shown in Figure 7.1 to Figure 7.3. Figure 7.1 shows the original part surfaces from two different views. Figure 7.2 is a plot of the control point of the tool as it moves along the toolpath. Three pictures were captured at different points in time. An additional picture was obtained from another viewpoint. The tool was drawn only on the first toolpath, and the tool motions were depicted with the toolpath in yellow and the tool axis vector in blue. The main reason for only showing the tool once is to reduce the graphics-related computation and the clutter of the image. Figure 7.3 shows the verification results based on the user-specified display parameters. Our system actually maintains and uses two copies of the set of parameters *intol*, *outtol* and *range of interest*. The first set is called the computational parameter set, and is the values of *intol*, *outtol* and *range of interest* used during a verification calculation. A second copy is called the display parameter set, and allows changing those parameters to values slightly different from those used during verification and displaying approximately correct results without redoing the verification calculations. That is, the user can arbitrarily set the display parameters to display the verification results without rerunning the NC verification process, as long as the display parameters are within the general range of the computational parameters, and the display *intol* and *outtol* are both within the computational *range of interest*. In this example, we ran the NC verification at 0.025mm (*intol* = *outtol* = 0.025mm) with *range of interest* equal to 5.0mm. However, we set a display band at 0.05mm and 5.0mm in Figure 7.3 (a), (b) and (c). Figure 7.3 (c) is a zoomed view of the

result. This picture appears to match perfectly the prototype part actually cut. Figure 7.3 (d) shows the result when we set the display tolerances to the 0.025mm values used for running the job. In other words, although the display parameters are different in Figure 7.3 (d) from those in Figure 7.3 (a), (b) and (c), the same verification computation was used for all the displays. The different display results can be obtained simply by redefining the display bands, which sets the reference between color and cut value. The color-coded verification result image shows the areas within or outside the display tolerance. Green represents the area cut within tolerance. Gouges deeper than *intol* vary from red to dark pink for distance from *intol* to *range of interest.* Undercuts greater than *outtol* are shown in hues from dark blue to light pink for distances from *outtol* to *range of interest.* For intelligibility of the output, intensities of all hues are varied according to the angle between the eyepoint and the part surface normal. From Figure 7.3, we can see that more gouging (red) area shows up, of course, when *intol* and *outtol* values are decreased (tolerances are tightened). For highest accuracy of the verification, display values should match the computational values, but for quick exploration of the regions cut within various tolerances, simple resetting of the display bands yields fairly accurate results with almost no computation time, once the initial verification has been done.

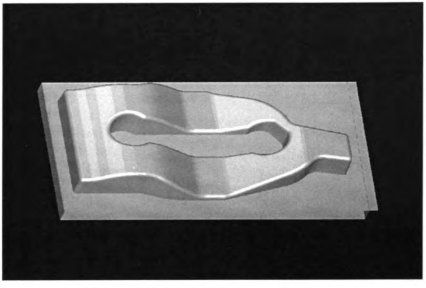


FIGURE 7.1 (a) Original Part Surfaces

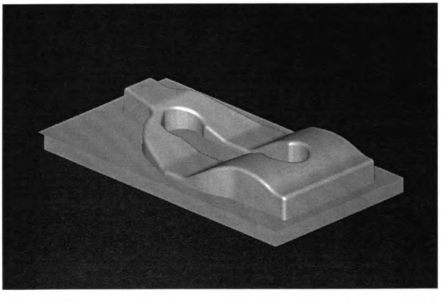


FIGURE 7.1 (b) Original Part Surfaces

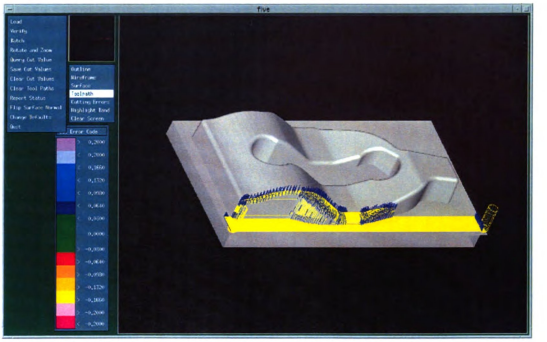


FIGURE 7.2 (a) NC Toolpath Display

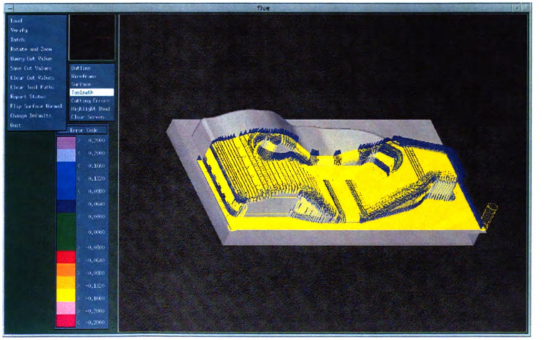


FIGURE 7.2 (b) NC Toolpath Display

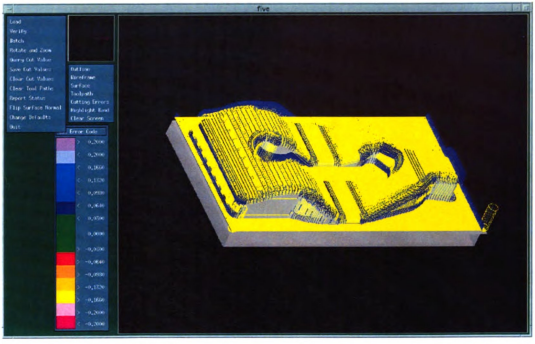


FIGURE 7.2 (c) NC Toolpath Display

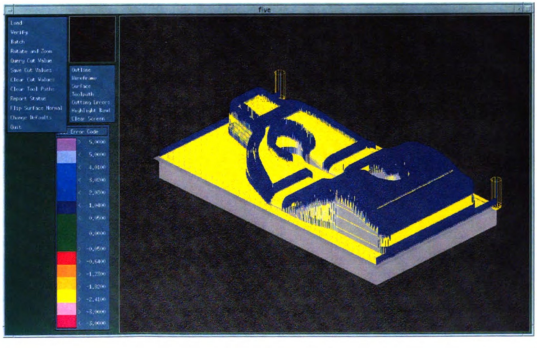


FIGURE 7.2 (d) NC Toolpath Display

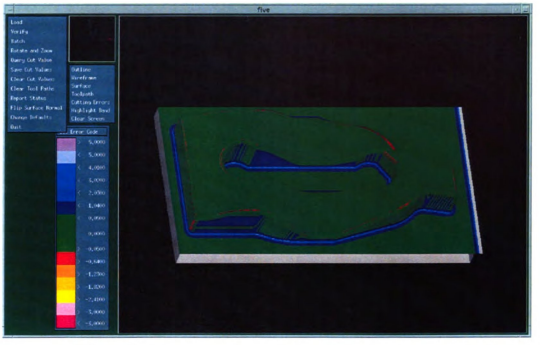


FIGURE 7.3 (a) Verification Result (*intol*=0.05mm, *outtol*=0.05mm)

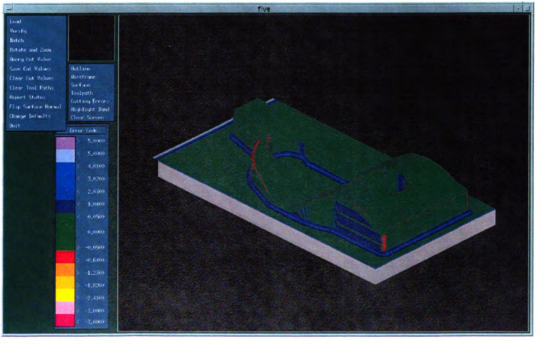


FIGURE 7.3 (b) Verification Result (*intol*=0.05mm, *outtol*=0.05mm)

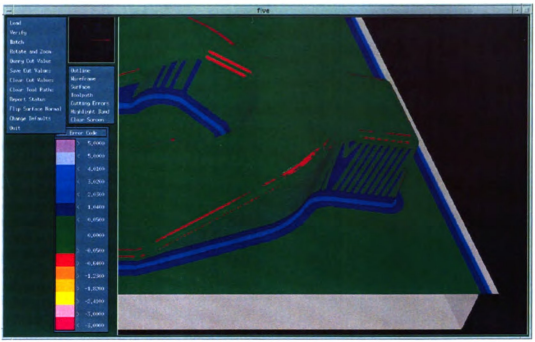


FIGURE 7.3 (c) Verification Result (*intol*=0.05mm, *outtol*=0.05mm)

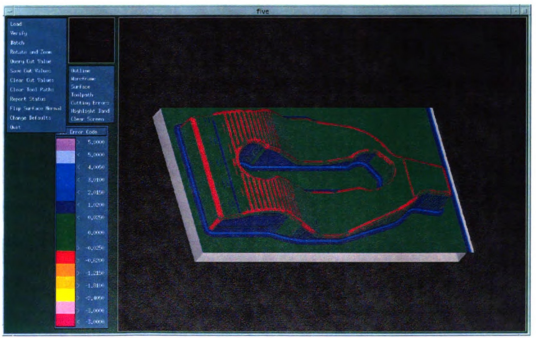


FIGURE 7.3 (d) Verification Result (Display For *intol*=0.025mm, *outtol*=0.025mm) Without Rerun Computational Job

In the second example, we verified three different toolpaths on the same sculptured part surfaces. The first toolpath was generated without considering gouge protection. The second case uses gouge-protected toolpath data. In both cases, the toolpath attempts to cut the entire part. It took much more computation time to complete the verification process for this part. It took 45 minutes 41 second to verify 215,933 surface points for 12,288 tool motions on a SPARC20 workstation with 64 Meg. memory, 60 Mhz CPU and SunOS 5.5. It took 56 minutes 39 seconds to verify the same surface points for the 17,500 tool motions of the second case in the same environment. Case three represents a so-called control cut or flowline cut example on the same part surfaces. This kind of cut tries to cut certain areas of the part determined by a so-called control surface. In all three cases, the same part surface file and cutter definition files were used. The verification parameters *intol* and *outtol* were set to 0.025mm, while *range of interest* was set to 10.0mm in all three cases.

The experimental results are shown in Figure 7.4 to Figure 7.6. Figure 7.4 depicts the original part surfaces. Figure 7.5 illustrates portions of the NC toolpath. Again, the tool was drawn only at the beginning of the toolpath, and tool motions were represented by the toolpath in yellow and the tool axis vector in blue. Figure 7.6 shows the verification results based on the verification parameter *intol* = 0.002mm, *outtol* = 0.002 and *range of interest* = 10mm. All the results are from the same verification process without rerunning the program. The display band in Figure 7.3 (a) is defined as *intol*=0.002mm and *outtol* = 0.05mm. Figure 7.3 (b) gives the result when *intol* is changed from 0.002mm to 0.05mm. A toolpath produced without considering gouge protection was used for both Figure 7.6 (a) and (b). In Figure 7.6 (c) and (d), we used a gouge-protected toolpath. Figure 7.6 (c) shows that the quality of the toolpath was improved a great deal, as expected. Further, the

verifier detects a certain area where a tool protrusion collided with the part. Figure 7.6 (d) demonstrates how we can investigate the gouged area. If we use the mouse to pick a gouged point of interest on part, the program tells us the depth of the gouge and which tool motion causes the problem. In Figure 7.6 (e), the result of a so-called control cut is shown. The toolpath only cuts a particular area of interest, under the direction of a control surface. This kind of cut is also widely used in the manufacturing world.

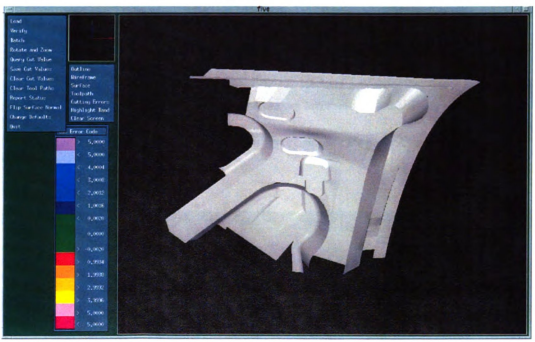


FIGURE 7.4 Original Part Surfaces for Example 2)

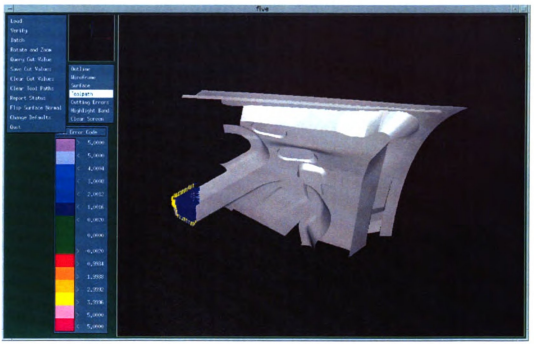


FIGURE 7.5 (a) NC Toolpath Display

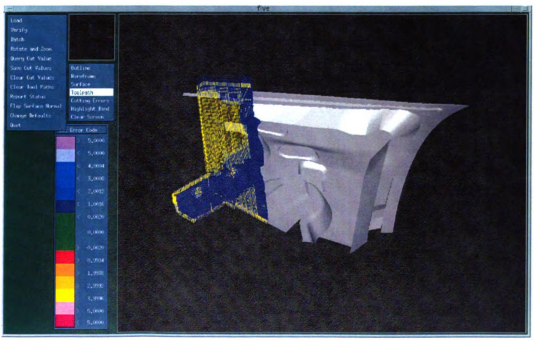


FIGURE 7.5 (b) NC Toolpath Display

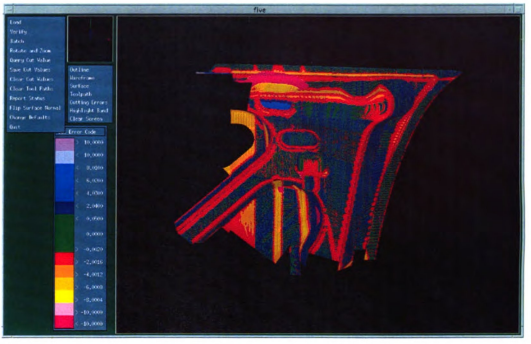


FIGURE 7.6 (a) Verification Result (*intol*=0.002mm, *outtol*=0.05mm)

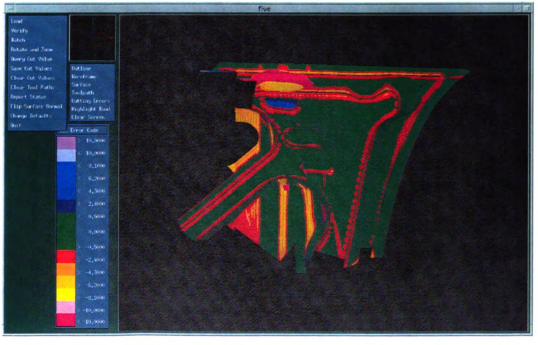


FIGURE 7.6 (b) Verification Result (*intol*=0.002mm, *outtol*=0.05mm) on Toolpath with Gouging

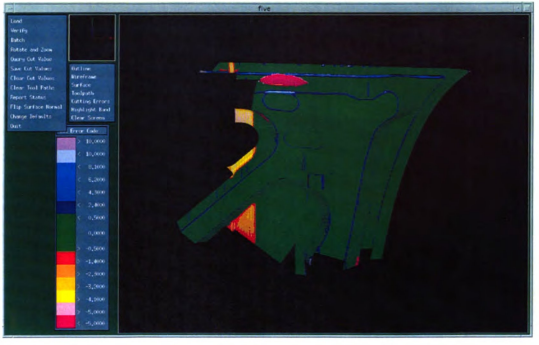


FIGURE 7.6 (c) Verification Result (*intol*=0.002mm, *outtol*=0.05mm) on Toolpath with Gouging Protection

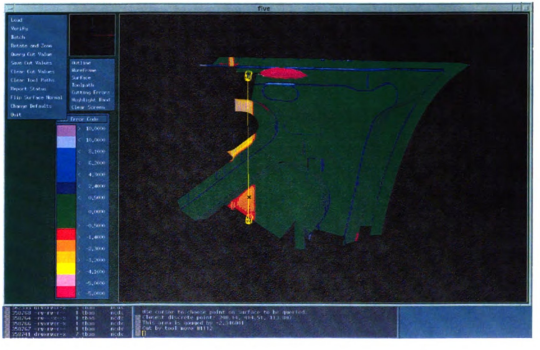


FIGURE 7.6 (d) Querying Verification Result (2.3mm Gouging at X point, and 1112th Tool Motion Causes the Gouging)

FIGURE 7.6 (e) Verification Result (*intol*=0.5mm, *outtol*=0.35mm) on Control Cut Toolpath with Gouge Protection

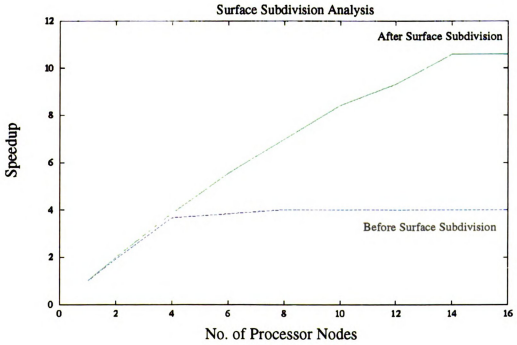


FIGURE 7.7 Speedup on Different Surface Subdivision

7.2 System Performance Evaluation

The proposed parallel NC simulation and verification approach provides a new tool to speed up the time-consuming manufacturing process. Full parallelism, better system performance, and high accuracy can be achieved using the parallel simulation scheme. The approach is especially good for SPMD machines because verification procedures on each processor are the same. We distribute the part surface in parametric form, instead of a set of surface points, to reduce the network traffic and parallelize more of the total workload. We believe that the approach to parallelization and process simulation exemplified here for NC verification is useful for many large-scale problems in manufacturing process simulation, such as spray robot simulation, FEM mesh generation, crash analysis, etc.

The proposed parallel NC toolpath verification scheme has been shown to be practical and realizable. We have done some experiments with it on a butterfly-architecture machine. The test model and toolpaths we have used are given in the previous section. The results to date have been quite encouraging and match what was predicted. Figure 7.7 is the result of two different experiments -- one distributes the part surfaces without considering surface subdivision, and the other subdivides surfaces before distributing them. The figure shows that the maximum speedup can be limited if surface subdivision is not done. As the blue line indicates, the maximum speedup is about 4, due to the unbalanced load among the processors. Some processor has much more work to do than others. The time spent on this node dominates the overall system performance and the end effects plays major role, so the maximum speedup is limited. When the surface data is subdivided -- in other words, more surface patches are generated -- a better processor load balance can be

achieved. The green line in Figure 7.7 shows that the maximum speedup is improved when part surfaces are subdivided. In the former experiment, there are 22 surface patches, while there are 51 surface patches in the latter experiment.

Figures 7.8 and 7.9 depict the relationships among problem sizes, processing times and speedup. The results have been quite positive. As shown in the figure, the peak speedup achieved to date is about 10, among 16 processors. This is mainly due to the network contention during reading the shared memory. As we can see, as the problem size becomes larger -- i.e., the surface discretization tolerance decreases and more surface points are created to represent the part surfaces -- the system speedup increases as the green line shows. The larger the number of surface points, the better the speedup which might be achieved, because more computations can be conducted in parallel. In the experiment, three different problem sizes -- large, medium and small -- were used, and they are represented by green, red and dotted curves, respectively. The total number of toolpath segments was kept the same for all three cases. The experiment data show that the green curve has the best speedup, peaking at 10.6. Due to the limitations of the Butterfly machine used, the largest sized run still used only 21,485 surface points.

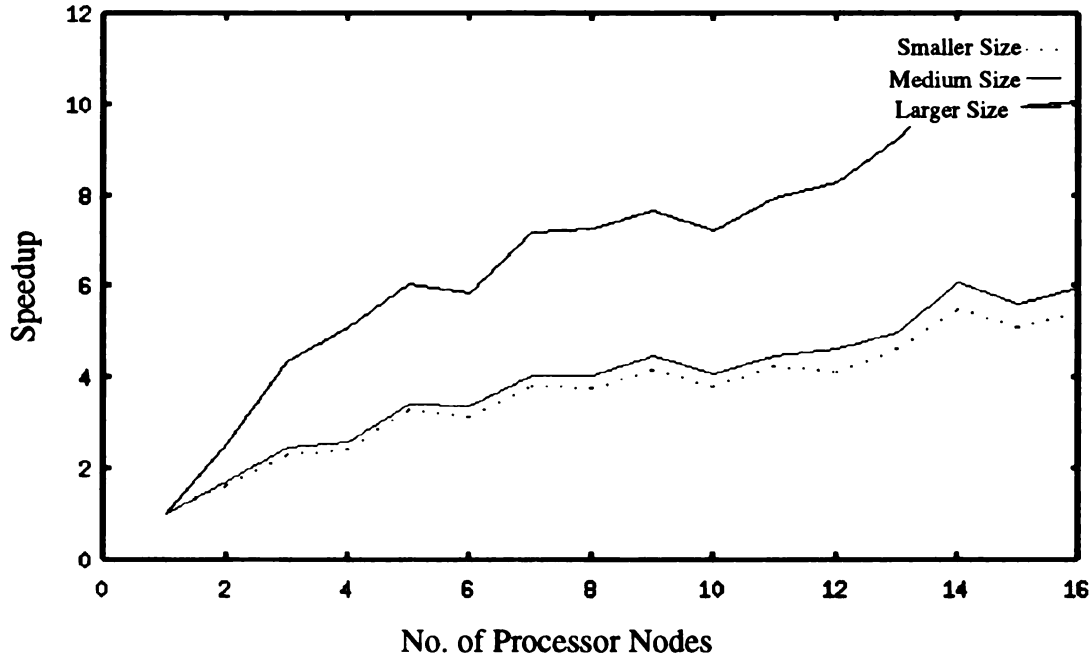


FIGURE 7.8 Speedup on Different Problem Sizes

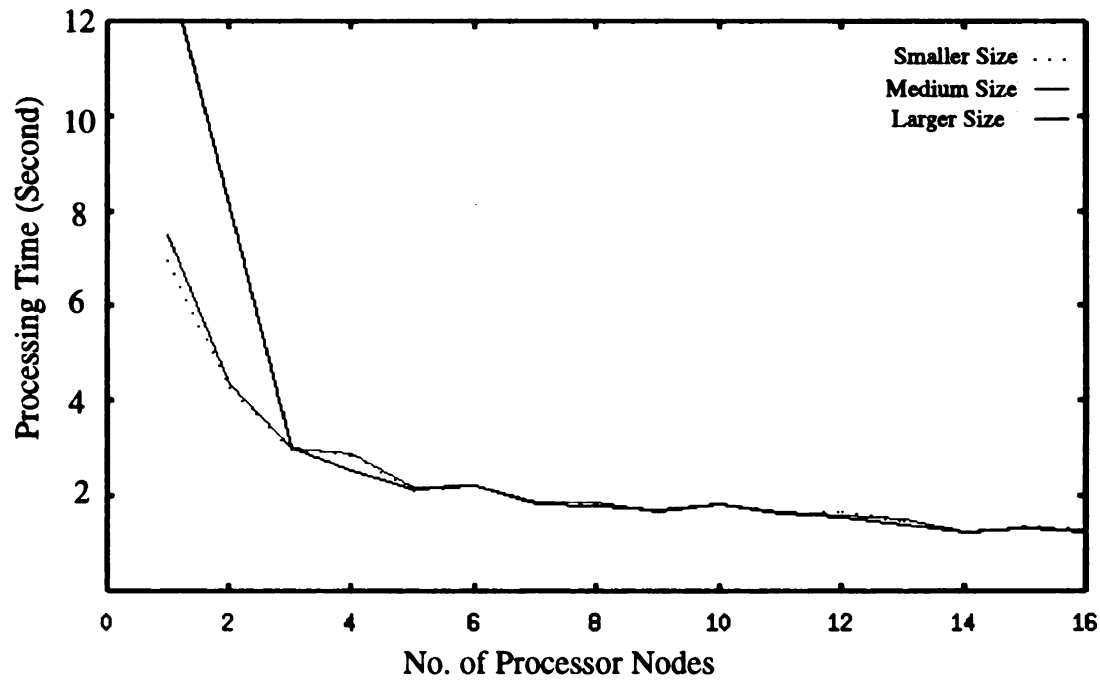


FIGURE 7.9 Processing Time on Different Problem Sizes

CHAPTER VIII

SUMMARY AND CONCLUSIONS

The high cost of conducting test runs of NC programs has created a strong demand for simulators and verifiers for such programs. Current NC simulation software on sequential machines greatly improves the verification procedure. However, such NC simulators are still inaccurate and most can handle only 3-axis milling with the simplest types of cutters. Given the former dearth of sophisticated software for generating optimal 5-axis toolpaths for more complex tools, this inadequacy of simulators or verifiers was not very noticeable. The current trend toward higher-capability NC programming systems raises the strong need for more sophisticated and accurate verification. This trend highlights the trade-off between verification time and accuracy of verification, strongly raising the desire for parallelization of verification or simulation processes. However, application of parallel or distributed architectures to such simulation and verification tasks, etc., is non-trivial and still rare. Such tasks are not "embarrassingly parallelizable," and decomposability of a verification algorithm can be aided significantly if it is designed with this in mind.

In this dissertation, a tool model and a surface model for NC simulation and verification are first presented. The tool motion envelope is implicitly defined by the ruled surface that is determined by the motion of the axis of the cutting tool. Part surfaces are defined by hundreds and thousands of triangles composed of discretized surface points.

The triangles are mainly used for display purposes, while surface points are the fundamental elements on which the NC verification can be performed. A novel surface discretization algorithm is developed for both surface tessellation and load balance estimation. The beauty of the algorithm is that it kills two birds with one stone -- it not only guarantees that tessellated surfaces are within user-specified tolerance, but also estimates and keeps load balance directly by the use of parameters of the original geometry, such as order of the surfaces, control points and knots. The load estimates produced facilitate good parallel NC verification processing by guiding the assignment of tasks to processors. Hence, we can say that the function we derived, which determines the relationships among geometry, load balance and program performance, is the key characteristic of our new parallel NC simulation and verification program.

The software developed using the algorithm has been proved useful and practical for verifying multi-axis NC machining programs with APT cutters. The input to our software includes three parts -- a standard IGES file that defines desired part geometry, CL files that define cutter locations or toolpaths during machining, and user-specified parameters that determine surface discretization tolerances and verification tolerances. As soon as input data is defined, a generic command file is created. The command file can drive the software on either a sequential or parallel machine. The user can also directly write a command file and run the software through command lines. The program begins by reading surface data, then pre-evaluates all the sculptured surfaces and estimates the load factors for each surface. It then discretize the surfaces and transforms all surface points into a computational space which is based on voxels. Toolpath segments are also transformed into voxel space. Only the surface points possibly affected by a given tool motion will be

used for further computation. The verification process mainly computes the distances between surface points and tool swept volumes, which are derived from the locus of the tool axis, without explicitly creating the surface boundary of the tool motion (swept volume). The result of the verification process is saved in a "cut value" data file that can be used to display the output images in a variety of formats. The output display image uses pseudo color coding to show the areas cut within tolerance and out of tolerance in different colors. A color map is provided to help the user determine the depths of gouging or undercut. The verification process is view independent, which allows the user to do detailed checking of the verification results without repeating the computationally intense verification calculations. The user can also query the cut value at any surface point of interest. The verification results may be used to adjust the toolpath to eliminate overcut or undercut regions, or to generate additional toolpaths using a smaller cutter to clean up the undercut regions.

There are some limitations of the system, especially for irregular toolpath patterns such as the ones generated by *control cuts*. As shown in the previous chapter, a *control cut* creates a cutter path which concentrates on a subregion of the part. In this case, the computational job loads are not equally distributed. Those surfaces with a high density of points may require no verification calculations, provided they are not affected by the toolpath, while most of the computational time is spent on a few surfaces with smaller point density. An improvement would be needed in our current algorithm in order to better allocate load in that circumstance. Pre-evaluation of surfaces taking into consideration the toolpath data, or dynamically balancing the load, could address that problem.

The algorithm brings high technology, in the form of parallel processing, into the "old world" in which most researchers still focus on finding better and more efficient algorithms to model the tool motion for the simulation and verification of NC machining. It is the result of interdisciplinary knowledge, including geometric computation, manufacturing processes, parallel computing, etc. There are two major differences between our current system and other NC verification systems. First, it is surface based, so that the effort needed to calculate the intersection of surface points and tool swept volumes is greatly reduced because each surface point has a local coordinate function for each tool motion. In most solid-modeling-based systems -- for instance, VeriCut of CG Tech. -- most of the verification time is spent on computing the boundary of each tool motion or combining the boundaries of all the tool motions, and on computing the intersection between the desired part surface and the tool swept volume by Boolean operations. Since tool swept volumes for multi-axis tool motion are very complicated, it requires huge CPU time to complete the job. The second characteristic of our system is its parallel processing basis, so that it requires much less time to perform a large verification job, compared to verification on a sequential machine. It addresses the load balance problem right at the beginning of surface evaluation. It tends to minimize communication intensity and redundant computation by passing parametric surface data and taking geometry locality into consideration. Full parallelism, better system performance, and high accuracy can be achieved using the parallel simulation scheme. The algorithm is especially good for SPMD machines because verification procedures on each processor are the identical. For large-scale problems, this algorithm can be applied and extended to distributed environments. In Yung's system[23], the only other parallel NC verification program available so far, two major factors -- load balance and system performance -- were not addressed. In addition, it is a CSG-based system.

The algorithm is realizable not only on parallel machines, but also in a distributed environment, such as networked workstations. In a distributed system, the network communication time is relatively large, so that it is not advisable to apply the approach to small problems. The differences between a parallel machine and a distributed system are in the communication overhead and system cost. A parallel machine can greatly reduce the communication overhead, but at a considerable cost. The overall system performance could become worse than the serial case in a distributed environment, if the problem size is not large enough. However, for more difficult verifications, with hours or days of execution time on single-CPU machine, the communication overhead is negligible, even on a distributed system. Therefore, a distributed system might be cost-effective.

The mathematical basis presented in this dissertation can also be used in other applications, such spray robot simulation, collision analysis, and welding process simulation. It can also be extended to a distributed computation environment, especially for certain real manufacturing problems which require hours to days of computation. For different applications, different issues must be considered. However, there are common foundations -- for example, finding the relationship between the optimal computational performance of a given application and the geometric parameters, problem size, average system communication time, surface discretization limits, etc. This is not a simple problem and is quite problem-dependent, since there are many dynamic system factors that could affect the results. Further study should focus on this area. Further study is also needed regarding maintaining dynamic load balance in case the initial balance is lost. There is always a trade-off among the number of processors, memory contention and network contention to achieve the optimal performance. More processors means larger com-

putation power of the system, but more memory contention and network contention during system initialization and post-processing. Finally, it may be very useful to consider building a closed loop application system or self-correcting system, which can generate a manufacturing solution, verify the results, discover necessary corrective actions and regenerate the solution, all based on user-specified tolerances and parameters. This would probably require linking several distinct technologies, such as knowledge based systems, supercomputers, manufacturing process simulators or verifiers, etc., to realize the dream of manufacturing automation.

APPENDIX A

DEFINITION OF THE APT CUTTER

In the APT language, a quite general cutting tool can be defined by seven parameters, as shown in Figure A.1. The statement defining a cutter or tool is as follows[34]:

CUTTER/d, r, e, f, a, b, L

where

d = The diameter of the circle generated by the intersection of the end and the side surfaces of the cutter.

r = The radius of the corner circle.

e = The distance from the corner circle center to the cutter center line.

f = The distance from the corner circle center to the plane passing through the cutter end center and perpendicular to the cutter center line.

a = The angle (degrees) between the cutter end surface and the plane perpendicular to the cutter axis (it lies in the range of $0 \le a \le 90$).

b = The angle (degrees) between the cutter side surface and the cutter axis (it lies in the range of $-90 \le b \le 90$).

L = The cutter height measured from the control point of the cutter.

Note that usually the corner circle should be tangent to the end and side surface.

For this seven-parameter cutter, the radius of the cutter, R(h), can be defined in terms of linear functions for three different ranges of h, the height along the cutter axis from the tip point of the cutter. The functions can be expressed as follows:

$$R(h) = \begin{cases} \dfrac{h}{\tan(a)} & 0 \le h < f - r\cos(a) \quad (a \neq 0) \\ e + \sqrt{r^2 - (h-f)^2} & f - r\cos(a) \le h < f - r\sin(b) \\ \dfrac{d}{2} + \left(h - \dfrac{d}{2}\tan(a)\right)\tan(b) & f - r\sin(b) \le h \le L \end{cases}$$

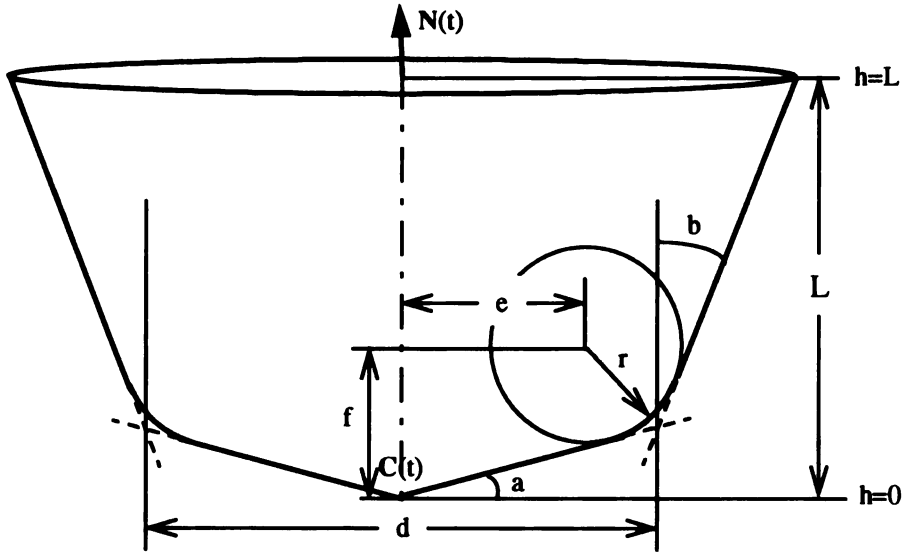


FIGURE A.1 Standard APT Seven Parameter Cutter

APPENDIX B

ESTIMATION OF UPPER BOUNDS ON SECOND DERIVATIVES OF A POLYNOMIAL B-SPLINE SURFACE

Suppose $\hat{S}(u, v)$ is a homogeneous polynomial B-spline surface with degree m and order $k1$ in u direction and degree n and order $k2$ in v direction.

$$\hat{S}(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{i,j} M_{i,k1}(u) N_{j,k2}(v) \qquad \text{(EQ B.1)}$$

In order to estimate the subdivision depths in u and v directions respectively, we need first to estimate the upper bounds of M_1, M_2 and M_3 defined by Filip *et al.*[25]. The solution of the upper bounds of M_1, M_2 and M_3 will guarantee that the tessellated surface be within ε tolerance.

According to de Boor[26], the first, second and nth derivative of a B-spline curve $C(t) = \sum_{i=0}^{m} P_i M_{i,k}(t)$, with degree m and order k, can be written as

$$\frac{d}{dt} C(t) = (k-1) \sum_{i=1}^{m} \frac{P_i - P_{i-1}}{t_{i+k-1} - t_i} M_{i,k-1}(t)$$

Let's define

$$P_i^{(1)} = \frac{P_i - P_{i-1}}{t_{i+k-1} - t_i}$$

$$P_i^{(2)} = \frac{P_i^{(1)} - P_{i-1}^{(1)}}{t_{i+k-2} - t_i}$$

$$P_i^{(n)} = \frac{P_i^{(n-1)} - P_{i-1}^{(n-1)}}{t_{i+k-n} - t_i}$$

Then first, second and nth derivative of $C(t)$ will be of the following forms:

$$\frac{d}{dt}C(t) = (k-1)\sum_{i=1}^{m} P_i^{(1)} M_{i,k-1}(t)$$

$$\frac{d^2}{dt}C(t) = (k-2)(k-1)\sum_{i=2}^{m} P_i^{(2)} M_{i,k-2}(t) \qquad \text{(EQ B.2)}$$

$$\frac{d^n}{dt}C(t) = (k-n)\ldots(k-1)\sum_{i=n}^{m} P_i^{(n)} M_{i,k-n}(t)$$

Now applying the properties to regular polynomial B-spline surface $\hat{S}(u, v)$

$$\hat{S}(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} P_{i,j} M_{i,k1}(u) N_{j,k2}(v) = \sum_{i=0}^{m} P_i^{(0)} M_{i,k1}(u)$$

where $P_i^{(0)} = \sum_{j=0}^{n} P_{i,j} N_{j,k2}(v)$

Therefore the first derivative of $\hat{S}(u, v)$ can be written as

$$\frac{\partial}{\partial u}\hat{S}(u, v) = (k1 - 1)\sum_{i=1}^{m} P_i^{(1)} M_{i, k1-1}(u) \qquad \text{(EQ B.3)}$$

where

$$P_i^{(1)} = \frac{P_i^{(0)} - P_{i-1}^{(0)}}{u_{i+k1-1} - u_i}$$

$$= \frac{1}{u_{i+k1-1} - u_i}\left[\sum_{j=0}^{n} P_{i,j} N_{j,k2}(v) - \sum_{j=0}^{n} P_{i-1,j} N_{j,k2}(v)\right]$$

$$= \frac{1}{u_{i+k1-1} - u_i}\sum_{j=0}^{n} (P_{i,j} - P_{i-1,j}) N_{j,k2}(v)$$

Now let us consider the second derivative of $\hat{S}(u, v)$

$$\frac{\partial^2}{\partial u}\hat{S}(u, v) = (k1 - 2)(k1 - 1)\sum_{i=2}^{m} P_i^{(2)} M_{i, k1-2}(u) \qquad \text{(EQ B.4)}$$

where

$$P_i^{(2)} = \frac{P_i^{(1)} - P_{i-1}^{(1)}}{u_{i+k1-2} - u_i}$$

$$= \frac{1}{u_{i+k1-2} - u_i}\left(\frac{\sum_{j=0}^{n}(P_{i,j} - P_{i-1,j})N_{j,k2}(v)}{u_{i+k1-1} - u_i} - \frac{\sum_{j=0}^{n}(P_{i-1,j} - P_{i-2,j})N_{j,k2}(v)}{u_{i+k1-2} - u_{i-1}}\right)$$

$$= \frac{\sum_{j=0}^{n}[(u_{i+k1-2} - u_{i-1})(P_{i,j} - P_{i-1,j}) - (u_{i+k1-1} - u_i)(P_{i-1,j} - P_{i-2,j})]N_{j,k2}(v)}{(u_{i+k1-2} - u_i)(u_{i+k1-1} - u_i)(u_{i+k1-2} - u_{i-1})}$$

$$\leq \frac{\max_{0 \leq j \leq n}[(u_{i+k1-2} - u_{i-1})(P_{i,j} - P_{i-1,j}) - (u_{i+k1-1} - u_i)(P_{i-1,j} - P_{i-2,j})]n}{(u_{i+k1-2} - u_i)(u_{i+k1-1} - u_i)(u_{i+k1-2} - u_{i-1})} \sum_{j=0} N_{j,k2}(v)$$

$$= \frac{\max_{0 \leq j \leq n}[(u_{i+k1-2} - u_{i-1})(P_{i,j} - P_{i-1,j}) - (u_{i+k1-1} - u_i)(P_{i-1,j} - P_{i-2,j})]}{(u_{i+k1-2} - u_i)(u_{i+k1-1} - u_i)(u_{i+k1-2} - u_{i-1})}$$

Let's now define

$$a_u = u_{i+k1-2} - u_{i-1}$$

$$b_u = u_{i+k1-1} - u_i$$

$$c_u = u_{i+k1-2} - u_i$$

$$P_i^{(2)} \leq \frac{\max_{0 \leq j \leq n}[a_u(P_{i,j} - P_{i-1,j}) - b_u(P_{i-1,j} - P_{i-2,j})]}{a_u b_u c_u} \qquad \text{(EQ B.5)}$$

From EQ. B.4 and EQ. B.5, we can have

$$\frac{\partial^2}{\partial u}\hat{S}(u,v) = (k1-2)(k1-1)\sum_{i=2}^{m} P_i^{(2)} M_{i,k1-2}(u)$$

$$\leq (k1-2)(k1-1)\max_{2\leq i\leq m} P_i^{(2)} \sum_{i=2}^{m} M_{i,k1-2}(u)$$

$$= (k1-2)(k1-1)\max_{2\leq i\leq m} P_i^{(2)}$$

$$\leq (k1-2)(k1-1)\max_{2\leq i\leq m}\max_{0\leq j\leq n} \frac{a_u(P_{i,j}-P_{i-1,j})-b_u(P_{i-1,j}-P_{i-2,j})}{a_u b_u c_u}$$

$$= (k1-2)(k1-1)\max_{2\leq i\leq m}\max_{0\leq j\leq n} \frac{a_u P_{i,j}-(a_u+b_u)P_{i-1,j}+b_u P_{i-2,j}}{a_u b_u c_u}$$

By definition from Filip *et al.*[25]

$$M_1 = \sup_{(u,v)}\left\|\frac{\partial^2}{\partial u^2}\hat{S}(u,v)\right\|$$

$$M_2 = \sup_{(u,v)}\left\|\frac{\partial^2}{\partial u\partial v}\hat{S}(u,v)\right\|$$

$$M_3 = \sup_{(u,v)}\left\|\frac{\partial^2}{\partial v^2}\hat{S}(u,v)\right\|$$

Hence, we can obtain Equation B.6 as following:

$$M_1 = \sup_{(u,v)}\left\|\frac{\partial^2}{\partial u^2}\hat{S}(u,v)\right\|$$

$$\leq (k1-2)(k1-1)\max_{2\leq i\leq m}\max_{0\leq j\leq n}\left\|\frac{a_u P_{i,j}-(a_u+b_u)P_{i-1,j}+b_u P_{i-2,j}}{a_u b_u c_u}\right\| \qquad \text{(EQ B.6)}$$

Similarly, by defining

$$a_v = v_{j+k2-2} - v_{j-1}$$
$$b_v = v_{j+k2-1} - v_j$$
$$c_v = v_{j+k2-2} - v_j$$

We can obtain Equation B.7.

$$M_3 \le (k2-2)(k2-1) \max_{0 \le i \le m} \max_{2 \le j \le n} \left| \frac{a_v \boldsymbol{P}_{i,j} - (a_v + b_v)\boldsymbol{P}_{i,j-1} + b_v \boldsymbol{P}_{i,j-2}}{a_v b_v c_v} \right| \qquad \text{(EQ B.7)}$$

Now let's derive the upper bound for M2 which depends on partial derivatives $\frac{\partial^2}{\partial u \partial v}\hat{\boldsymbol{S}}(u,v)$.

$$\frac{\partial}{\partial u}\hat{\boldsymbol{S}}(u,v) = (k1-1)\sum_{i=1}^{m} \boldsymbol{P}_i^{(1)} M_{i,k1-1}(u)$$

$$= (k1-1)\sum_{i=1}^{m} \frac{\sum_{j=0}^{n} (\boldsymbol{P}_{i,j} - \boldsymbol{P}_{i-1,j}) N_{j,k2}(v)}{u_{i+k1-1} - u_i} M_{i,k1-1}(u)$$

$$= (k1-1)\sum_{j=0}^{n} \left[\sum_{i=1}^{m} \frac{(\boldsymbol{P}_{i,j} - \boldsymbol{P}_{i-1,j}) M_{i,k1-1}(u)}{u_{i+k1-1} - u_i} \right] N_{j,k2}(v)$$

$$= (k1-1)\sum_{j=0}^{n} A_j N_{j,k2}(v)$$

$$\frac{\partial^2}{\partial u \partial v}\hat{S}(u,v) = \frac{\partial}{\partial v}(k1-1)\sum_{j=0}^{n} A_j N_{j,k2}(v)$$

$$= (k1-1)(k2-1)\sum_{j=1}^{n}\frac{A_j - A_{j-1}}{v_{j+k2-1}-v_j}N_{j,k2-1}(v)$$

$$= (k1-1)(k2-1)\sum_{j=1}^{n}\left[\sum_{i=1}^{m}\frac{(\boldsymbol{P}_{i,j}-\boldsymbol{P}_{i-1,j})M_{i,k1-1}(u)}{u_{i+k1-1}-u_i}\right.$$

$$\left. -\sum_{i=1}^{m}\frac{(\boldsymbol{P}_{i,j-1}-\boldsymbol{P}_{i-1,j-1})M_{i,k1-1}(u)}{u_{i+k1-1}-u_i}\right]\frac{N_{j,k2-1}(v)}{v_{j+k2-1}-v_j}$$

$$=(k1-1)(k2-1)\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{\boldsymbol{P}_{i,j}-\boldsymbol{P}_{i-1,j}-\boldsymbol{P}_{i,j-1}+\boldsymbol{P}_{i-1,j-1}}{(u_{i+k1-1}-u_i)(v_{j+k2-1}-v_j)}M_{i,k1-1}(u)N_{j,k2-1}(v)$$

$$=(k1-1)(k2-1)\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{\boldsymbol{P}_{i,j}-\boldsymbol{P}_{i-1,j}-\boldsymbol{P}_{i,j-1}+\boldsymbol{P}_{i-1,j-1}}{b_u b_v}M_{i,k1-1}(u)N_{j,k2-1}(v)$$

$$\le (k1-1)(k2-1)\max_{1\le i\le m}\max_{1\le j\le n}\frac{\boldsymbol{P}_{i,j}-\boldsymbol{P}_{i-1,j}-\boldsymbol{P}_{i,j-1}+\boldsymbol{P}_{i-1,j-1}}{b_u b_v}$$

$$\times\sum_{i=1}^{m}\sum_{j=1}^{n}M_{i,k1-1}(u)N_{j,k2-1}(v)$$

$$\le (k1-1)(k2-1)\max_{1\le i\le m}\max_{1\le j\le n}\frac{\boldsymbol{P}_{i,j}-\boldsymbol{P}_{i-1,j}-\boldsymbol{P}_{i,j-1}+\boldsymbol{P}_{i-1,j-1}}{b_u b_v}$$

Therefore, the upper bound, EQ. B.8, of M_2 is as following:

$$M_2 = \sup_{(u,v)}\left|\frac{\partial^2}{\partial u \partial v}\hat{S}(u,v)\right|$$

$$\le (k1-1)(k2-1)\max_{1\le i\le m}\max_{1\le j\le n}\left|\frac{\boldsymbol{P}_{i,j}-\boldsymbol{P}_{i-1,j}-\boldsymbol{P}_{i,j-1}+\boldsymbol{P}_{i-1,j-1}}{b_u b_v}\right| \qquad \text{(EQ B.8)}$$

LIST OF REFERENCE

1. Voelcker, H. B., and Hunt, W.A., "The Role of Solid Modeling in Machine-Process Modeling and NC Verification," *Proceedings of SAE 1981 International Congress and Exposition*, Detroit, Michigan, February, 1981

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

3. Fridshal, R., Cheng, K. P., Duncan, D., and Zucjer, W., "Numerical Control Part Program Verification System," *Proceedings Conference CAD/CAM Technology in Mechanical Engineering*, MIT Press, Cambridge, Massachusetts, 1982, pp. 236-254

4. Chappel, I.T., "The use of Vectors to Simulate Material Removed by Numerically Controlled Milling," *Computer-Aided Design*, Vol. 15, No. 3, May 1983, pp156-158

5. Wang, W. P., *Solid Geometric Modeling for Mold Design and Manufacture*, Ph.D. Dissertation, Cornell University, 1984

6. Wang, W.P., and Wang, K. K., "Geometric Modeling for Swept Volume of Moving Solids," *IEEE Computer Graphics and Applications*, December, 1986, pp.8-17

7. Wang, W.P., and Wang, K. K., "Real Time Verification of Multi-Axis NC Programs with Raster Graphics," *Proceedings of IEEE International Conference on Robotics and Automation*, April, 1986, pp7-10

8. Van Hook, t., "Real-Time Shaded NC Milling Display," *Computer Graphics [Proceeding of SIGGRAPH]*, Vol. 20, No. 4, August 1986, pp. 8-17

9. Satio, T., and Takhashi, T., "Comprehensible Rendering of 3D Shapes," *Computer Graphics [Proceeding of SIGGRAPH '90]*, Vol. 24, No. 4, 1990, pp. 197-206

10. Satio, T., and Takhashi, T., "NC Machining with G-buffer Method," *Computer Graphics [Proceeding of SIGGRAPH '90]*, Vol. 25, No. 4, 1991, pp. 207-216

11. Huang Y., and Oliver J. H., "NC Milling Error Assessment and Tool Path Correction," *Computer Graphics Proceedings*, Annual Conference Series, 1994, pp. 287-294

12. Oliver, J. H., and Goodman, E. D., "Color Graphic Verification of N/C Milling Programs for Sculptured Surface Parts," *First Symposium on Integrated Intelligent Manufacturing*, ASME, New York, 1986

13. Oliver, J. H., and Goodman, E. D., "Computational Verification of Numerical Control Programs for Sculptured Surface Parts," in *Computers in the Design, Construction, and Operation of Automobiles*, T.K.S. Murthy and C.A. Brebbia, Eds., Springer Verlag, New York, 1987, pp. 105-119

14. Oliver, J.H., D. Wysocki and E.D. Goodman, "Gouge Detection Algorithms for Sculptured Surface NC Generation," *ASME Computer-Aided Design and Manufacture of Cutting and Forming Tools*, PED-Vol. 40, pp. 39-44, December, 1989

15. Oliver, J. H., and Goodman, E. D., "Direct Dimensional NC Verification," *Computer Aided Design*, Vol. 22, No. 1, 1990, pp. 3-10

16. Chang, K.Y. and Goodman, E.D. "A Method of NC Toolpath Interference Detection fro a Multi-Axis Machining System," *Control of Manufacturing Processes*, DSC-VOL. 28, PED-Vol. 52, ASME, pp. 23-30, December, 1991

17. Jerard, R.B., Drysdale, R. L., Hauck, K., and Schaudt, B., "Method for Detecting Errors in Numerically Controlled Machining of Sculptured Surfaces," *IEEE Computer Graphics & Applications*, Vol. 9, No. 1, January 1989, pp. 26-39

18. Jerard, R. B., and Robert L. Drysdale, "Methods for Geometric Modeling, Simulation and Spatial Verification of NC Machining Programs," in *Product Modeling for Computer-Aided Design and Manufacturing*, J. Turner, J. Pegna and M. Wozny (Eds.), Elsevier Science Publishers B.V. (North-Holland), 1991, pp. 39-52.

19. Chang, K.Y., *Graphical Verification of Multi-Axis Numerically Controlled Machining Programs for Sculptured Surface Parts*, Ph.D. Dissertation, Michigan State University, 1991

20. Jane L. Hawkins, "The Wonder of B-Splines, Guide to the Mystery and Power of B-Splines and their Variations", Case Center Technical Report, Michigan State University, June, 1987

21. Correns, Martin, "Numerical Evaluation of Nonuniform Rational B-Spline Surfaces," Technical Report, Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, 1989

22. Menon, J. P., and Robinson, D. M., "High Performance NC Verification via Massively Parallel Raycasting," Technical Report, Cornell University, June, 1992

23. Yung, Y. K., *Boundary Surfaces of Tool Swept Volumes using Massively Parallel Algorithm*, Ph.D. Dissertation, Boston University, 1991

24. Cheng, F., "Estimating Subdivision Depths for Rational Curves and Surfaces," *ACM Trans. on Graphics*, Vol. 11, No. 2, April, 1992, pp 140-151

25. Filip, D., Magedson, R., Markot, R., "Surface Algorithms using Bounds on Derivatives," *Computer Aided Geometric Design*, 3, 4, December, 1986, pp 295-311

26. Carl de Boor, *A Practical Guide to Splines*, Springer-Verlog New York Inc.

27. Farin, G., *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide*, Academic Press, San Diego

28. Ni, L., "Parallel Processing", class notes, Computer Science Department, Michigan State University.

29. Mortenson, M. E., *Geometric Modeling*, John Wiley & Sons Inc., 1985

30. Preparata, F. P. and Shamos, M. I., *Computational Geometry: An Introduction*, Springer-Verag New York Inc., 1985.

31. Anderson, R.O., "Detecting and Eliminating Collision in NC Machining", *Computer-Aided Design*, Vol. 10, No. 4, July 1978, pp. 231-237.

32. IGES "Initial Graphics Exchange Specifications, Version 3.0," doc. No. NBSIR 86-3359 Nat. Bur. of Stds., Gaithersburg, MD, USA, 1986

33. IIT Research Institute, *APT Part Programming*, McGraw-Hill Book Company, New York, 1967

34. Chang, C. H. and Melkanoff, M. A., *NC Machine Programming and Software Design*, Prentice-Hall Inc., New Jersey, 1989.

35. Akl, S.G., "The design and Analysis of Parallel Algorithms", Prentice Hall, Englewood Cliffs, 1989

36. Aggarwal, A., Chazelle, B., Guibas, L., O'Dunlaing, C., and Yap, C., "Parallel Computational Geometry", Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 1985, pp. 468-477

37. BBN GP1000 manual

38. Mach 1000 and Uniform System Manual

39. Yunching Huang, James H. Olive, "NC Milling Error Assessment and Tool Path Correction", Computer Graphics Proceedings, Annual Conference Series, 1994, pp287-194

40. CAM-I inc. "APT4 Sculptured Surfaces Part Program Manual," Jan., 1985

41. Ford Motor Company, PDGS User Manual, Release 26, 1996

42. Ford Motor Company, CHIPS User Manual, Release 26, 1996