IOWA STATE UNIVERSITY
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

1993

Dimensional verification and correction of five-axis numerically controlled milling tool paths

Yun-Ching Huang
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

 Part of the Mechanical Engineering Commons

Recommended Citation
Huang, Yun-Ching, "Dimensional verification and correction of five-axis numerically controlled milling tool paths " (1993). *Retrospective Theses and Dissertations*. 10825.
https://lib.dr.iastate.edu/rtd/10825

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

www.manaraa.com

94

13986

U·M·I

MICROFILMED 1994

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Order Number 9413986

Dimensional verification and correction of five-axis numerically controlled milling tool paths

Huang, Yun-Ching, Ph.D.

Iowa State University, 1993

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Dimensional verification and correction of five-axis numerically controlled milling tool paths

by

Yun-Ching Huang

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department: Mechanical Engineering
Major: Mechanical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1993

To Jim and Jeuiling

couldn't have done it without you

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Numerically controlled (NC) milling technology is a production process that directs a cutter through a set of pre-recorded sequential trajectories to fabricate a desired part from raw stock. The technology is capable of producing free-form sculptured surfaces while maintaining tight milling error tolerance. NC milling technology is, therefore, widely used in the production of complicated, high precision, and low quantity products such as molds, dies, and aerospace parts, etc. These products, especially molds and dies, affect many other subsequent production processes, thus the influence of NC milling overall quality and productivity is profound.

In order to improve the accuracy and reliability of NC milling process, verification methods are used, prior to actual production, to check milling tool paths for potential problems such as milling error, collision, improper machining parameters, tool wear, etc. These problems will typically result in undesirable consequences such as unqualified products, machine damage, and personnel injuries. Hence, NC verification is a very important procedure for NC production. Traditionally, NC verification is conducted by observing line drawings of tool paths and performing test milling on soft inexpensive materials. These methods are time-consuming, expensive, and prone of error. Hence they are gradually being replaced by analytical methods. Analytical methods are typically implemented to graphically simulate the milling process off-line and, in some cases, verify milling error, tool assembly collision, and other machining parameters. Consequently, with the help of these tools, NC tool path programmers can visualize the shape of milled parts on a computer monitor and detect potential problems in an more efficient, inexpensive, and accurate way.

1.1 Analytical NC Milling Simulation and Verification Methods

Analytical methods of NC milling simulation and verification are distinct from techniques used to model milling phenomenon and formulate milling problems. These methods can be categorized into three approaches

including direct solid modeling, discrete vector intersection, and spatial partitioning representation. Each of these approaches has been applied to five-axis NC verification but with varying ranges of applicability and degrees of success. The following discussion summarizes the research underlying each approach.

1.1.1 Direct solid modeling approach

The direct solid modeling approach is typically implemented by using constructive solid geometry (CSG) or boundary representation (B-rep) solid modeling systems. Since the regularized Boolean set operations, union, difference, and intersection, are supported in these modeling systems, the implementation of milling simulation is straightforward: the workpiece and tool are modeled by solids, and the material removal process is simulated via a series of regularized Boolean difference operations that subtract successive tool models from the workpiece. To achieve higher accuracy, a tool swept volume, instead of individual tool instances, is generally used in the difference operation. Thus the milling process can be realistically represented and the result is an explicit solid model of the milled workpiece.

A three-axis milling simulation and verification system was proposed in Sungurtekin and Voelcker [1986] which employs a CSG solid modeling system. CSG primitives such as blocks, spheres, cylinders, etc., are used to model the workpiece, fixtures, and milling tools in a milling setup. This system is also capable of representing linearly swept objects, which is applied to extrude a milling tool model along a three-axis tool trajectory to represent its swept volume. Simulation of milling is performed by utilizing regularized Boolean difference operations that subtract the tool swept volumes from the workpiece model to generate a solid model of the milled workpiece. The mathematical formulation of a seven-parameter APT tool sweeping over a general five-axis trajectory, as illustrated in Figure 1.1, were developed in Narvekar, et al. [1992].

Since the milled part is explicitly defined by a solid representation scheme, CSG, for example, subsequent analysis and computation of milling error, volume removal rate, or milling dynamics can be performed. For instance, the milling error is computed as the difference between milled part (A) and designed part (B) by using regularized Boolean difference operations, i.e., A-B represents the solid geometry of undercut material and B-A represents

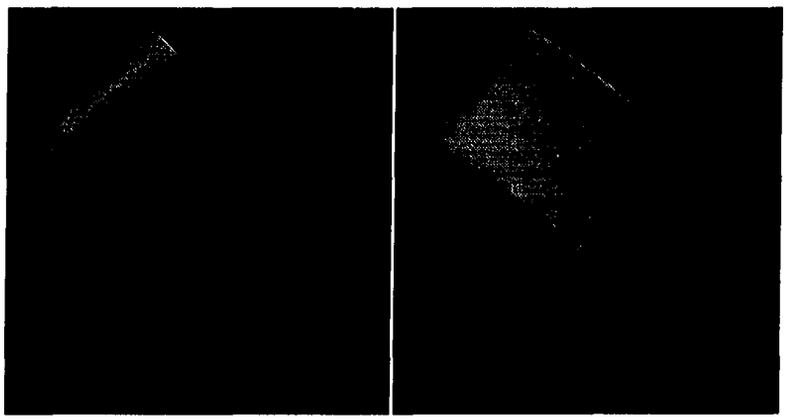


Figure 1.1 Five-axis tool swept volume

overshoot relative to the design surfaces. Furthermore, the dimension of undercut or overshoot at a specified point ($\boldsymbol{P}$) of A can be obtained by computing the minimum distance between $\boldsymbol{P}$ and B. The verification of volume removal rate and machining dynamics are performed from the computation of intersection volume between tool swept volume and workpiece model. Dividing the intersection volume by the time yields the volume removal rate, hence the reaction forces between tool and workpiece can be computed.

The direct solid modeling approach is theoretically capable of presenting accurate results of NC verification, however, its application remains limited by the complexity of five-axis swept volume formulation and the evaluation of Boolean set operations between solid entities. Although primitives combined by means of regularized Boolean set operators in CSG can be displayed on a computer monitor using ray tracing techniques without explicitly combining them [Atherton, 1983], ray-tracing five-axis swept volumes is still a time-consuming process [Narvekar, et al., 1992].

1.1.2 Discrete vector intersection approach

The discrete vector intersection approach verifies milling error by computing distances between a set of pre-selected points on design surfaces and tool swept surfaces [Oliver, 1986, Jerard et al., 1989, Chang and Goodman, 1991]. Each design surface point has an associated vector (typically the outward normal), called a point-

vector pair, as shown in Figure 1.2. Verification is performed via intersection between point-vector pairs and tool swept surfaces.

The discrete vector intersection approach is best described in terms of three sub-tasks: discretization, localization, and intersection [Oliver and Goodman, 1990]. The discretization task transforms the design surfaces into a sufficiently dense distribution of surface point-vector pairs to approximate the original design surfaces. The localization process provides a means of extracting a plausible subset of point-vector pairs for each tool motion. Finally, intersection calculation determines the distance between each surface point in the subset and the tool swept surfaces.

The discretization algorithm proposed by Jerard et al. [1989] discretizes the design surfaces into a surface points set (SPS) either by an equally spaced points in Cartesian space or an irregular spacing of points in parametric space. The former uses a grid of equally spaced x-y points to determine the z projecting points on the design surfaces. The latter method selects points in the parametric space of the design surfaces such that closer spacing is used in areas of high curvature. Typical experiments showed the irregular spacing of points in parametric space discretization generates fewer points for a given accuracy, thus increasing the system performance. The associated vector of each point are given by either the unit z-axis vector [Jerard et al., 1989] or the surface outward normal

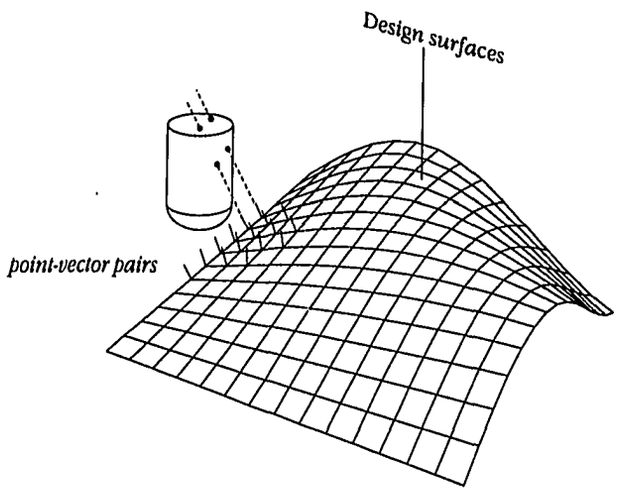


Figure 1.2 Illustration of discrete vector intersection approach

[Oliver, 1986], however, verification results shows that using surface outward normals are comparably more accurate. Oliver [1986] utilized an discretization method which applies an efficient scanline sculptured surface shading algorithm to generate surface coordinate values and normal vector at each pixel (picture element of a raster-type display device) that is in the design surface image. Thus, the amount of surface points is minimized for a given viewing direction and resolution. Chang and Goodman [1991] introduced an object-space surface discretization method to enhance the efficiency of this verification approach. This method discretizes a workpiece and associated holding fixtures into a set of points by limiting the chordal deviation between each adjacent pair of points. Thus the approximation error of the triangular mesh formed by the discrete points is controlled.

The three-axis milling localization algorithm in Oliver [1986] exploits the invariance property of tool axis, i.e., since the tool axis is fixed, a bounding box can be easily established to limit the number of discretized surface points for verification. Localization of five-axis milling is more complicated because the tool axis varies. Two methods were proposed in Jerard et al. [1989], including a short normal method and an average normal method. Both methods assign lengths to vectors associated with points in the SPS and use a localization method variant from the method used for three-axis applications. Furthermore, a 3-D voxel data structure is established to store the discrete points [Chang and Goodman, 1991] for fast localization. This data structure facilitates efficient tool swept volume localization via simple bounding box comparisons. This algorithm presents an effective solution for the localization problem, however, it requires substantial computational effort for the construction of the voxel data structures.

The intersection task computes the intersection of point-vector pairs and tool swept surfaces, and the distances between the intersection points and surfaces points indicate milling errors. Also, the relative location of a intersection point to the design surfaces is determined, i.e., if inside the design surface (relative to the surface normal), negative value (overshoot) is returned, otherwise, positive (undercut) [Oliver, 1986, Jerard et al., 1989]. Similar work in intersection of five-axis tool swept surfaces and rays was presented in Narvekar et al. [1992], which can be used for five-axis applications.

Graphical representation of computed milling errors is typically presented by ranges of color depicting cut values on the shaded image of the design surfaces, so the severity of undercuts and overshoots on the design surfaces can be visually depicted and the amount of milling error can be decoded from an associated color table. For

example, Oliver [1986] develops a color-coding system which uses hues for the cut values and intensities for surface normals, hence the resulting images are very clear and realistic.

Previous research in the discrete vector intersection approach has developed implementations which are efficient, versatile, and accurate. In summary, the discretization algorithms based on the surface curvature or chordal deviation can generate a sufficient number of data points for a given accuracy, hence the memory requirement and computational cost is minimized. The localization problem can be solved by the 3-D voxel data structure to reduce redundant computations of intersection. Furthermore, intersection algorithms have been implemented for most general cases of five-axis milling tool swept volumes. However, the discrete vector intersection approach is not capable of performing milling simulation, or computing material removal rate and machine dynamics.

1.1.3 Spatial partitioning representation approach

The primary disadvantage of the direct solid modeling method for NC verification, i.e., the computational complexity of evaluation of regularized Boolean set operations, motivated the spatial partitioning representation approach. This approach decomposes a solid object into a collection of basic elemental components to simplify the processes of regularized Boolean set operations to one-dimensional z-depth comparison tasks. Moreover, the spatial partitioning representation approach can take advantage of hardware-based systems for enhanced performance [van Hook, 1986, Menon and Robinson, 1992]. Several proposed implementations differ only in the types of basic element used to approximate the solid. These include the dexel data structure [van Hook, 1986], the G-buffer data structure [Saito and Takahashi, 1991], the ray-representation [Menon and Robinson, 1992], and the Graftree data structure [Kawashima et al., 1991].

Van Hook [1986] implements a three-axis NC milling simulation in which solid objects are scan-converted into a set of depth elements, or dexels, as illustrated in Figure 1.3. A dexel is a rectangular solid extending along the z-axis of the screen coordinate system, and is recorded in a data structure containing a pair of depth values for the near and the far ends of the dexel. A two dimensional array, called a dexel data structure and is used to address pixels of a display screen and organize dexels. Hence, it is also called an extended z-buffer data structure. Regularized Boolean

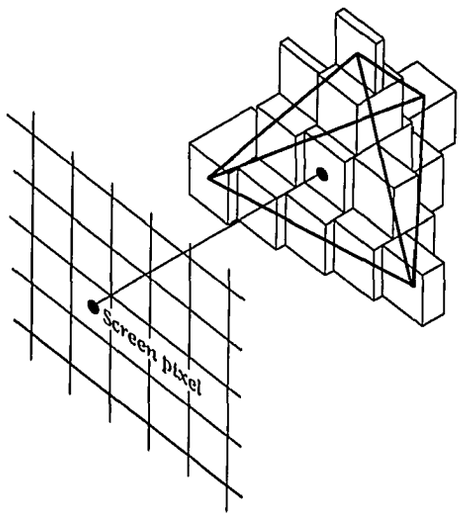


Figure 1.3 Dexel data structure of a tetrahedron

set operations in the dexel data structure are performed by comparing and updating the depth values of a finite number of dexels, hence this implementation results in very efficient and realistic milling simulation.

The G-buffer data structure [Saito and Takahashi, 1991] and the ray-representation [Menon and Robinson, 1992] are similar to the dexel data structure. A G-buffer (geometric buffer) is a two-dimensional array similar to an image buffer. Each G-buffer element contains one geometric property for all pixels, such as depth, surface normal vector, etc. Objects are scan converted into pixel resolution, and several G-buffers are constructed to store a depth value and a geometric property at each pixel, respectively. This representation is primarily applied to computer graphics rendering and tool path generation, however, elementary cut depth verification, tool load, collision avoidance, and feed rate control are also addressed. The ray representation is implemented in combination with a specific hardware platform called the RayCasting Engine (RCE), and hence performing scan conversion processes and image display tasks is very efficient. Five-axis tool motion is supported in this implementation by means of a discrete union of instances, i.e., swept volumes are approximated by a group of tool models that are very close to each other. Hence the need of tool swept volume computation is eliminated. This approach is very useful for milling simulation, or visual verification. Other applications of this ray-representation include tolerance checking, machining dynamics, and touch-sense probe verification.

Kawashima et al. [1991] introduces a Graftree data structure to model the milling process. The Graftree is a geometric-modeling method based on a spatial subdivision technique used for generating octree models. Thus a specific conversion scheme is needed to convert CSG models into the Graftree format. Regularized Boolean set operations are performed by manipulating the Graftree of each object by means of space occupancy tests and a sub-dividing process. A removed volume method was used for milling simulation. This method records the intersecting volume between each tool motion and the workpiece, hence the shape of the workpiece at any instance of the milling process is the Graftree model of the original workpiece with the volumes of previously executed tool motions removed. Verification of milling tool paths is done by a shape evaluation process which compares the machined object with the design surface at discrete points. The distance, or milling error, is then displayed by color-coded line segments on the design surface.

All of these spatial partitioning representation implementations share some common advantages such as efficient regularized Boolean set operations and realistic milling simulation. These advantages are very useful in visual detection of gross milling errors and in milling process animation. Furthermore, the volume removed at each tool motion can be easily calculated, although it is only approximate. The accuracy of the volume approximation depends on the size of the basic element, so the smaller the basic element, the more accurate the result. Despite these advantages, however, the spatial partitioning approach has failed to address dimensional milling verification capabilities of comparable complexity and accuracy as those provided by the discrete vector intersection approach.

1.2 Motivation of Research

Despite the endeavor of past research, no satisfactory implementation has emerged in the literature. The direct solid modeling approach is too computationally expensive to be practical for manufacturing applications. Consequently, this approach is not considered as a good solution to NC milling problems. The discrete vector intersection approach is capable of presenting dimensional milling error, however, its success is limited for several reasons. First, realistic milling simulation is not supported. Second, tool path modification is not developed. Such a task depends on the experience and skill of NC tool path programmers to manually correct CL data to avoid errors.

Third, the discrete vector intersection approach presents milling error on the design surfaces, not on the milled part. Thus the actual milling process can not be visualized and the relative position of the design surfaces with respect to a workpiece is difficult to locate. Consequently, the verification result can not be used as a reference for further machining processes. Finally, milling error is not the only potential problem in an NC milling program, other problems such as tool wear, material removal rate, machine dynamics analysis, etc., can not be addressed with the discrete vector intersection approach.

In contrast to the discrete vector intersection approach, the spatial partitioning approach succeeds in realistic representation of the milling process which is very practical to industry applications. The part image displayed on a computer monitor in an accurate representation of the actually milled part, thus NC programmers can have better control over the milling process and workpiece. However, verification of the milled part model against design surfaces is not well developed, so the current implementations using this approach are intended more for visual simulation than verification.

One of the reasons that the spatial partitioning approach is not well suited to NC milling verification implementation is that the display method is not versatile, i.e., is view dependent. For implementations of this approach, a workpiece is decomposed into a very large number of basic elements, thus the display task must be optimized to achieve real-time performance. Consider the dexel data structure [van Hook, 1986] for example, its image display method simplifies the task to pixel level, so displaying the dexel data structure in a fixed graphics window can be done in constant time. However, the image can not be arbitrarily rotated and thus is somewhat limited for actual manufacturing applications. Another drawback is the requirement of high memory capacity to store the basic elements. Although hardware-based system can be constructed to overcome these drawbacks [van Hook, 1986, Menon and Robinson, 1992], the cost is not acceptable for most industry end-users.

The most successful dimensional verification approach in the past research is the discrete vector intersection approach. A real-time five-axis NC milling verification implementation is reported in Chang and Goodman [1991]. However, industry applications simply need more capabilities for a NC milling verification system than the past implementations can offer. For instance, the functions of milling simulation, tool path correction, machine dynamics analysis, and tool wear prediction are serious concerns.

To develop a complete NC milling verification system that achieves all of these objectives, this research adopted the spatial partitioning approach as the basis for a comprehensive system. The primary thrust of the research is to overcome the drawbacks of the spatial partitioning approach, i.e., view-dependency and high memory capacity requirement, and incorporate the advantages found in other approaches. The goal of the research is to develop an NC milling verification system that is capable of presenting realistic simulation, dimensional verification and tool path correction. A unique five-axis tool path generation approach based on the NC milling verification system is also proposed to develop gouge- and collision-free tool paths.

1.3 Overview of Dissertation

This dissertation presents an implementation of the spatial partitioning method for five-axis NC milling simulation and describes a *discrete dexel NC milling verification* within the simulation algorithm. The discrete dexel NC milling verification algorithm is unique in several respects. First, five-axis tool paths are simulated by realistic animation for visual verification of tool paths and the volume removed by a tool motion can be computed for machine dynamic analysis. Second, color-coded information depicting ranges of milling error is displayed on the milled workpiece surface, not the design surfaces. Thus the verified milled workpiece image closely matches an actual workpiece. Third, in contrast to the discrete vector intersection approach, the design surfaces are not discretized, so the associated computation time is saved, and precise milling errors are computed efficiently. Forth, a gouge elimination algorithm that utilizes the verification result is implemented to assist tool path modification. Finally, elimination of unexpected collisions between the tool assembly and workpiece is incorporated in the verification algorithm, thus the resulting modified tool paths are practical for immediate industrial application.

Chapter Two presents the dexel representation of solid and methods of its construction and manipulation. Algorithms for optimizing the setup of the dexel coordinate system, scan converting solid geometry, and performing dexel-based Boolean set operations are also discussed. Since the spatial partitioning method decomposes a solid into a large number of elemental components, efficient and accurate visualization of the dexel representation is a critical task. Chapter Three addresses this issue by introducing an image space display method, a physical shape display

method, and a contour display method. The image space display method is similar to the one proposed in van Hook [1986] and is used for efficient graphical milling simulation. The physical shape display and contour display method are proposed for viewing-independency.

Chapter Four introduces a five-axis milling simulation technique and a dimensional verification algorithm based on the dexel representation. The algorithm simultaneously measures discrepancies and depicts the milling errors on the milled surface during a milling simulation. Realistic in-process verification is presented. To graphically encode and decode milling errors, a color lookup table is built such that every color in the table represents a range of cut depth values. Hence the verification result clearly reveals the tool path accuracy. To assist the tool path modification process, an automatic tool path correction technique including gouge- and collision-elimination subtasks is developed in Chapter Five.

Chapter Six presents a software implementation of the NC milling verification system, in which several experimental examples are conducted and the software performance is summarized. Finally, Chapter Seven concludes the dissertation and discusses the future research opportunities.

2. DEXEL REPRESENTATION OF SOLID GEOMETRY

A dexel representation derived from the dexel data structure [van Hook, 1986] is introduced which utilizes rectangular solid elements to approximate free-form solid geometry. The dexel representation of a solid is constructed in a dexel coordinate system via a scan conversion process, and, furthermore, is manipulated by using dexel-based Boolean set operations. Since approximation accuracy of the dexel representation heavily relays on the setup of the dexel coordinates, an algorithm is designed to automatically establish the dexel coordinates based on a user-specified tolerance to increase the overall accuracy.

2.1 Dexel Coordinate System

The dexel coordinate system is defined by an origin point $\boldsymbol{O}$, a depth vector $\boldsymbol{v}_d$, and an orientation vector $\boldsymbol{v}_o$ in the world coordinate system. Basis vectors of the dexel coordinate system are given by,

$$\begin{bmatrix} \boldsymbol{v}_x \\ \boldsymbol{v}_y \\ \boldsymbol{v}_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_d \times \boldsymbol{v}_o \\ \boldsymbol{v}_x \times \boldsymbol{v}_d \\ \boldsymbol{v}_d \end{bmatrix} \tag{2.1}$$

The vectors $\boldsymbol{v}_d$ and $\boldsymbol{v}_o$ are analogous to the vectors typically required to define a viewing transformation in computer graphics applications, i.e., the viewing direction and the view-up vector, respectively [Foley et al., 1990]. In van Hook's dexel data structure, the depth vector $\boldsymbol{v}_d$ was limited to the viewing direction, thus the view is fixed once dexel data structure has been constructed. In this implementation of the dexel representation, however, information and properties of each dexel are stored relative to an independent dexel coordinate system and can be transformed into either the world coordinate system or the screen coordinate system via coordinate transformations. Thus the dexel representation is view-independent and more general than the original definition.

Let $\boldsymbol{i}_1$, $\boldsymbol{i}_2$ and $\boldsymbol{i}_3$ be the unit vectors in the direction of $\boldsymbol{v}_x$, $\boldsymbol{v}_y$ and $\boldsymbol{v}_z$, respectively. The transformation between the dexel coordinate system and the world coordinate system is given by,

$$\boldsymbol{P}^w = \begin{bmatrix} \boldsymbol{i}_{1x} & \boldsymbol{i}_{2x} & \boldsymbol{i}_{3x} & \boldsymbol{O}_x \\ \boldsymbol{i}_{1y} & \boldsymbol{i}_{2y} & \boldsymbol{i}_{3y} & \boldsymbol{O}_y \\ \boldsymbol{i}_{1z} & \boldsymbol{i}_{2z} & \boldsymbol{i}_{3z} & \boldsymbol{O}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \boldsymbol{P}^d = \boldsymbol{M} \cdot \boldsymbol{P}^d \tag{2.2}$$

where $\boldsymbol{P}^d$ and $\boldsymbol{P}^w$ denote homogeneous point coordinates in the dexel and the world coordinate systems, respectively, and $\boldsymbol{O}$ represents the world space location of the dexel coordinate system origin. The inverse form of Equation (2.2),

$$\boldsymbol{P}^d = \boldsymbol{M}^{-1} \cdot \boldsymbol{P}^w \tag{2.3}$$

is used to transform points in the world coordinate system to the dexel coordinate system.

Dexel locations are referenced by a two-dimensional grid in the xy-plane of the dexel coordinate system, called the *dexel plane*, and each grid point is addressed by an integer pair, e.g., (I_x, I_y) as illustrated in Figure 2.1. Assuming the grid points are equally spaced along x- and y-axis by a distance w, the dexel coordinate values of each grid point are computed by $(I_x w, I_y w)$. Since each dexel is physically a rectangular solid located on a dexel plane grid point extending along the z-axis of the dexel coordinate system, the length of a dexel is determined by a z-depth pair (z_n, z_f), where the subscripts n and f denote near and far z values, respectively.

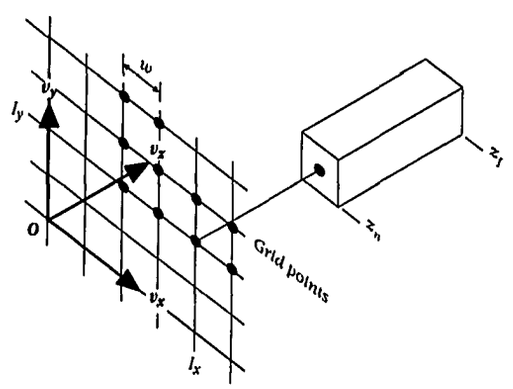
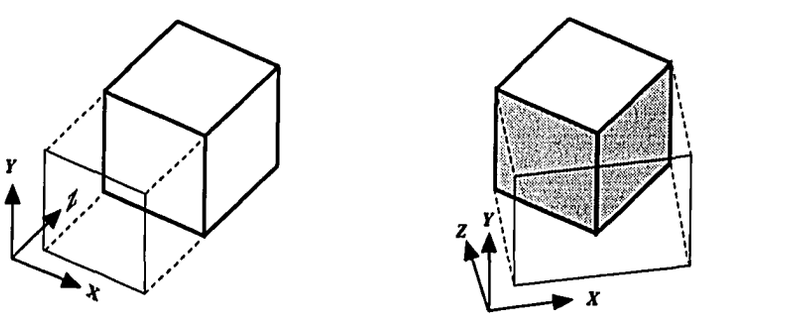


Figure 2.1 Dexel coordinate system

2.2 Setup of Dexel Coordinate System for Specific Resolution

A dexel coordinate system can be established by arbitrarily selecting the depth vector and orientation vector. In a typical software implementation, it is not necessary to specify the value of each coordinate of the vectors; an effective graphical user interface can be used to rotate, translate, and scale the solid or wireframe image of the workpiece until the area of interest is exposed to the user. Such a setup process is generally sufficient for experienced NC programmers who know where milling problems may occur on the design surface or where the essential areas of a part may be. Alternatively, computational algorithms can be implemented to automate the setup process. Furthermore, since the dexel coordinate system is independent of the screen coordinate system, the viewing direction can be chosen arbitrarily by the viewer, while the underlying dexel coordinate system is transparently generated by computational algorithms.

To assist the process, an algorithm is developed to automatically select a dexel coordinate system for near-optimal dimensional resolution, which thus enhances the accuracy of surface approximation. Consider a cube, for example, in which the maximum dimension of a dexel is exactly a side of the cube, as shown in Figure 2.2(a). Since all faces of a cube are identical, there are six orientations of dexel coordinate system along which the cube can be precisely represented. In typical application, however, some areas of the model are more important than others, and thus motivate a more general orientation. For example, assuming two faces of the cube are emphasized, as the shaded areas shown in Figure 2.2(b), an alternative depth vector may be generated as the summation of the normal



(a) Dexel coordinate system setup for a cube (b) Setup for a cube with weighted areas

Figure 2.2 Examples of dexel coordinate system of a cube

vectors of the emphasized planes. In addition, it is desirable to make as many edges parallel to the dexel plane x- or y-axis as possible via the selection of an orientation vector to reduce the error of edge approximation.

The automatic setup process of a dexel coordinate system includes several steps: first, determine a depth vector; second, calculate an orientation vector; and finally find the minimum resolution of the dexel grid according to a specified accuracy requirement. Note that the main goal of the algorithm is to find a near-optimal setup in a cost-effective way, not necessarily the optimal coordinate system.

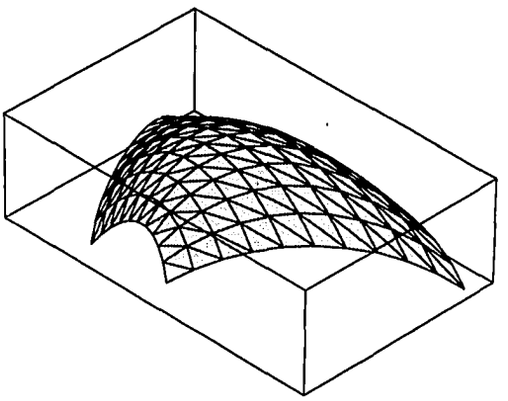
A *weighted normal summation* algorithm is developed to find an appropriate depth vector of a dexel coordinate system. This algorithm starts by triangulating the design surfaces based on the surface curvature. Then, two weights, *area* and *priority*, are assigned to each triangle for a final weighted depth vector. The area weight enables normals corresponding to larger triangles to be dominant in the generation of the depth vector. The priority weight is user-specified and applied to emphasize certain areas and thus apply more dexels per unit area.

The weighted vector summation approach is, in essence, the sum of all scaled normal vectors to generate a depth vector of the dexel coordinate system, i.e.,

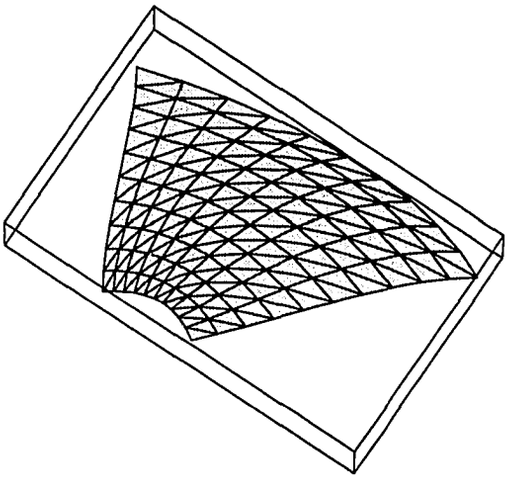
$$\boldsymbol{v}_d = \sum_{i=1}^{m} A_i P_i \boldsymbol{n}_i \tag{2.4}$$

where the surfaces are discretized into m triangles and A, P, and $\boldsymbol{n}$ denote the area, priority weight, and unit normal vector, respectively. If the summation generates a zero vector, the normal vector with the maximum weight is used. Applying this algorithm to the cube example generates a zero vector if no priority weight is given, thus any of the six normal vectors can be used as the depth vector. An application of the algorithm to a bi-cubic Bézier surface is show in Figure 2.3(b).

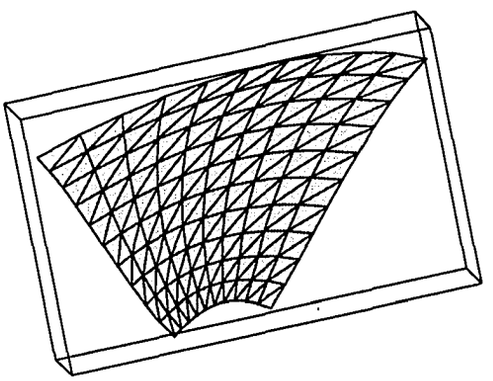
After the depth vector is selected, the orientation vector, which determines the accuracy of edge approximations, must be determined to define a unique dexel coordinate system. Again, consider the cube in Figure 2.2 for example. A good orientation aligns its edges with the dexel coordinates, as shown in Figure 2.2(a), thus the cube can be exactly represented by dexels. To find an orientation vector to suit this purpose, a *least bounding area* algorithm is developed. This algorithm computes a bounding rectangle on the dexel plane that completely contains



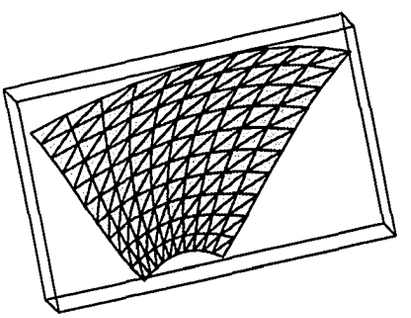
(a) Initial orientation of a surface and its bounding box



(b) Surface oriented along a computed depth vector



(c) Orienting surface in (b) using an orientation vector generated from the least bounding area algorithm



(d) Surface resized according to the dexel size computed by using Equation (2.15)

Figure 2.3 An example of weighted normal summation algorithm

the projection area of the design surfaces and has a minimum area. Since the orientation is independent of the depth vector (z-axis), the projected area of the design surfaces on the dexel plane along the depth-axis is fixed for any orientation vector. Consequently, the smaller the bounding rectangle area, the better the dexel representation approximates the design surfaces.

The least bounding area algorithm is implemented using a sampling method, i.e., sequentially rotating the projected surface by a small increment and finding the minimum bounding rectangle area among the sampled

orientations. The necessary range to be sampled is 90° since the bounding area, denoted by $A(\theta)$, is identical to $A(\theta+90)$, $A(\theta+180)$, and $A(\theta+270)$.

The minimum area is computed by,

$$A(\theta) = \left| max\left(\begin{bmatrix} V_{i_x} \\ V_{i_y} \end{bmatrix}\right) - min\left(\begin{bmatrix} V_{i_x} \\ V_{i_y} \end{bmatrix}\right) \right|_j \quad , i = 0 \ldots m,\ j = 0 \ldots n \tag{2.5}$$

where V_i dentes the i-th vertex of the triangular mesh, j denotes the j-th rotational sampling step, m is the number of vertices, and n is the steps of sampling. An alternative approach for choosing a dexel orientation is to use optimization methods, however, since the discretization of surfaces into triangles limits the accuracy of $A(\theta)$, the generated optimal solution is generally no better than the least bounding area algorithm. Furthermore, since the dexel grid has finite resolution, a near-optimal solution is generally sufficient.

After the dexel basis is established, the dimension of the uniform dexel face, i.e., w, as shown in Figure 2.1, must be determined. The resolution of the dexel coordinate system dominates the accuracy of approximation along the depth axis. Denoting a specified accuracy value by t and a radius of surface curvature by r, the local approximation error E is given by,

$$E = r - \sqrt{r^2 - \frac{w^2}{4}} \leq t \tag{2.6}$$

as illustrated in Figure 2.4. Equation (2.6) is used to find the approximation error for a given dexel resolution and a local radius of curvature. Rearranging Equation (2.6) yields,

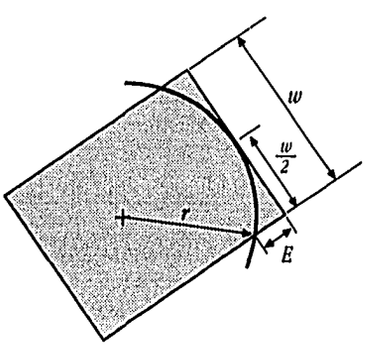


Figure 2.4 Estimation of approximation error

$$w = 2\sqrt{2Er - E^2} \tag{2.7}$$

which is used to compute a dexel resolution to satisfy a given approximation error and a given global minimum radius of curvature. The global minimum radius of curvature is generally difficult to compute, if not impossible. Even worse, for surfaces having C^0 continuity the radius may approach to zero. Therefore, a practical estimation is to use the tool radius for the radius of curvature in Equation (2.6) and Equation (2.7) to evaluate approximation error and dexel size.

2.3 Scan Conversion for Dexel Models

A scan conversion process is used to convert solid objects into a dexel representation. Parallel rays are fired from a sub-set of grid points on the dexel plane to intersect solid objects and dexels are formed from the segments of rays that are in the interior of the objects. Finally, a set of dexels is generated to represent the object. An example of scan conversion of a tetrahedron is shown in Figure 2.5, where the rays are parallel to the z-axis and the grid points for scan conversion are limited by a bounding rectangle containing the projected area of the tetrahedron on the dexel plane.

During the scan conversion process, the near- and far-depth of each dexel are generated, as well as surface outward normal vectors at the corresponding ends of each dexel. These surface normal vectors are utilized to

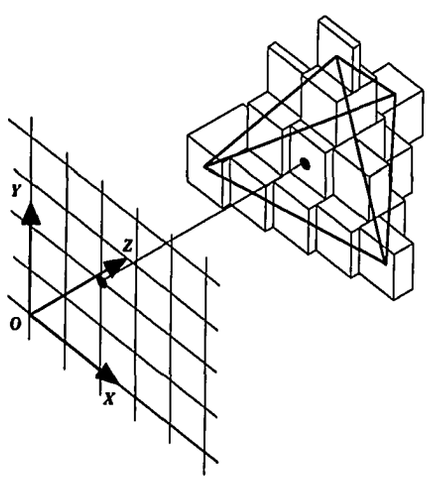


Figure 2.5 Dexel representation of a tetrahedron

determine the shade of the front and back faces of a dexel. Furthermore, since a solid object has properties, for example, color, material type, function, etc., these properties may also be recorded in the dexel representation.

The dexel data type, as illustrated in Figure 2.6, is implemented in the *C* language by the data structure,

```
typedef struct {
    float near_z, far_z;
    int near_color, far_color;
    long next_dexel;
    int type, status;
} dexel;
```

where *near_z* and *far_z* are the near and far depth values of a dexel; *near_color* and *far_color* are dot products of near- and far-normal vectors and a lighting vector $\boldsymbol{v}_l$ originating from an infinite light source, respectively; *next_dexel* records the index of the next dexel, or null if no dexel follows; *type* is used to denote the owner of a dexel, e.g., cutter, workpiece, fixture, etc. Finally, *status* is used by the software implementation and is discussed in Chapter seven.

Objects that are involved in a milling process are typically the workpiece, milling tool, and fixtures. The objects can be represented by several basic primitives including blocks, spheres, cylinders, etc. Scan conversion of these primitives are based on algorithms of ray/surface intersection. The following section briefly discusses these algorithms.

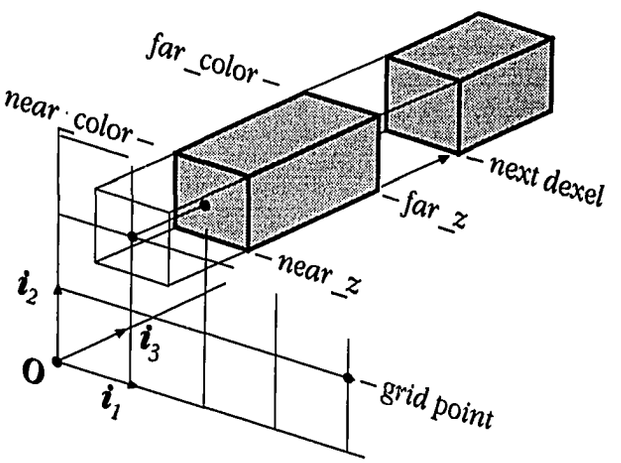


Figure 2.6 Dexel data structure

2.3.1 Scan conversion of blocks

A block is bounded by six planes. Thus scan conversion of block primitives amounts to construct the dexels that are contained by the planes. Consider plane represented in the parametric form,

$$\boldsymbol{F}(u,v) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q}_x + \boldsymbol{a}_x u + \boldsymbol{b}_x v \\ \boldsymbol{Q}_y + \boldsymbol{a}_y u + \boldsymbol{b}_y v \\ \boldsymbol{Q}_z + \boldsymbol{a}_z u + \boldsymbol{b}_z v \end{bmatrix} \tag{2.8}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$, as illustrated in Figure 2.7, are vectors parallel to each side of the bounded plane which originate at a corner point $\boldsymbol{Q}$. The dimension, width and height, of the bounded plane are specified by the magnitude of $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively. Hence the interior and boundary points of the plane are defined by a pair of parameters u and v within the range of [0,1].

Defining a ray $\boldsymbol{R}$ by a point $\boldsymbol{P}$ and a unit vector $\boldsymbol{c}$, the parametric formulation of the ray is given by,

$$\boldsymbol{R}(w) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \boldsymbol{P}_x + \boldsymbol{c}_x t \\ \boldsymbol{P}_y + \boldsymbol{c}_y t \\ \boldsymbol{P}_z + \boldsymbol{c}_z t \end{bmatrix} \tag{2.9}$$

The intersection point of a plane $\boldsymbol{P}$ and a ray $\boldsymbol{R}$ exists at,

$$\begin{cases} \boldsymbol{a}_x u + \boldsymbol{b}_x v - \boldsymbol{c}_x t + \boldsymbol{Q}_x - \boldsymbol{P}_x = 0 \\ \boldsymbol{a}_y u + \boldsymbol{b}_y v - \boldsymbol{c}_y t + \boldsymbol{Q}_y - \boldsymbol{P}_y = 0 \\ \boldsymbol{a}_z u + \boldsymbol{b}_z v - \boldsymbol{c}_z t + \boldsymbol{Q}_z - \boldsymbol{P}_z = 0 \end{cases} \tag{2.10}$$

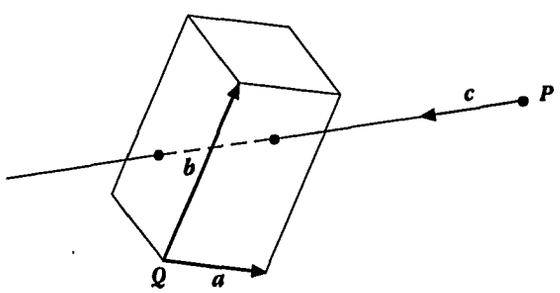


Figure 2.7 Intersection of a ray and a block

where the values of u, v, and t are solved directly. The values of u and v must be within [0,1] to be on the bounded plane and the value of t indicates the signed distance from $\boldsymbol{P}$ to the plane. Note that if the ray is perpendicular to the plane normal, there is no solution.

A block is a convex object, so a ray, if not perpendicular to any plane normal of the block, has at most two intersection points with the block. Therefore, the algorithm for scan conversion of a block searches for intersection points between a ray and the six bounded planes, but stops when two intersection points are found. Furthermore, if no intersection point is found for five planes, there is no need to intersect the sixth.

The solution of each ray intersection is saved in an 2-D array of dexels for the dexel representation of the block. Denoting the t parameters of the intersection points by t_1 and t_2, the smaller is saved in *near_z*, and the other in *far_z*. The dot products of normal vectors of the near and far intersected planes and the lighting vector $\boldsymbol{v}_l$ determine the values of *near_color* and *far_color*, respectively. The *next_dexel* is null because the block is convex, and *type* is given to identify the dexel.

2.3.2 Scan conversion of spheres

Denoting the center of a sphere by $\boldsymbol{O}$, the minimum distance d, as illustrated in Figure 2.8, between point $\boldsymbol{O}$ and ray $\boldsymbol{R}$ determines whether the ray intersects the sphere. Using the formulation of a ray in Equation (2.9), the distance is computed by,

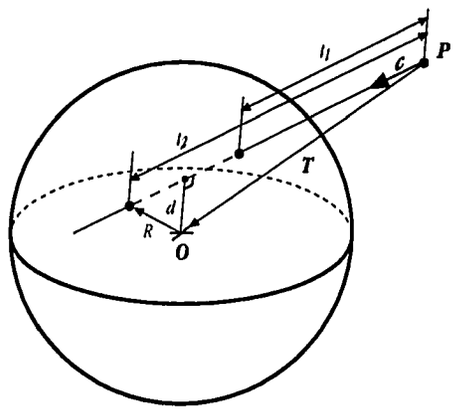


Figure 2.8 Intersection of a ray and a sphere

$$d = \sqrt{|T|^2 - |T \cdot c|^2} \tag{2.11}$$

where T is the vector between P and O. If d is greater than the radius of the sphere, denoted by R, there is no solution for intersection. If d is equal to R, within a specified tolerance, no solution is also concluded. In the other case, the depths of near and far intersection points are computed by,

$$t_n, t_f = T \cdot c \pm \sqrt{R^2 - d^2} \tag{2.12}$$

thus t_n and t_f define the dimension of a dexel. The unit normal vector at each intersection point is determined by the vector connecting the center O and the intersection points, i.e.,

$$n_n, n_f = \frac{(P + (t_1, t_2)\, c) - O}{R} \tag{2.13}$$

n_n and n_f are then used to determine the value of *near_color* and *far_color*, respectively. The *next_dexel* is null because a sphere is convex.

2.3.3 Scan conversion of cylinders

A bounding ellipse scan conversion algorithm is implemented to efficiently construct the dexel representation of an arbitrarily oriented cylinder. This algorithm scan converts only a single cross-section of the cylinder, then offsets this ellipse-shaped cross-section to form other portions of the cylinder. The ellipse cross-section is computed parallel to either the x- or the y-axis depending on the projection of cylinder axis, i.e., if its projection onto the x-axis is greater than its projection on the y-axis, the ellipse cross-section is computed on a constant x plane (x-scan), otherwise, on a constant y plane (y-scan). Since the algorithm is capable of adjusting scanning direction according to the orientation of cylinder axis, the only special case is when the cylinder axis is parallel to the z-axis; the scan conversion process is thus simplified to perform intersections with two parallel circular disks.

Assuming y-scan is employed for a scanning conversion process (x-scan is similar), as illustrated in Figure 2.9, the ellipse cross-section is constructed along the lowest scan line of the bounded cylinder. Rays fired from the y-

scan line intersect the infinite cylinder defined by radius R, axis $\boldsymbol{e}$, and axis point $\boldsymbol{Q}$. The intersection points of the rays thus construct the dexel representation of a ellipse disk lying on the xz plane. The vector between each intersection point and $\boldsymbol{Q}$ is also projected to the cylinder axis to compute the axial projection distance, i.e., a on Figure 2.9, which is used to determine if the intersection point is on the surface of the bounded cylinder.

To find the intersection points between each ray and the infinite cylinder, the minimum distance d between the axis and the ray, as illustrated in Figure 2.10, is first computed by,

$$d = (\boldsymbol{P}-\boldsymbol{Q}) \cdot \frac{\boldsymbol{e}\times\boldsymbol{c}}{|\boldsymbol{e}\times\boldsymbol{c}|} = (\boldsymbol{P}-\boldsymbol{Q}) \cdot \boldsymbol{N} \tag{2.14}$$

Note that the cross product of $\boldsymbol{e}$ and $\boldsymbol{c}$ can not be zero because they are checked for parallelism prior to the calculation. If d is larger than the radius of the cylinder R, no intersection can be found. If d is equal to R, two identical

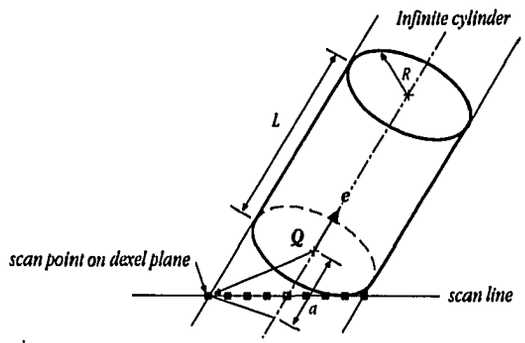


Figure 2.9 Scan conversion of cylinder using a bounding ellipse algorithm

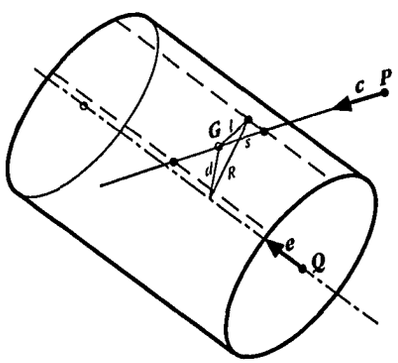


Figure 2.10 Intersection of a ray and a cylinder

solutions exist, however, no dexel is created for zero depth. Finally, if d is less than R, two intersection points can be found. Denoting the nearest point on the ray to the cylinder axis by $\boldsymbol{G}$, the distance between $\boldsymbol{G}$ and either intersection point is given by,

$$s = \frac{\sqrt{(R^2 - d^2)}}{\sin\phi} \tag{2.15}$$

where ϕ is the angle between $\boldsymbol{e}$ and $\boldsymbol{c}$. The intersection points are then computed by $\boldsymbol{G} \pm s\boldsymbol{c}$ and are used to construct a dexel.

After the dexel representation of the ellipse disk and the associated axial projection distances are computed, copies of the ellipse are offset and stacked up to form the dexel representation of the cylinder. Denoting the projection of a unit tool axis on the dexel coordinates by t_x, t_y, and t_z, respectively, the address of each dexel of the ellipse disk of the i-th copy along the y-axis is offset by,

$$(\Delta x, \Delta y) = \left(\frac{i\ t_x}{t_y}, i\right) \tag{2.16}$$

as illustrated in Figure 2.11. However, since only integer values are allowed to address a grid point, Δx is rounded to a lower integer value. Consequently, the depth value must be corrected to avoid errors due to the address shift. Assuming $\Delta x = \Delta x_i + \Delta x_d$, where subscripts i and d denotes the integer part and the decimal part of Δx, respectively, (and Δx_d is positive), the offset amount of Δz_j for the j-th dexel on the ellipse disk is thus computed by using a linear interpolation,

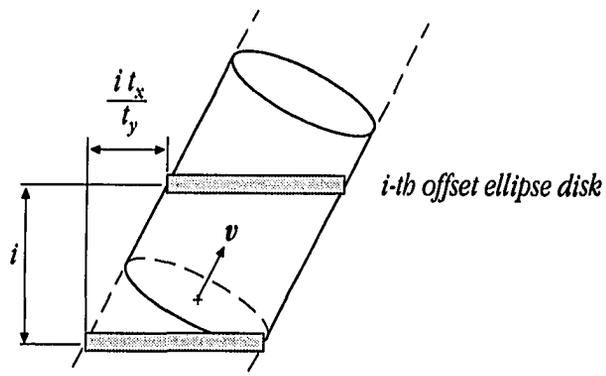


Figure 2.11 Offset ellipse disk

$$\Delta z_j = \frac{it_z}{t_y} w + (z_j - z_{j-1})\,\Delta x_d \tag{2.17}$$

where subscript n and f denote the *near-* and *far*-depth, respectively. Similarly, quadratic interpolation scheme can also be used to find the depth correction.

Since a cylinder is bounded, several cases may occur during the stacking process. Figure 2.12 shows a "side-view" of several individual dexels contained between ellipse disks at various intersections with a cylinder. In Case 1, the axial projection of both near and far ends of the dexel are outside the cylinder, i.e., out of the range $[0, L]$, where L is the length of the cylinder. Hence there is no dexel for the cylinder at this location. In Case 2, only one side of the dexel is on the bounded cylinder, thus the dexel is truncated. The detailed illustration of Case 2 is shown in Case 2-a. The length to be truncated, denoted by t, is computed by,

$$t = \frac{(z_f - z_n)\,(a_n - L)}{a_n - a_f} \tag{2.18}$$

where a_n and a_f are the axial projection distances of the near and far side, respectively, and z_n and z_f are the depth values of the dexel. Hence the new near depth is $z_n + t$ and the associated normal vector is the tool axis. In Case 3, the dexel is completely inside of the cylinder, so no further computation is needed. Finally, in Case 4, the dexel passes the cylinder ends at both sides, thus both ends of the dexel are truncated in a fashion similar to Case 2-a.

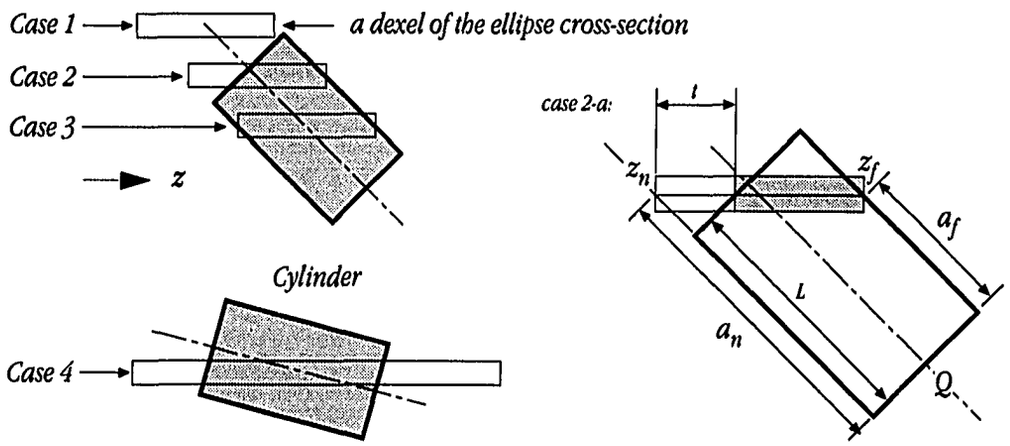


Figure 2.12 Cases of stacking process of cylinder scan conversion

2.4 Regularized Boolean Set Operations on Dexel Models

Regularized Boolean set operations, including union, intersection, and difference, are easily implemented for the dexel representation. Since dexel faces are uniform, the operations are a one-dimensional depth comparison processes. The union operation ($\cup$) either merges two intersecting dexels together if the depths overlap or, if the depth ranges are separated, constructs a link between these two dexels, called a *dexel chain*, as illustrated in Case 2 of Figure 2.13. The intersection operation ($\cap$) forms a dexel that commonly belongs to two dexels, i.e., the overlapping range, or generates *null* if there is no intersection. The difference operation (-) removes the intersecting portion from the dexel to be subtracted.

Denoting the dexels for an operation by A and B, and the resulting dexel by C, algorithms of dexel Boolean operations written in pseudocode is as follows:

```
Boolean_union(A, B, C)
{
```

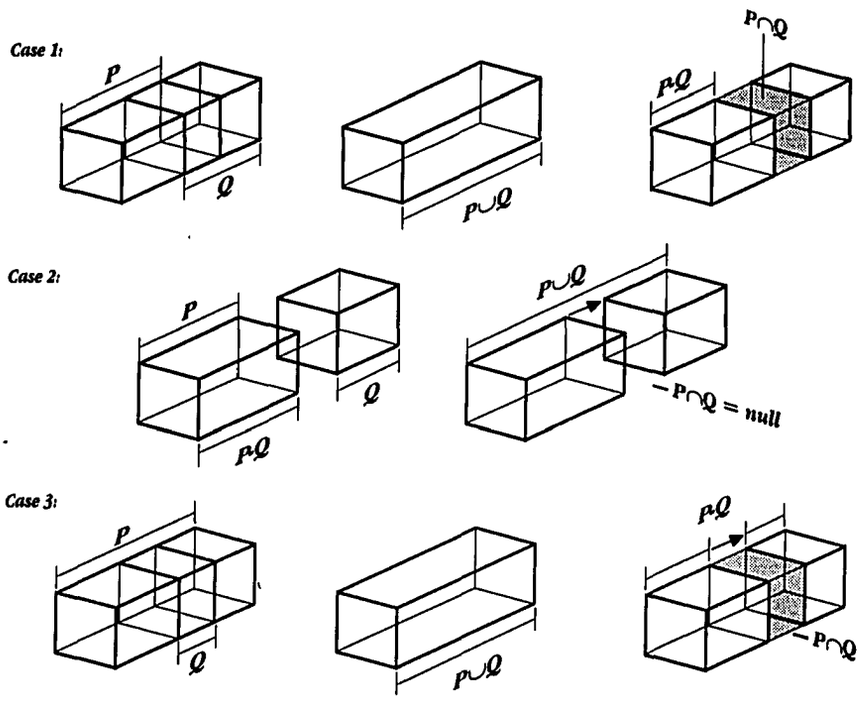


Figure 2.13 Illustration of Boolean operations

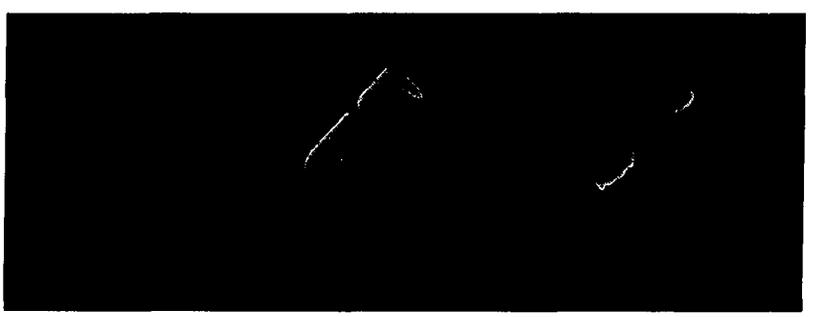
```
    if(A.near>B.far) C=B→A;                                  /* A and B are separated */
    else if(A.far>B.near) C=A→B;
    else {
        C.near=(A.near<B.near)?A.near:B.near;
        C.far=(A.far>B.far)?A.far:B.far;
    }
}
Boolean_intersection(A,B,C)
{
    C.near=(A.near>B.near)?A.near:B.near;
    C.far=(A.far<B.far)?A.far:B.far;
    if(C.near>=C.far) C=Φ;                                   /* A and B do not intersect */
}
Boolean_difference(A,B,C)                                    /* C=A-B */
{
    if(A.near>B.far || A.far<B.near) C=A;                    /* A and B do not intersect */
    else {
        if(A.near<B.near) {
            C.near=A.near;
            if(A.far<B.far) {                                /* A is divided into two pieces by B */
                C.far=B.near;
                D.near=B.far;
                D.far=A.far;
                C=C→D;
            }
            else C.far=B.near                                /* C is a front portion of A */
        }
        else {
            if(A.far<=B.far) C=Φ;                            /* A is completely inside of B */
            else {
                C.near=B.far;                                /* C is a rear portion of A */
                C.far=A.far;
            }
        }
    }
}
```

The validity of these regularized Boolean set operations are obvious, so no proofs are provided. The time complexity is $O(1)^{\dagger}$ because the basic operations such as comparing and assigning values can be done in constant time.

These regularized Boolean set operations are much simpler than those used in the CSG and B-rep solid modeling systems. Hence makes the dexel representation in computationally faster than CSG and B-rep in schemes for manipulation and image rendering. Furthermore, the results of operations in the dexel representation are guaranteed to be valid because all dexels have positive depth lengths.

Dexels generated from the regularized Boolean set operations, called the *child-dexels*, inherit properties of dexels that are involved in the operations, called the *parent-dexels*. These properties include *near_color*, *far_color*, and *type*. For example, in Case 1 of Figure 2.13, the new dexel *P-Q* will have *near_color* from *P*'s, *far_color* from *Q*'s *near_color*, and *type* from *P*'s. Similar rules are applicable to other operations. However, intersecting two dexels of different type, the type of the child-dexel is ambiguous. To eliminate the ambiguity, a priority index is assigned to each dexel type. Thus the type of the higher priority parent is given to the child-dexel.

An example of the dexel-based Boolean set operations is given in Figure 2.14. A union operation is performed for constructing the inverted T-shaped solid in Figure 2.14(a). The union of the inverted T-shaped solid and a ball-end milling tool is shown in Figure 2.14(b). Figure 2.14(c) shows a result of regularized Boolean difference operations which will be used intensively in graphical NC milling simulation applications.



(a) (b) (c)

Figure 2.14 Shaded image illustration of Boolean operations

---

† The ***O***-notation denotes the upper bound of a function. A function $g(n)$ is $O(f(n))$ for another function $f(n)$, if there exist constants c and N, such that, for all $n \geq N$, $g(n) \leq cf(n)$.
The ***O***-notation is used to analyze the worst case time complexity (running time) of an algorithm, e.g., $O(1)$ denotes an algorithm can be done in constant time, $O(n)$ in linear time with respect to the input size n, etc.

3. VISUALIZATION OF A DEXEL REPRESENTATION

The image-space display method [van Hook, 1986] is the most efficient way of visualizing dexel-based objects. However, because dexel coordinates are aligned with the screen coordinates, only the front and the back views can be displayed once the dexel representation is constructed. To visualize dexel-based objects in other viewing directions, the entire dexel representation needs to be reconstructed, which severely limits applications in manufacturing and engineering modeling. Therefore, a *physical shape display* method and a *contour display* method are introduced to overcome the limitation and achieve view-independency of dexel representations.

3.1 Image Space Display Method

The image space display method aligns the depth vector of a dexel coordinate system with the viewing vector of the screen coordinate system, and thus each grid point on the dexel plane corresponds precisely to a certain number of pixels of the display screen. Figure 3.1 demonstrates a one-to-one mapping of this visualization method. Since the depth vector of the dexel coordinate system is parallel to the viewing direction, only the nearest face of a dexel chain at each grid point is visible. The *near_color* of the nearest dexel at each pixel is directly mapped

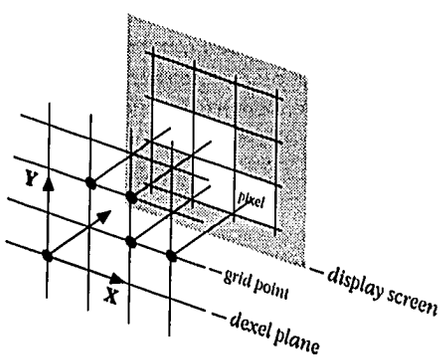


Figure 3.1 Image-space display method

to the video buffer that controls the screen display, therefore, this method is very efficient and needs no complicated graphics functions to achieve high quality animation. If no dexel information is stored at a pixel location, a background color is used. Hence, it is viable to use a photograph as the background to create a realistic scene of animation without resorting to 3D geometric modeling.

The parallel property of the depth vector and viewing vector limits the transformation capability of this display method, i.e., object images are only translatable but not arbitrarily rotatable. Consequently, to look at dexel-based objects from other viewing directions than the front and the back of the dexels, the entire dexel data structure must be re-construction[van Hook, 1986]. This limitation is not a severe drawback in image rendering applications. However, most manufacturing and engineering applications require intensive transformations of a model. Consider the NC milling verification application, for example. Finished parts need to be examined closely to detect every possible problem. This requirement motivates the development of a physical shape display method and a contour display method which do not require re-construction of dexel representation to perform general transformations.

3.2 Physical Shape Display Method

Dexels are physically blocks with uniform faces and variable depths. Their orientation and locations in the world coordinate system are well defined by the depth vector and the transformation matrix in Equation (2.2). Therefore, the dexel representation can be display as a group of blocks in the world coordinate system. This visualization method is similar to those used for displaying voxels and octrees data structures[Samey, 1989]. Consider visualization of octrees for example. One approach is to display the farthest octant first, then the three adjacent octants that share a face with the farthest octant in any order. As each octant is displayed, its descendents are displayed recursively in a depth-first fashion. Since no node in this enumeration can obscure any node enumerated after it, a z-buffer algorithm is not required. Furthermore, because each node is a cube, at most three of its faces are visible and necessary for display.

Dexels can be visualized in a similar fashion. The display sequence starts from the deepest grid point in the z-axis of the screen coordinate system, as illustrated in Figure 3.2. Dexels in a dexel chain are displayed in a back-to-

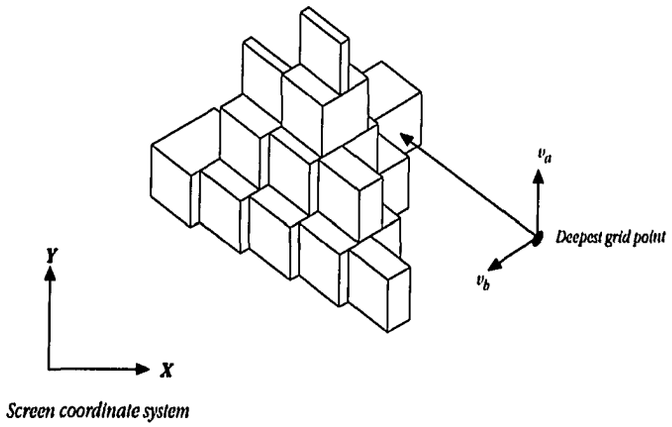


Figure 3.2 Visualization of dexel representation by the physical shape of dexels

front fashion. After dexels at a gird point are displayed, the two adjacent grid points are recursively processed in the same fashion. This visualization method shares similar characteristics with the octree method, i.e., at most three faces are visible, a depth-first implementation, and no need for a z-buffer algorithm. However, this display method is time-consuming when the number of dexels is large.

3.3 Contour Display Method

The contour display method generates contours that connect the dexel points (center points of the front and the back dexel faces) along constant x or y grid points, thus a set of equally spaced contours are displayed to represent the geometry of dexels. The main goal of the visualization method is to present an efficient display method to access the information on every portion of the dexel-based object. Furthermore, the generated contours can be used to construct a triangular mesh for more realistic dexel visualization [Ekoule et al., 1991, Meyers and Skinner, 1992].

Assuming constant y contours are to be generated, the contour display method first selects a starting grid point that has the smallest x coordinate value, then sequentially traverses through all dexels in the constant y grid points. The basic rule of contour traversal is, from the current dexel point, step to the next higher grid point in the x-

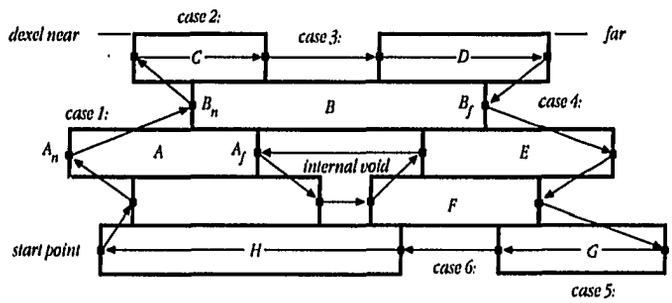


Figure 3.3 Cases of dexel point connection

direction if the current dexel point is in the near side, otherwise, step to the next lower one. Several cases of dexel point connection, as illustrated in Figure 3.3, are classified in the contour generation process.

In Case 1, dexels A and B overlap and thus the next dexel point to be connected from A_n is the near dexel point in next higher grid point, i.e., B_n. Case 4 is similar to Case 1 except it handles the dexel points at the far side, thus the far dexel point of the deepest overlapping dexel at the next lower grid point is connected, i.e., B_f is connected to E_f. In Case 2, there is no dexel at the next higher grid point, thus the far dexel point is connected, i.e., C_n is connected to C_f. Similarly, Case 5 handles the far dexel points. In Case 6 there is a dexel, F, at the next higher grid point which overlaps the current dexel, G, however, it also overlaps with the dexel in front of G, i.e., H, therefore, the far side of dexel H is connected. Similarly, Case 3 is for far dexel points.

The time complexity of the contour generation algorithm is dominated by the length of dexel chain, denoted by n_l, and the number of dexels in constant y-grid points, denoted by n_d. To determine connection type (case), $\boldsymbol{O}(n_l)$ time is needed. Such computation is needed at every instance of traversing all dexels thus $\boldsymbol{O}(n_l n_d)$ is expected for the contour generation algorithm. Typically, for milled parts, n_l is far less than n_d, so generating all contours of a dexel model is dominated by $\boldsymbol{O}(n)$ time, where n is the number of dexels in the model.

Only six cases can occur during contour generation, however, there can be internal voids that do not connect to any outer contour. The internal void is discovered via a scanning process that checks if any dexel in a constant y grid points is not visited twice after a contour is generated. The first discovered dexel point that is not visited twice in a previous contour generation is used as a starting point. Hence all external and internal contours can be generated recursively.

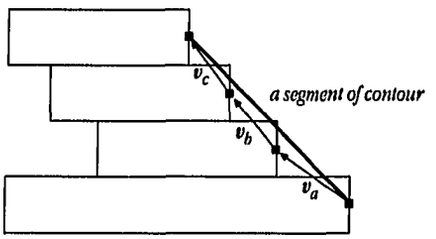


Figure 3.4 Elimination of linearly spaced dexel points

A typical dexel representation of solid geometry can easily generate millions of dexel points on the contours, however, not all dexel points should necessarily to be included in the contours., e.g., dexel points that are of the same color and linearly spaced, as shown in Figure 3.4. Such intermediate dexel points are eliminated. A tolerance range [-ε,ε] is given for linearity checking, i.e., $|v_a \times v_b| < \varepsilon$ and $|v_a \times v_c| < \varepsilon$, thus vectors within the tolerance range are considered parallel. This dexel point elimination process can reduce, depending on the complexity of the geometry, a certain amount of unnecessary dexel points in the contours. Therefore, the graphic display can be more efficient and easier to manipulate. Figure 3.5 demonstrates an example of the contour display method in which 71% of the original 195,767 dexel points are eliminated. The original image is displayed at the top left corner.

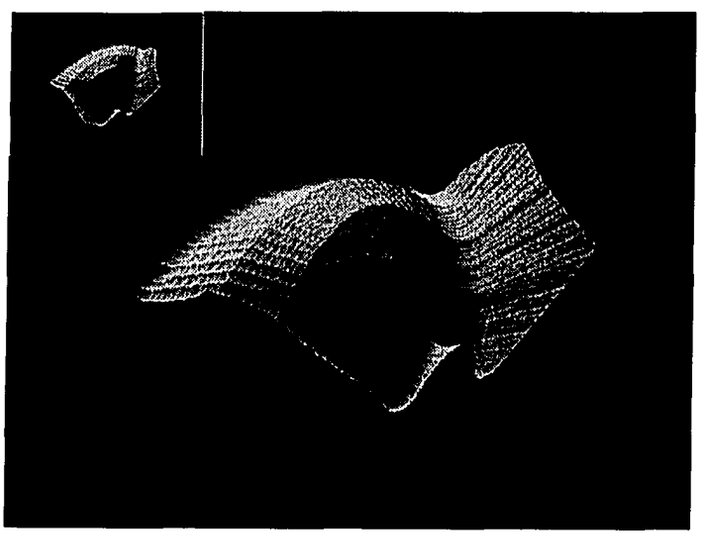


Figure 3.5 Contour display of dexel representation

4. FIVE-AXIS NC MILLING SIMULATION AND VERIFICATION

NC milling is an expensive process, in terms of the machine cost, labor, and material, so simulation of tool paths is usually performed prior to actual milling to visually detect potential problems. The objective of analytical milling verification methods is to analyze tool paths in greater details, i.e., not only depicting the location of error, but also the dimension. However, none of existing analytical methods can perform NC simulation and dimensional verification simultaneously. To achieve the objective, a *discrete dexel NC verification* method is implemented. This method is based on the dexel representation of solid geometry to exploit the respective advantages of the normal vector intersection and spatial partitioning representation approaches. The resulting system provides a realistic milling simulation with color-coded dimensional verification results displayed on the part surface.

4.1 NC Milling Simulation

The NC milling simulation algorithm utilizes the dexel representation of solid geometry to model cutters, workpiece stock and fixtures in a milling setup. It applies regularized Boolean difference operations to simulate the material removal phenomena between a moving tool and workpiece during the milling process. The moving tool is controlled by an *instances of motion* approach which successively updates the workpiece model, thus the computational expense for scan converting a swept volume is eliminated without loosing accuracy. Since the dexel representation is highly organized data structure and the regularized Boolean set operations are efficient, real-time performance is achieved in simulating the NC milling process.

4.1.1 Instances of motion

The instances of motion approach approximates tool swept volumes to dexel resolution, therefore, the result is theoretically identical to a scan conversion of a tool swept volume model, but the computational complexity

is much lower. Let the start and end CL points of a tool motion be denoted by $\boldsymbol{P}$ and $\boldsymbol{Q}$, respectively, and the corresponding unit tool axes by $\boldsymbol{u}$ and $\boldsymbol{v}$, as shown in Figure 4.1. Transforming the CL points and tool axes into the dexel coordinate system by using Equation (2.2) yields $\boldsymbol{P}'$, $\boldsymbol{Q}'$, $\boldsymbol{u}'$, and $\boldsymbol{v}'$, respectively. The linear sweeping vectors of the top and the bottom center points, denoted by $\boldsymbol{s}$ and $\boldsymbol{t}$, are given in the dexel coordinate system by,

$$\begin{aligned} \boldsymbol{s} &= \boldsymbol{Q}' - \boldsymbol{P}' \\ \boldsymbol{t} &= \boldsymbol{s} + L\,(\boldsymbol{v}' - \boldsymbol{u}') \end{aligned} \tag{4.1}$$

The total number of instances of motion, denoted by n, is given by,

$$n = \frac{max\,(|s_x|, |s_y|, |s_z|, |t_x|, |t_y|, |t_z|)}{w} \tag{4.2}$$

The tool location and axis defined in the world coordinate system at each instance of motion, denoted by $\boldsymbol{I}^w$, are then defined by,

$$\boldsymbol{I}_i^w = [\ \boldsymbol{P} + \frac{i}{n}(\boldsymbol{Q} - \boldsymbol{P})\ ,\ \boldsymbol{u} + \frac{i}{n}(\boldsymbol{v} - \boldsymbol{u})\ ] \tag{4.3}$$

where $i = 1, \ldots, n$. The coordinate values of instances in the dexel coordinate system, denoted by $\boldsymbol{I}^d$, is given by,

$$\boldsymbol{I}_i^d = [\ \boldsymbol{P}' + \frac{i}{n}\boldsymbol{s}\ ,\ \boldsymbol{u} + \frac{i}{n}\boldsymbol{t}\ ] \tag{4.4}$$

A dexel representation of tool model is thus constructed at every instance to sequentially update the workpiece by Boolean difference operations. Since scan converting an equivalent tool swept volume into a dexel representation

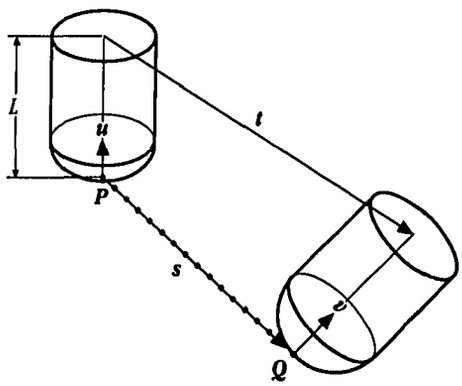


Figure 4.1 Computing instances of tool motion

would be limited by the size of dexels, the instance of motion approach provides the same result, but does not require creation and scan conversion of the swept volume.

4.1.2 Three-axis NC milling simulation

The invariance of tool axis vector in a three-axis milling process enables the dexel representation of a tool to be translated to every instance of motion by simply translating the z-depth and correspondent x-y address. Thus to represent the material removal process, regularized Boolean difference operations are performed between the workpiece model and each translated instance of the tool model. Thus, the three-axis NC milling simulation algorithm follows,

1. *Scan convert the workpiece and fixtures and store the dexels in an array W.*
2. *Graphically display W.*
3. *Scan convert the milling tool located at the origin of the dexel coordinate system and store the dexels in an array T.*
4. *Read the first CL point in the tool path file to be verified.*
5. *Read the next CL point in the data file, if end-of-file, the simulation algorithm is complete.*
6. *Compute the instance of motion using Equation (4.4).*
7. *For every instance of motion,*
   a. *Offset the dexel array T by* $(\frac{I^d_{tx}}{w}, \frac{I^d_{ty}}{w})$, *and translate the dexel depth by* I^d_{tz} *with correction*
   b. *Perform regularized Boolean difference operations between W and T, i.e.,* $W = W - T$.
   c. *Display* $W \cup T$
   d. *Erase the tool image.*
8. *Goto step 5.*

The depth correction method used in Step 7-*a* is similar to the method described in the bounding ellipse scan conversion algorithm for cylinder primitive, i.e., Equation (2.17) is used to find the depth offset.

The time complexity of the simulation algorithm is dominated by the number of dexels in a tool model, denoted by n_t, and by the total instances of tool motion, denoted by n_i. Hence an $\boldsymbol{O}(n_i\, n_t)$ performance is

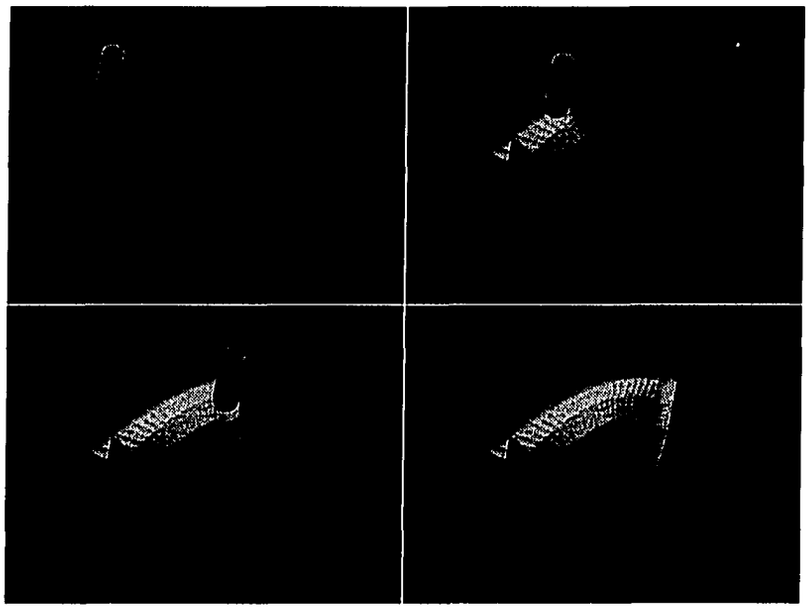


Figure 4.2 Sequential images of a three-axis NC milling simulation

expected. An example of the three-axis NC milling simulation algorithm applied on a fan-shaped surface is demonstrated in Figure 4.2, in which the grey-colored block depicts the workpiece, the orange-colored blocks represent the machine table, and the shape of workpiece is updated via regularized Boolean difference operations. The total running time of this example, on a SGI Indigo/Elan R3000 workstation, is 130 seconds for 2,818 instances of motion, or 21.67 instances per second.

4.1.3 Five-axis NC milling simulation

In three-axis milling simulations, the dexel representation of the cutting tool is scan converted only once and is translated to all instances of motion with depth correction. However, in five-axis milling simulation, since the tool orientation generally changes for every instance, the tool must be scan converted at every step. This distinction makes five-axis simulation more computationally intensive than three-axis simulations. The computation time can be reduced by checking whether the tool axis is fixed during a motion, in which case the faster three-axis simulation method can be applied to the segment of motion. The algorithm of five-axis simulation follows,

1. *Scan convert the workpiece and fixture and store the dexels in an array W.*
2. *Display W.*
3. *Read the first CL point in the data file.*
4. *Read the next CL point in the data file, if end-of-file is reached, the simulation algorithm is complete.*
5. *Check if the tool motion is three-axis, if yes, use steps 3, 6, and 7 of the three-axis simulation algorithm. Otherwise, compute the instance of motion using Equation (4.3).*
6. *For every instance of motion,*
   a. *Scan convert the milling tool.*
   b. *Perform regularized Boolean difference operations between W and T, i.e., W=W-T.*
   c. *Display W∪T.*
   d. *Erase the tool image.*
7. *Goto step 4.*

The time complexity of scan conversion is $\boldsymbol{O}(n_t)$ where n_t is the number of tool dexels. Hence, the time complexity of five-axis simulation algorithm is $\boldsymbol{O}(n_i\, n_t)$. The algorithm is similar to the three-axis one, however, the scan conversion process is performed at every instance of motion.

An example of the five-axis milling simulation algorithm applied on the fan-shaped surface is shown in Figure 4.3. The total processing time (on the same computer) is 573 seconds to simulate 5,099 instances of tool motion, or 8.90 instances per second.

4.2 Discrete Dexel NC Verification

An important property of the dexel representation is that the dexel points are on the surface of the represented object. This property results from the scan conversion process which intersects object with rays to construct dexels. Furthermore, since the dexel coordinate system is fixed for all scan-converted objects, this property also holds for dexel representations resulting from regularized Boolean set operations. The discrete dexel

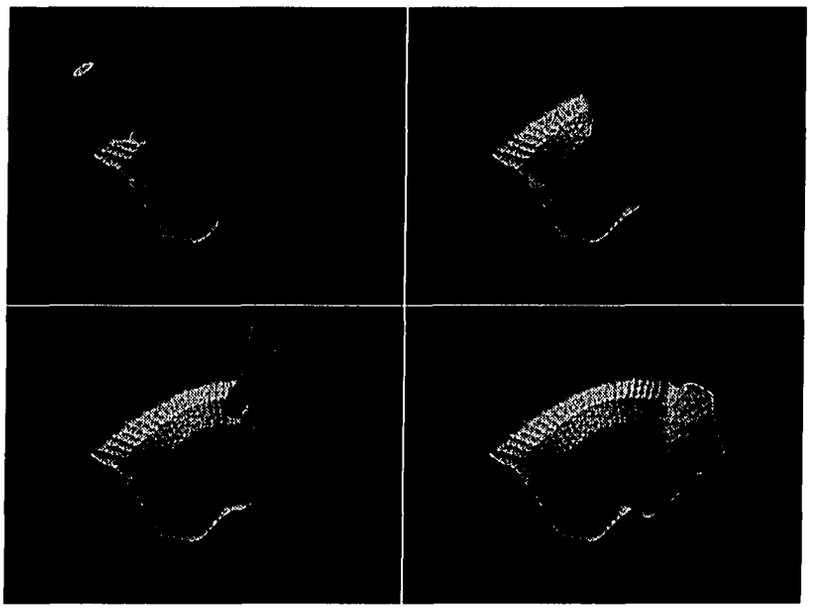


Figure 4.3 Sequential images of a five-axis NC milling simulation

NC verification algorithm exploits this property and is capable of computing the deviation between the dexel-based milled surfaces and actual design surfaces during the simulation process.

4.2.1 Discrete dexel NC verification algorithm

The discrete dexel NC verification algorithm is in essence an inverse formulation of the discrete vector intersection approach. The discrete vector intersection approach measures milling errors from the design surfaces to the tool swept volume. Instead, discrete dexel verification computes the errors from the milled part to the design surfaces. So the localization and intersection sub-tasks of the former approach are replaced by a minimum-distance calculation between center points of dexels and design surfaces. The minimum-distance calculation is only performed on dexels that are updated by regularized Boolean difference operations, thus no additional localization effort is needed. Figure 4.4 illustrates an instance of the verification algorithm, in which a dexel is updated by a regularized Boolean difference operation and the new dexel point $\boldsymbol{C}$ is verified for milling error by a minimum-distance calculation algorithm.

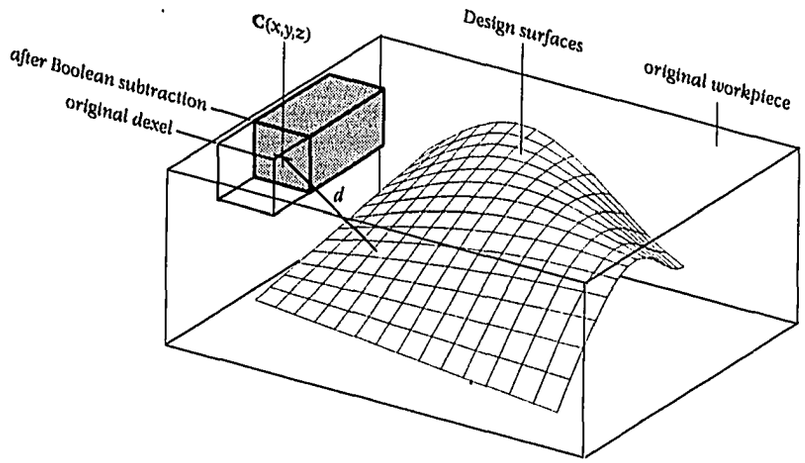


Figure 4.4 Illustration of the discrete dexel verification algorithm

4.2.2 Minimum-distance calculation

The minimum-distance calculation algorithm computes the minimum distance d between a dexel point and the design surfaces. Since design surfaces are not discretized, several methods based on an orthogonal property [Pegna and Wolter, 1990, Barnhill and Kersey, 1990] can be utilized to find a surface near-point. The orthogonal property is due to the fact that the minimum distance between a space point $\boldsymbol{P}$ and a surface $\boldsymbol{S}(u,v)$ occurs at a surface point $\boldsymbol{Q}$, on which vector $\boldsymbol{PQ}$ is perpendicular to the surface tangent plane, as illustrated in Figure 4.5. The orthogonal property is formulated in two ways including a perpendicular method and a minimum norm method.

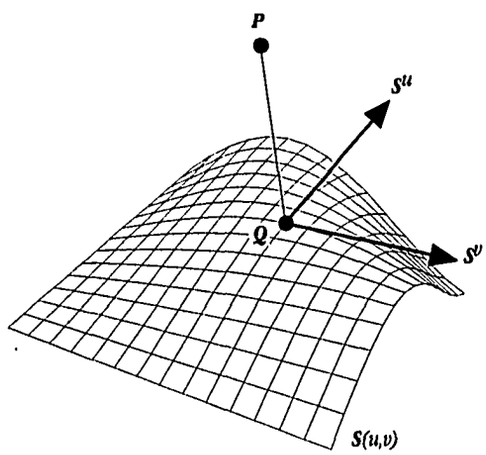


Figure 4.5 Orthogonal property of surface near point

By definition, the tangent plane of the surface at $\boldsymbol{Q}$ is formed by the tangent vectors $\boldsymbol{S}^u$ and $\boldsymbol{S}^v$, the perpendicular method formulates a system of two equations,

$$\begin{aligned}(\boldsymbol{P}-\boldsymbol{Q})\cdot\boldsymbol{S}^u &= 0\\ (\boldsymbol{P}-\boldsymbol{Q})\cdot\boldsymbol{S}^v &= 0\end{aligned} \tag{4.5}$$

where superscripts u and v denote the partial derivatives of the surface along u- and v-direction, respectively, and $0\leq u, v\leq 1$. Equation (4.5) generates a system of two nonlinear equations and is solved by using the Newton/ Raphson method[Kincaid and Cheney, 1991] with a convergence condition,

$$\left| \boldsymbol{S}^u\Delta u+\boldsymbol{S}^v\Delta v \right| \leq \varepsilon \tag{4.6}$$

where ε is a given tolerance value. Since the surface is bounded in the parameter space, solution to Equation (4.5) may not exist. However, the Newton/Raphson method will still converge to the nearest point on the surface boundary that generates the minimum distance.

The minimum norm method formulates the problem by minimizing the projection of $\boldsymbol{PQ}$ on the tangent plane. Thus a system of two equations that computes a search direction in the parameter space is given by[Barnhill and Kersey, 1990],

$$\begin{bmatrix}\Delta u\\ \Delta v\end{bmatrix} = \begin{bmatrix}u_{i+1}-u_i\\ v_{i+1}-v_i\end{bmatrix} = \left[\boldsymbol{J}^T\boldsymbol{J}\right]^{-1}\boldsymbol{J}^T\left[\boldsymbol{P}-\boldsymbol{Q}\right] \tag{4.7}$$

where $\boldsymbol{J}$ is the Jacobian matrix evaluated at (u_i,v_i),

$$\boldsymbol{J} = \left[\boldsymbol{S}^u\ \boldsymbol{S}^v\right]_{u_i,\,v_i} \tag{4.8}$$

The values of Δu and Δv obtained in Equation (4.8) are used to refine an initial guess (u_0,v_0) until Δu and Δv satisfy the convergence condition given in Equation (4.6).

These two methods are similarly efficient and accurate. The difference is that the perpendicular method requires evaluation of second order surface derivatives, so it is generally less efficient than the minimum norm method. However, the information provided by the second order derivatives improve the speed of convergence and the robustness of the software implementation.

4.2.3 Graphical representation of milling errors

The results of dimensional milling error verification are displayed by several hues depicting the depth of cut. A color lookup table is prepared to interpret a cut value into a proper hue for dexel display. The lightness of a hue is determined by the normal vector recorded in each dexel, thus a realistic object image can be rendered. Figure 4.6 demonstrates an example of the graphical representation of milling verification using the perpendicular projection method during a milling simulation process which took 956 seconds for 5099 instances of motion, or 5.34 instances per second. In this figure, color-coded milling error information is encoded and displayed on the milled surface and the color bar shown at the left-hand side depicts the range of milling error.

The green color represents errors that are within a specified tolerance range (-0.01 to 0.01, in this example), the upper blue colors represent the amount of undercut, relative to the nominal design surfaces, and the lower colors represent overshoot. The upper and lower bound of the color map is determined by a user specified range (-0.04 to 0.04 in this case), called *range of interest* [Oliver and Goodman, 1990]. Hence given a depth of cut, the corresponding color to depict the error is obtained via the color lookup table. However, for overshoot values

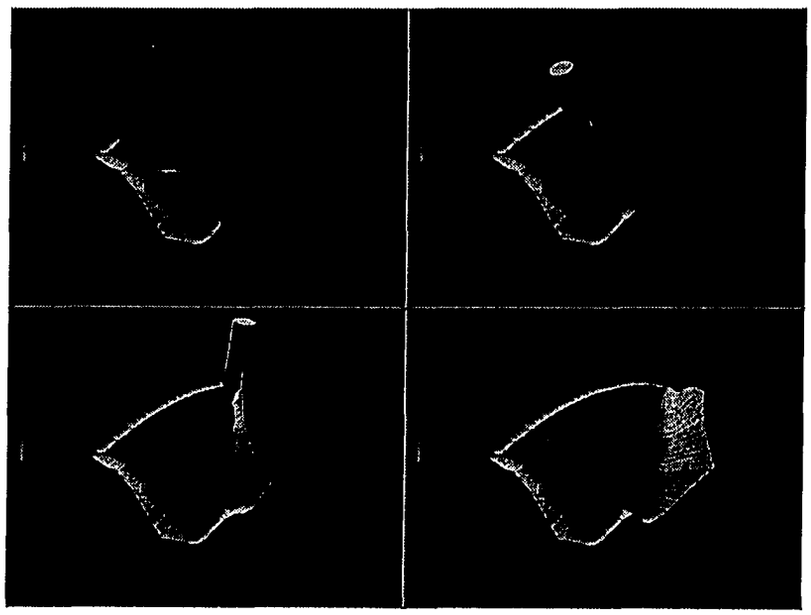
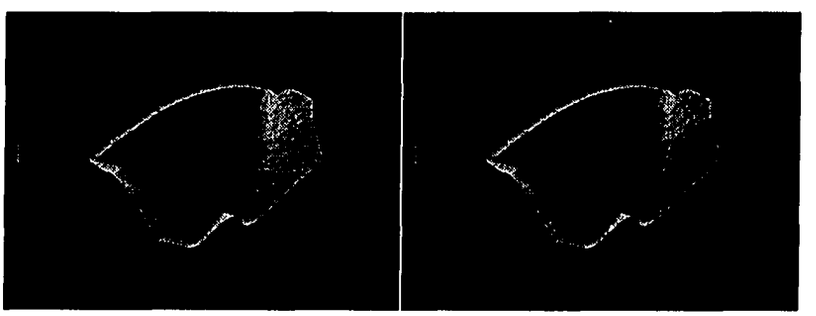


Figure 4.6 A result of the discrete dexel NC verification algorithm

deeper than the lower bound of the range of interest, the color depicting the deepest overshoot is used. For undercut larger than the highest bound, the color of milling tool is displayed.

The range of interest and tolerance can be changed to reveal different details of the milled surface. For example, Figure 4.7(a) shows the result generated from a range of interest [-0.04, 0.08] and a tolerance range [0, 0.02], hence the gouged area appears larger than in Figure 4.6. Figure 4.7(b) shows the result from a range of interest [-0.08, 0.08] and a tolerance range [-0.02, 0]. The ranges of interest and tolerance are also used in Figure 4.8 to demonstrate verification using the minimum norm projection method. The overall quality and efficiency is similar.

Assuming the minimum distance calculation can be done in $\boldsymbol{O}(1)$ time, the time complexity of the verification algorithm performed simultaneously with the simulation process is also $\boldsymbol{O}(n_i\, n_t)$. Note that the time complexity is proportional to the instances of motion for an in-process error verification, therefore, a post-process



(a) Range of interest: [-0.04, 0.08]
tolerance: [0, 0.02]

(b) Range of interest: [-0.08, 0.08]
tolerance: [-0.02, 0.0]

Figure 4.7 Verification using different ranges of interest and tolerances

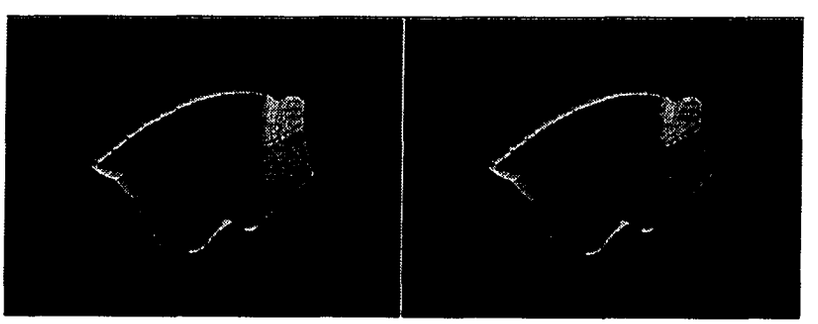


Figure 4.8 Verification using minimum norm projection method

verification approach that does verification after simulation is more efficient for cases with large values of n_t. Post-process verification scans through all dexels of the workpiece model and verifies dexel points that have colors of the tool, i.e., updated dexel points. Hence the time complexity for post-process verification is $\boldsymbol{O}(n_w)$, where n_w is the number of dexels of workpiece. The performance is generally far better than in-process verification. For example, Figure 4.7 was generated from a post-process verification with running time of 585 seconds (573 seconds for five-axis simulation plus 12 seconds for verification) which is considerably faster than the running time observed for in-process verification (956 seconds).

5. TOOL PATH CORRECTION

The main objective of an NC milling verification system is to present potential problems in tool paths so that NC programmers can modify the paths to avoid these problems. Tool path modification is typically done by manually changing the CL data of the tool path in question or by moderately altering the design surface model and re-generating new tool paths. These two approaches are generally time-consuming, inaccurate, or, even worse, may introduce more problems. Therefore, to eliminate the complexity of tool path modification, a *reduction of intersecting volume* algorithm and a *tool assembly sensing* algorithm are developed to modify the tool paths to eliminating gouges and collisions, respectively. The verification algorithm described in Chapter 4 serves as a gouge detection process to provide the gouge elimination algorithm with milling error information. The gouge elimination algorithm modifies tool paths so that the resulting depth of cut is equal to lower limit of a specified tolerance values. Furthermore, unexpected collision between the tool assembly and workpiece or fixtures is also detected via the use of virtual sensors on a tool assembly model and is avoided by alternating the tool axis orientation.

5.1 Gouge Elimination

Assuming a tolerance is given by $[t_l, t_h]$, a gouge is defined by a depth of cut deeper than the lower bound of the tolerance, i.e., t_l. The main objective of the reduction of intersection volume algorithm is to reduce the depth of cut to meet the tolerance specification via tool path modification. The algorithm computes the intersection volume of the tool and workpiece using the regularized Boolean intersection operation, then checks the volume for gouge using the discrete dexel verification algorithm. Detected gouges are removed by modifying the tool path by using a *guide vector*. Assuming m gouging points are detected at an instance of tool motion, the guide vector $\boldsymbol{G}$, as illustrated in Figure 5.1, is computed by,

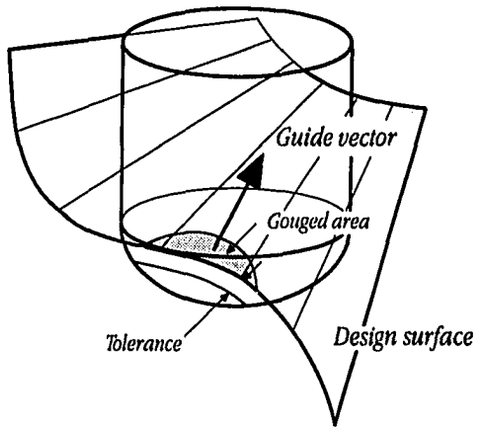


Figure 5.1 Gouge elimination by using the guide vector

$$\boldsymbol{G} = \sum_{i=1}^{m} (d_i - t_i)\,\boldsymbol{n}_i \tag{5.1}$$

where d_i is the depth of gouge, $\boldsymbol{n}_i$ is the normal vector at each dexel face point, and t_i is the lower bound of tolerance. If the summation in Equation (5.1) is zero or within a range $[-\varepsilon,\varepsilon]$, where ε is a small value, the tool axis is used for the guide vector to avoid numerical error. The guide vector is normalized and its magnitude, denoted by h, is assigned to the maximum depth of gouge, i.e.,

$$h = max(d_i - t_i),\ i = 1 \ldots n \tag{5.2}$$

The reduction of intersecting volume algorithm is performed prior to the regularized Boolean difference operation thus the workpiece is not updated until the gouge is eliminated. The algorithm for detecting gouge and modifying a tool motion defined by CL points $\boldsymbol{A}$ and $\boldsymbol{B}$ follows,

1. *Compute the intersection volume of the milling tool at CL point* $\boldsymbol{A}$ *and store the volume in D*
2. *Perform dimensional verification for D*
3. *If no gouge is detected, goto step 4. Otherwise,*
   a. *Compute a guide vector* $\boldsymbol{G}$ *using Equation (5.1) and an offset amount h by Equation (5.2)*
   b. *Offset* $\boldsymbol{A}$ *by* $h\boldsymbol{G}$*, i.e.,* $\boldsymbol{A} = \boldsymbol{A} + h\boldsymbol{G}$
   c. *Goto step 1 to ensure the new CL point* $\boldsymbol{A}$ *is free from gouge*

4. *Eliminate gouge at **B** using steps 1, 2, and 3 (**A** and **B** are then free from gouge after this step)*
5. *start_instance ← **A**, end_instance ← **B***
6. *Compute the instances of motion between start_instance and end_instance*
7. *Subtract the tool model at start_instance from the workpiece*
8. *For every instance of motion, except for the start_instance and end_instance, perform:*
   a. *Compute the volume of intersection and record the volume in D*
   b. *If there is no gouge, subtract D from the workpiece. Otherwise,*
      i. *Find a new gouge-free CL via step 1-3*
      ii. *Ensure no gouge is generated between the previous instance and the new CL point by using this algorithm recursively*
9. *Subtract the tool model at end_instance from the workpiece*

An example of the algorithm applied in 2D is illustrated in Figure 5.2. In this example, the tool follows the tool motion defined by CL points ***A*** and ***B*** and thus runs though the workpiece. The reduction of intersecting volume algorithm first moves CL point ***B*** to a new location ***B'*** to avoid gouge, then recursively modifies the intermediate instances of motion to eliminate gouges.

Since the start and end instances of a tool motion are pre-processed to be free from gouge and the number of instances is limited by the dexel resolution, i.e., the displacement of the tool at each instance is less than or equal to the size of a dexel, the algorithm will move the tool away from gouges. Hence the algorithm will always converge. The algorithm can not be formulated as a post-processing step after milling simulation because the intersection

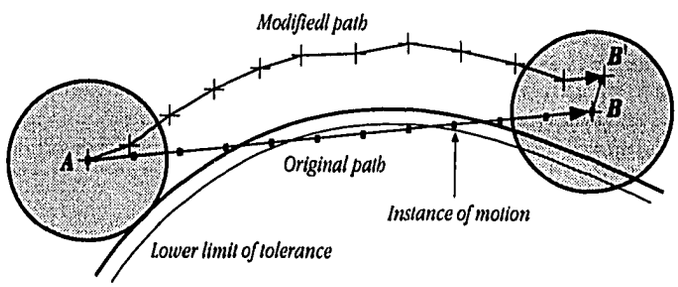


Figure 5.2 A 2D example of gouge elimination and tool path modification

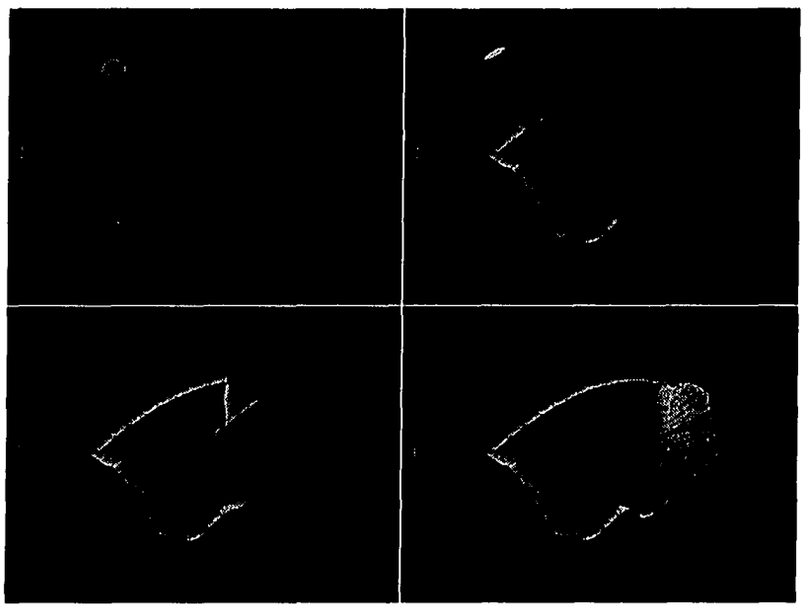


Figure 5.3 Performing gouge elimination during milling simulation

volume is time-dependent. Furthermore, it is very difficult to identify the CL points that induce a gouge after simulation is complete.

Moving a CL point to avoid a gouge at an instance can be done in $O(1)$ time. Thus the time complexity of the reduction of intersecting volume algorithm is also $O(n_i\, n_t)$. An application of this algorithm to the fan-shaped surface shown in Figure 4.6 is demonstrated in Figure 5.3. Gouges are completely eliminated and a modified tool path is generated to replace the original one. The running time for this example is 2402 seconds for 5099 instances of motion, or 2.123 instances per second.

5.2 Eliminating Interference Between Tool and Fixtures

An extension of the gouge elimination method is applied to detect and correct cutting on holding fixtures. Since dexels of a fixture carry a specific index to identify themselves, cutting on a fixture can be easily detected by checking the index of dexels involved in regularized Boolean intersection operations. To eliminate cutting on fixtures, the normal vector of the intersected fixture dexel, i.e., n_i in Equation (5.1), is given by the plane normal, while the

amount of gouge is assigned to the size of a dexel, i.e., w, because the step size between tool instances is typically no more than w. Therefore, the gouge elimination formulation expressed in Equation (5.1) holds for eliminating interference between the tool and holding fixtures. Note that this type of interference problem is usually better addressed by changing the location of the fixtures. Thus modifying the tool path to avoid fixture interference is not recommended unless no other alternative location for the fixture is available.

5.3 Collision Detection and Elimination

Unexpected collision between the entire tool assembly and the workpiece or fixtures is also a major hazard that can damage machines and cause severe injuries to machine operators. Such collisions must be eliminated before these consequences occur in actual milling productions. However, the collision problem is seldom considered in tool path generation algorithms partially because the workpiece setup and the dimensions of NC machine vary widely. Consequently, for three-axis milling applications, collisions are typically manually detected and removed, as for instance when the length of the cutter is less than the depth of cut and the cut is being made in the vicinity of fixtures. For five-axis milling, the additional rotational motions make prediction of collisions much more difficult, not to mention avoidance. Collision avoidance relies heavily on the skill of machine operators performing trial milling, which is time consuming and costly. Therefore, an algorithm for detecting and eliminating collisions of five-axis tool paths is developed to assist tool path modification.

5.3.1 Representation of milling tool assembly

A tool assembly is divided into several sections such as *toolholder*, *shank*, and *flute*, as shown in Figure 5.4. The flute section is the only valid area for material removal and is modeled by a dexel representation. The toolholder and shank are the sections that must avoid all contact. Thus the terminology *tool assembly* used in the following discussion excludes the flute section.

Each section of the cutter assembly is modeled by a cone primitive (a cylinder is a special case of cone) and the cone is approximated by several circular cross sections. Each circular cross section is discretized into points,

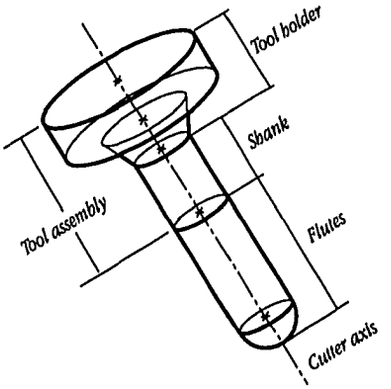


Figure 5.4 Milling tool assembly

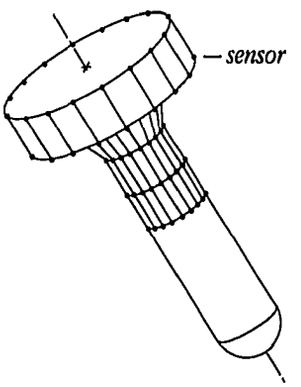


Figure 5.5 Approximation of tool assembly

called *sensors*, as shown in Figure 5.5. The density of sensors on each cross section is dynamically adjustable according to dexel size. More sensors on a tool assembly will generate better results of collision detection and elimination.

5.3.2 Collision elimination algorithm

Since each sensor is fixed on a tool assembly, the dexel coordinate values of each sensor can be computed at any cutter location. Therefore, by comparing the z depth value of each sensor with the dexel representation of workpiece and fixtures, as illustrated in Figure 5.6, collisions can be detected whenever a sensor is inside of the dexels or too close to a dexel. Furthermore, a safety range can be assigned to the fixtures to further ensure the

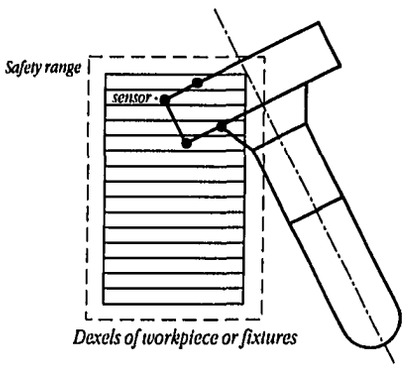


Figure 5.6 Collision detection via depth comparison

avoidance of tool paths, i.e., the safety range keeps the tool assembly at least a certain distance away from the workpiece and fixtures.

The collision detection algorithm generates an array that contains all colliding sensors at an instance of tool motion. This set of sensors is utilized to analyze the type of collision and to establish an *escape vector* that can effectively lead the cutter away from collisions. Two types of collision are classified on the basis of a semicircular test. The semicircular test first projects all colliding sensors to the top circle of the tool assembly via offsets along the tool axis. Then it checks if all projections of sensors lie in one half of the circle, in which case, as illustrated in Figure 5.7, the first type of remedial action is taken, otherwise, the second type, as illustrated in Figure 5.8, is used to eliminate the collision.

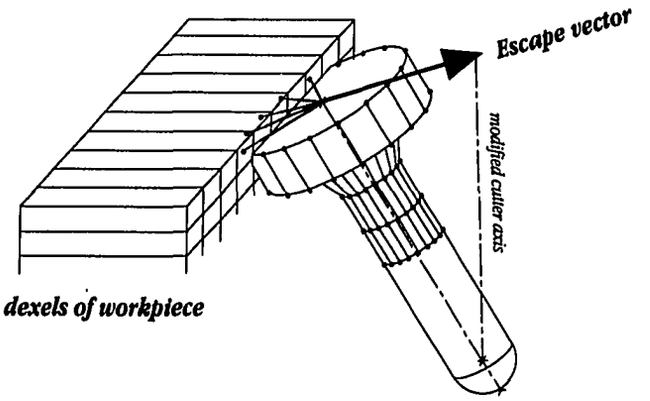


Figure 5.7 Eliminate collision by pivoting tool axis

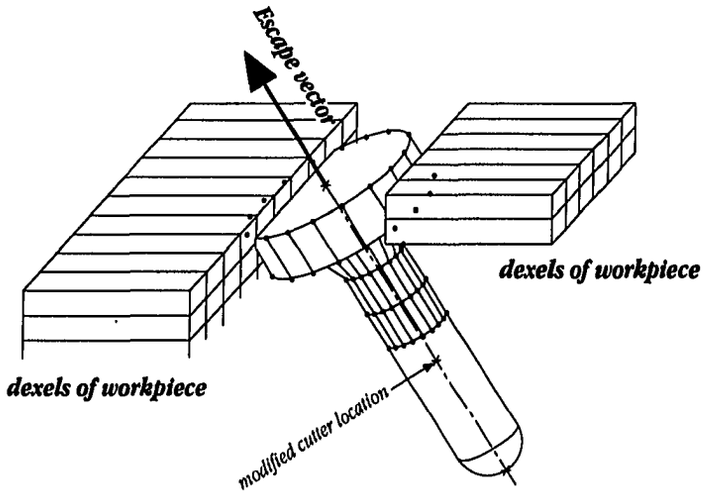


Figure 5.8 Eliminate collision by translating tool location

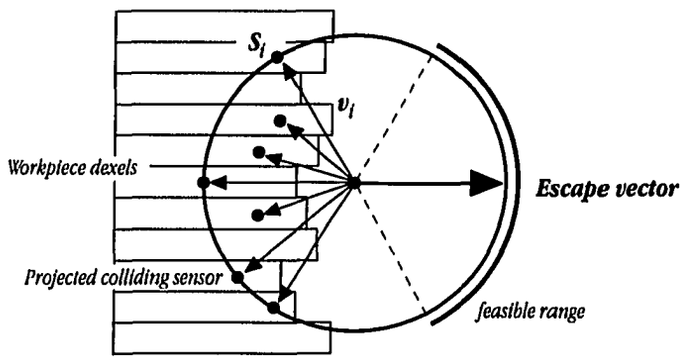


Figure 5.9 Collision detection via depth comparison

The escape vector for the first type of collision is established perpendicular to the tool axis and is used to tilt the tool axis away from the workpiece and fixtures. Denoting the projected sensors on the top circle by $\boldsymbol{s}_i$, the unit vectors that start from the circle center point to $\boldsymbol{s}_i$ by $\boldsymbol{v}_i$, the escape vector, $\boldsymbol{E}$, as illustrated in Figure 5.9, is computed by first finding the span of $\boldsymbol{v}_i$'s, then inverting the sign of the central vector in the span for $\boldsymbol{E}$. A new tool orientation $\boldsymbol{u}'$ is thus computed by,

$$\boldsymbol{u}' = \frac{\boldsymbol{l} \cdot \boldsymbol{u} + \boldsymbol{w} \cdot \boldsymbol{E}}{| \boldsymbol{l} \cdot \boldsymbol{u} + \boldsymbol{w} \cdot \boldsymbol{E} |} \tag{5.3}$$

where l is the length of the tool assembly including the flutes section. Note that a five-axis milling machine generally does not support 360° axial orientation for the rotational degrees of freedom, i.e., the admissable angular range between the tool axis and spindle axis is limited. Thus if $\boldsymbol{u}'$ reaches the limit, the second case of tool path modification is employed.

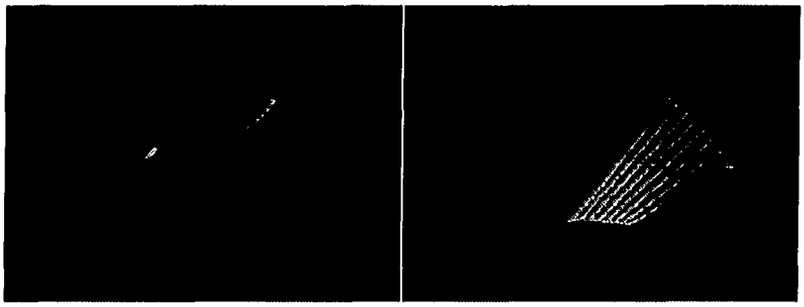
For the second type of collision, the escape vector is set equal to the tool axis to modify the CL point. The new CL point $\boldsymbol{P}'$ is given by,

$$\boldsymbol{P}' = \boldsymbol{P} + w \cdot \boldsymbol{E} \tag{5.4}$$

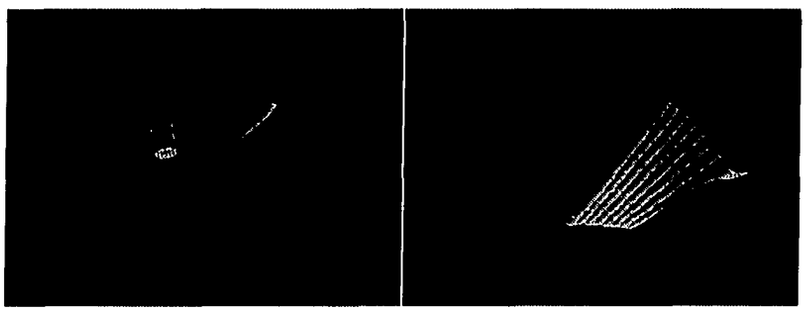
The optimal magnitude for adjusting the cutter location is difficult to compute because the dexel representation does not have continuous surface definition, so the width of a dexel, which is small relative to the workpiece, is used as a step size for numerical iteration. Denoting the start and end CL points by $\boldsymbol{A}$ and $\boldsymbol{B}$, the algorithm is as follows,

1. *Record the world coordinate values of all sensors in an array*
2. *start_instance*$\leftarrow \boldsymbol{A}$, *end_instance* $\leftarrow \boldsymbol{B}$
3. *Compute the instance of motion between start_instance and end_instance using the five-axis simulation algorithm.*
4. *For every instance of motion,*
   a. *Compute the location of sensors in the dexel coordinate system*
   b. *If no collision is found, perform Boolean difference operation between workpiece and tool, then proceed to the next instance. Otherwise,*
      i. *Determine the type of collision and compute the escape vector.*
      ii. *Find a new CL point for which collision is eliminated.*
      iii. *Set the start_instance by the current instance, and the end_instance by the new CL point. Recursively use steps 2, 3, and 4 to eliminate intermediate collisions.*

The time complexity of the collision elimination algorithm is $\boldsymbol{O}(n_i\, n_s)$, where n_s denotes the number of sensors. Note that the modified tool paths of the collision detection and elimination algorithm are not guaranteed to



(a) Collision occurs during five-axis milling



(b) Collision detected and eliminated

Figure 5.10 Comparison of results with and without collision elimination

be gouge free. Therefore, it is necessary to combine this algorithm with the gouge elimination algorithm to determine a final location of a new CL point on which collision is eliminated and accuracy is maintained.

Figure 5.10 demonstrates the effectiveness of the collision detection and elimination algorithm by comparing the results of NC simulation with and without application of the algorithm. In Figure 5.10(a), the cutter is too short to cut through the left hand side portion of the workpiece thus it runs underneath the workpiece, which is impossible in real milling operations. In Figure 5.10(b), a wireframe model of the tool assembly is attached to the dexel-based tool and is used to detect collision. Finally, a workpiece produced by the collision-free tool path is obtained. The running time for the example shown in Figure 5.10(a) is 241 seconds for 1215 instances of motion, or 5.04 instances per second, and the running time for Figure 5.10(b) is 249 seconds. The time used for performing collision detection and elimination is comparably negligible because the number of sensors is far less than the number of dexels of a tool model.

5.4 Integration of Gouge- and Collision-Elimination

Collision detection and elimination algorithm does not guarantee the modified tool path being gouge-free, especially when flat- and toroidal-end milling tools are used. Thus, the reduction of intersection volume algorithm for gouge-elimination is needed in the collision elimination process.

The mechanism for integrated path correction is straight-forward: the two algorithms iteratively perform tool path modification until both gouges and collisions are removed. The algorithm follows,

1. *has_gouge ← 0 and has_collision ← 0*
2. *Detect gouge via regularized Boolean intersection operation. If found, has_gouge ← 1. Otherwise go to step 4.*
3. *Perform one iteration of gouge-elimination (gouge need not be completely eliminated)*
4. *Detect collision. If found, has_collision ← 1. Otherwise go to step 6.*
5. *Perform one iteration of collision-elimination (gouge need not be completely eliminated)*
6. *has_gouge==0 and has_collision==0 → done. Otherwise, go to step 1.*

The integrated path correction process performs recursively until no gouge and collision are found. Figure 5.11(a) demonstrates a verification result with no path correction and Figure 5.11(b) shows the effectiveness of the integrated path correction algorithm, which took 389 seconds for 1215 instances, or 3.12 instances per second.

Since both algorithms modify tool paths to achieve their respective goals, a potential problem is that the algorithms may mutually cancel the gains of tool path correction, i.e., collision-elimination introduces gouge and gouge-elimination introduces collision. However, several facts are noted in the integration process of path correction process. First, the previous instance of motion of the current tool location is guaranteed gouge- and collision-free. Second, the remedial actions of the algorithms are different, i.e., the gouge-elimination algorithm translates the tool location while the collision-elimination mainly changes the orientation of the tool. Furthermore, the tool will be translated along the tool axis if the second case of collision occurs or the orientation limit of an NC machine is reached. Therefore, the path correction process can always be done in a finite number of recursive steps.

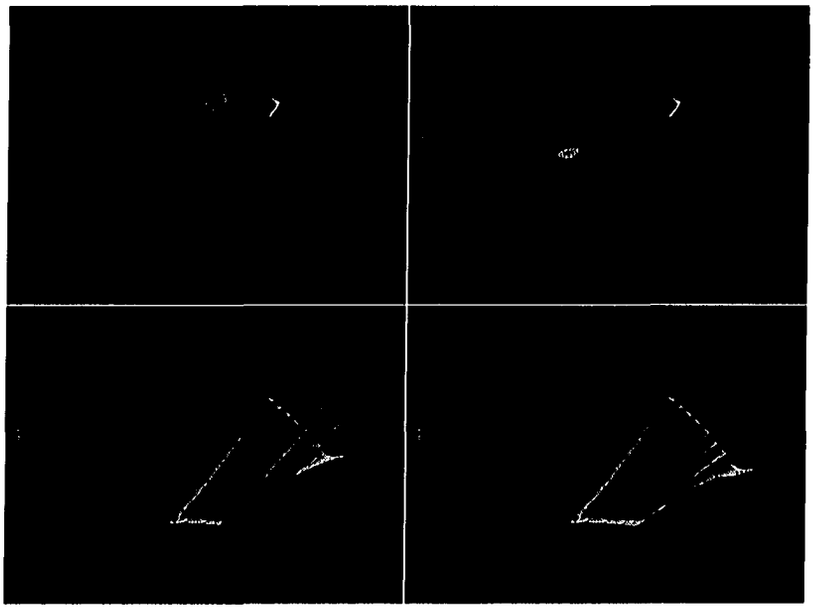


Figure 5.11 Sequential images of tool path correction algorithm

5.5 Five-Axis Tool Path Generation

The path correction algorithm also presents a unique method for five-axis tool path generation: generate initial five-axis tool paths and modify the paths to be gouge- and collision-free. The CL data for an initial five-axis tool path may be generated from a three-axis tool path generation algorithm [Huang and Oliver, 1992] and the tool axis vectors are obtained from the surface normal vectors at the cutter contact points. The initial tool paths are then modified by using the tool path correction algorithm to avoid potential gouge and collision. The advantage of this tool path generation approach is that the tool path modification can be performed after a the workpiece setup is determined, i.e., the size and shape of workpiece and the locations of fixtures is decided. Therefore, no additional check is necessary and the tool paths are ready for immediate application.

6. SOFTWARE IMPLEMENTATION

The discrete dexel NC milling verification algorithm is implemented using the ANSI *C* language and the graphics functions of the IRIS Graphics Library (GL) on a Silicon Graphics workstation platform. Since a workpiece is decomposed into a large set of dexels, the major considerations during the implementation were the memory consumption for dexel storage and the efficiency of dexel operations. A linked-list data structure is constructed to optimize memory usage and enable linear time regularized Boolean set operations. Several other enhancements of the simulation and verification algorithms are also implemented to support real-time performance, including instance of motion control, local graphics update, distance-square lookup scheme, initial surface-near point lookup method, and regional search algorithm. Finally, experimental results and performance statistics are presented and discussed.

6.1 Memory Allocation for Dexel Coordinate System

The initial size of the data array for storing the dexel representation of the workpiece and fixtures is estimated by the number of grid points in the bounding rectangle on the dexel plane. Assuming the bounding rectangle is computed as W by H, as shown in Figure 6.1, the required array size to address all grid points is $W/w+1$

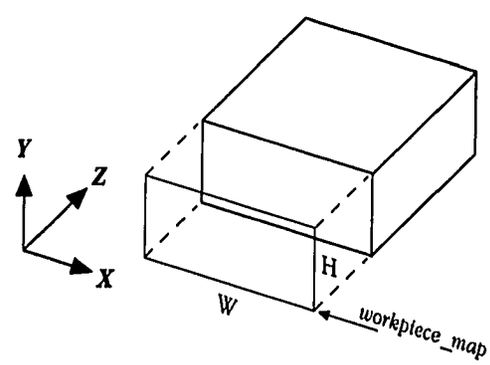


Figure 6.1 Memory allocation of data array for addressing grid points

and $H/w+1$, denoted by W^d and H^d, respectively. Also denoting the total number of grid points by *grid_point_no* $(= W^d H^d)$, two arrays are allocated. The first is a 2D array of data type *unsigned long*, denoted by *workpiece_map* $[W^d][H^d]$, which records the index of the first dexel of each grid point in the bounding rectangle or *0* if no dexel exists. The other is an 1D array of data type *dexel*, denoted by *dexel_list*, which stores the dexel data of the workpiece and fixture model. The size of *dexel_list* is initially *grid_point_no*, however, it is dynamically adjusted to accommodate the number of required dexels. The space complexity to support simulation and further computations is proportional to the grid points in the bounding rectangle, i.e., $O(grid\_point\_no)$.

The data arrays *workpiece_map* and *dexel_list* form a linked-list data structure [Manber, 1989] in which the root is a grid point in *workpiece_map*. Insertion and deletion functions on the data structure are implemented to manipulate the linked-list in $O(1)$ time.

To attach a dexel to a grid point, an index of free dexel in *dexel_list* is assigned to the corresponding cell of *workpiece_map*. After a dexel is assigned, the index moves to the next free dexel for further assignment. For example, assuming the index of the current free dexel is *free_dexel_id*, assigning a dexel to a grid point *(ix, iy)* is done by setting *workpiece_map[ix][iy]=free_dexel_id*, thus the dexel information can be stored in *dexel_list [workpiece_map[ix][iy]]*, and the index of the current free dexel is offset by 1. Deeper dexels at the grid point are similarly assigned. Therefore, the length of a dexel chain at any grid point is flexible. The first dexel record, *dexel_list[0]*, is reserved as a dummy record to terminate the linked-list.

The scan conversion process of the workpiece and fixtures installs dexels into *workpiece_map* by retrieving free dexels in the array *dexel_list* as described above and dexel chains are formed by assigning the index of the following dexel to the struct member *next_dexel*. Finally, the memory space allocated in array *dexel_list* is organized to represent objects in the dexel coordinate system. The initial allocation of *dexel_list* may not be sufficient for a complicated dexel representation, in which case the storage space is reallocated by an increment of *grid_point_no*. Note that the dexel chain is a directed linked list, i.e., the next dexel can be found in the struct member *next_dexel*, but the previous dexel can not be determined in such an obvious way. Traversal (linear scan) from the grid point is needed to locate the previous dexel of a given dexel. All dexel chains eventually connect to the dummy dexel, *dexel_list[0]*, thus traversal stops when the dummy dexel is reached.

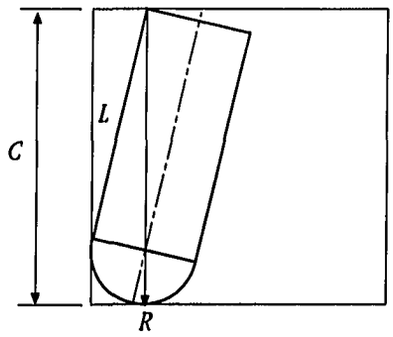


Figure 6.2 Maximum projection length of a ball-end milling tool

Another data array, call *tool*, for storing the dexel representation of the milling tool is also allocated. Since a milling tool is a convex object, at most one dexel can exist on a grid point, hence the size of array *tool* is fixed after the dexel size is determined. For three axis simulation, the dimension of tool array is computed by finding the bounding rectangle on the dexel plane which contains the projection of the tool. For five-axis simulation, a square array, as illustrated in Figure 6.2, is allocated whose dimension is determined by computing the maximum projection length of a tool, denoted by C, using,

$$C = R + \sqrt{L^2 + R^2} \tag{6.1}$$

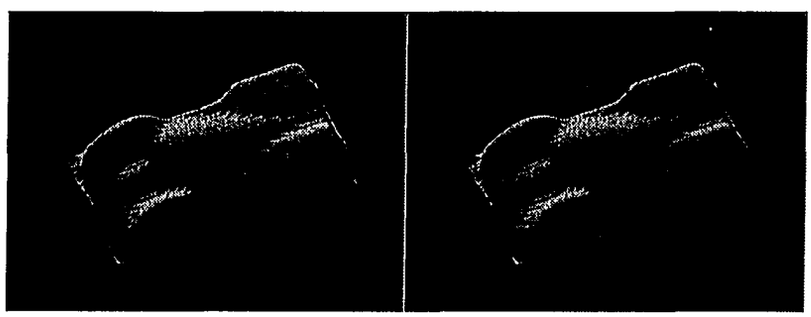
The allocated data array, *tool*, is always sufficient for constructing tool dexel model at any orientation.

6.2 Milling Simulation Implementation

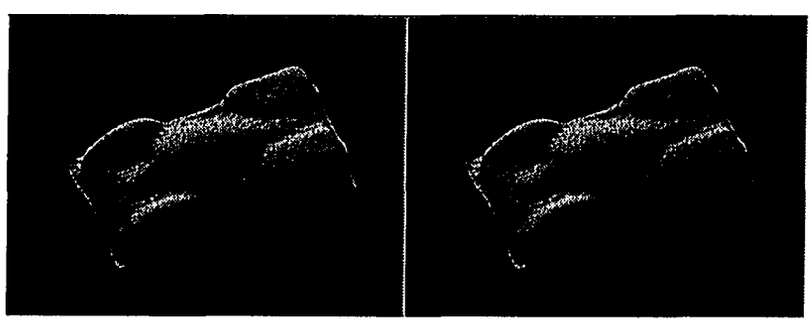
The major computational task for NC milling simulation is regularized Boolean difference operations. For five-axis simulation, additional scan conversion of five-axis milling tools at every tool instance is also required. However, the scan conversion of five-axis milling tools is optimized via the bounding ellipse scan conversion algorithm which reduces the processing time difference between three- and five-axis milling simulation. To further accelerate the simulation, a control on instances of motion is implemented to reduce the computation of scan conversions and regularized Boolean difference operations. Furthermore, graphical display of simulation is implemented by using local update to achieve optimal performance.

6.2.1 Instance of motion control

Simulations of three- and five-axis milling were implemented as described in Chapter 4. To achieve higher performance, while sacrificing accuracy, a control over the instances of tool motion is implemented. The control reduces the number of instances computed in Equation (4.2) by dividing n by 2^k, where k is an adjustable positive real value. For $k=0$, all instances of motion are simulated. For very large values of k, the value of n drops to less than 1 thus only instances at the *CL* points are simulated, which is useful for previewing tool paths. Figure 6.3 demonstrates several results at different k values (0, 1, 2, and 4) and the performance statistics are given in Table 6.1. The lower the k value, the better the simulation results. Note that a severe performance penalty is observed between the best simulation mode ($k=0$) and the second best simulation ($k=1$) without gaining obviously better result. Therefore, all further examples of milling simulation and verification are performed at $k=1$.



(a) $k = 0$ (b) $k = 1$



(c) $k = 2$ (d) $k = 4$

Figure 6.3 Three-axis simulation results of several setting of k values

Table 6.1 Performance of three-axis milling simulation at several k values

	$k=0$	$k=1$	$k=2$	$k=4$
number of instances	*14,664*	*7,713*	*4,187*	*3,969*
processing time (sec.)	*398*	*211*	*114*	*109*
instances/sec.	*36.844*	*36.554*	*36.510*	*36.413*
seconds/inst.	*0.027*	*0.027*	*0.027*	*0.027*

6.2.2 Graphical display of milling simulation

The graphical display of milling simulation was implemented using the image space display method. Although the image is not rotatable during the simulation process, thus implementation achieves high efficiency for graphical display. The contour display method can be implemented for in-process dynamic viewing, however, it will increases the computational load due to the updating of contours affected by regularized Boolean difference operations. Also, the display quality is not as good as the image space display method. Therefore, the contour display method was implemented as a post-process to visualize the finished part from arbitrary viewing directions.

To optimize the display task in the image display method, two image arrays are implemented in addition to the double-buffer support of the SGI hardware. The first array is used to record the image of the workpiece and fixtures, and is mapped directly to the screen buffer for efficient global image update. The other array is used to record the tool image for local update. These two arrays is updated simultaneously with regularized Boolean difference operations, however, only the tool image is displayed superimposed over the previous workpiece image in the front buffer to effect the simulation process, thus no swapbuffers function call is needed. To further reduce the graphic tasks, the entire simulation update can be disabled, and graphical display can be achieved as a post process.

Milling verification is implemented based on the simulation function with additional minimum-distance calculations to obtain the dimension of milling error. The minimum norm projection and perpendicular method are both coded to compute the milling error because they are similarly accurate and efficient.

6.3 Performance Enhancements

Several unique algorithms including a distance-square color lookup scheme, an intial projection point lookup data structure, and a regional search method were formulated to enhance the robustness and efficiency of the implementation.

6.3.1 Distance-square color lookup scheme

The projection function returns a surface near-point of a given space point, so the distance is computed by the length of the vector between them. Since the minimum-distance calculation is used intensively in verification, distance squares, instead of distances, are used to check the depth of cut hence eliminating the square root computations to find an actual distance. An error range lookup table is constructed to obtain proper color coding for a given distance square. Assuming a range of interest is given by $[r_b, r_l]$, tolerance by $[t_b, t_l]$, and n hues for overshoot and undercut errors, respectively. The error range lookup table $\boldsymbol{R}[i]$ is computed by,

$$\boldsymbol{R}[i] = \begin{cases} 0 \le i < n, & (sign)\left(r_l + i \times \dfrac{(t_l - r_l)}{n}\right)^2 \\ i = n, & (sign)\, t_b^2 \\ n < i \le 2n, & (sign)\left(t_b + (i-n) \times \dfrac{(r_b - t_b)}{n}\right)^2 \end{cases} \tag{6.2}$$

where $(sign)$ denotes the sign of the variable before square operation. Therefore, given a distance square, the corresponding index i in $\boldsymbol{R}$ can be found by using a binary search in $\boldsymbol{O}(\log n)$ time [Manber, 1989], thus index i depicts a color in the color lookup table to graphically encode the milling error on the part surface.

6.3.2 Initial projection point data structure

In the projection methods, the Newton/Raphson method [Kincaid and Cheney, 1991] is used to search for the surface-near point in a 2D surface parameter space. The robustness and efficiency of the search are heavily

dominated by the choice of initial point. Hence a voxel data structure is implemented to supply initial surface point that are in the vicinity of the given space point.

The initial projection point lookup algorithm first discretizes the design surfaces into a set of points that are equally spaced in the parameter space, i.e., constant increments in u- and v-parameters. The surface points are then stored in a voxel data structure [Foley et al., 1990, Chang and Goodman, 1991] that is also constructed in the dexel coordinate system. Thus, for a given space point, the index of the corresponding voxel can be determined by direct computation and the surface points stored in the voxel are then compared with the space point. The nearest surface point is used for the initial point of projection functions.

Assuming a surface is defined by a n by m control point net, the surface is discretized into $(nr)\times(mr)$ points, where r is a resolution factor to adjust the density of surface point sampling; in practice, $r=2$ is general sufficient for typical surfaces. The total number of the surface points, denoted by N_p, is therefore nmr^2. The coordinate values of the surface points are computed in the world coordinate system. Denoting the set of surface points by $\boldsymbol{S}^w$, the dexel coordinate surface point set, denoted by $\boldsymbol{S}^d$, is calculated via coordinate system transformation using Equation (2.3).

The voxel data structure is constructed corresponding to the number of points in $\boldsymbol{S}^w$, i.e., the larger the number of surface points, the smaller the unit voxel. Assuming a bounding box of the design surfaces, denoted by $\boldsymbol{B}$ as shown in Figure 6.4, is computed in the dexel coordinate, the unit size of a voxel in the data structure is determined by,

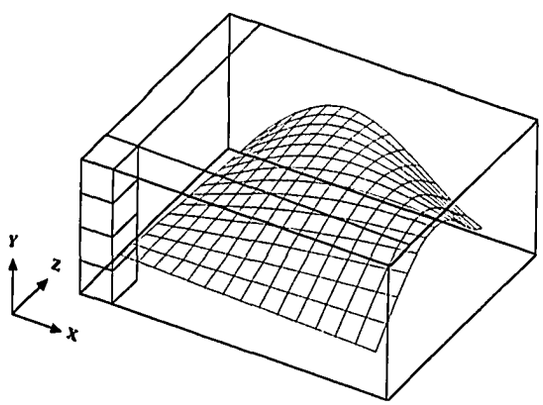


Figure 6.4 Voxel data structure for initial surface point lookup

$$v = \frac{B_{max}}{k\sqrt{N_p}} \tag{6.3}$$

where, B_{max} is the maximum dimension in x-, y- and z-coordinates of the bounding box, and k, called discretization factor, is a control factor to refine of the unit dimension of a voxel. The size of the voxel data structure, denoted by V, is then determined by,

$$(V_x, V_y, V_z) = \left(\frac{B_x}{v}, \frac{B_y}{v}, \frac{B_z}{v}\right) \tag{6.4}$$

The voxel data structure is implemented in a 3D array of variable type *long*, such that each cell of the 3D array points to a cell of a 1D array that records the surface points. The surface point array is implemented in a data structure defined by,

```
typedef struct {
    int sid;
    float u, v;
    float P[3];
    long next;
}
```

where *sid* denotes the surface index, u and v are the surface parameters, P contains the surface point coordinate values, and *next* points to the next surface point. Thus a linked list data structure is formed. Each cell of the 3D voxel data structure points to a linked list whose nodes are recorded in the 1D data array.

After the voxel data structure is constructed, the surface points in S^w are converted to the dexel coordinate system and installed in the data structure by,

$$(I_x, I_y, I_z) = \left(\frac{S_x^d - B_{lx}}{v}, \frac{S_y^d - B_{ly}}{v}, \frac{S_z^d - B_{lz}}{v}\right) \tag{6.5}$$

where I is the address of a corresponding cell in the 3D data structure, and B_l is the lower corner point of the surface bounding box. If the cell of address I is not empty, i.e., it contains other points, the new point pushes the existing points and becomes the first node of the linked list. Finally, all points in S^w are organized in the voxel data structure to provide fast lookup of initial point for projection function.

In some cases the surface points set $\boldsymbol{S}$ may not be dense enough to provide an initial point, i.e., many cells in the voxel data structure are empty. Therefore, if direct retrieval of intial points fails, i.e., the computed address records no points, the adjacent cells must also be checked until a non-empty cell is found. The search process is time-consuming and violates the objective of the voxel data structure. To solve the problem, every point in the voxel data structure is additionally inserted into its adjacent empty cells up to h-levels. The parameter h is called an expansion factor. In other words, all empty voxels that are less than h voxels away from a given voxel g are assigned to the value of g. Consequently, the neighborhood information is also recorded in the voxel. The value of h can be set to a large value to ensure that all empty cells that are near to the design surfaces are filled with initial surface points. However, it generally takes longer time for the pre-processing task and may waste computation time for finding projection of points that are far away from the surface. Experiments showed that expansion factor $h=4$ is sufficient for surfaces that are discretized using resolution factor $r=2$ and discretization factor $k=2$. However, these factors are very sensitive to the shape of design surfaces, higher settings of these factors can enhance the robustness, at the expense of computation time, for general NURBS surfaces.

6.3.3 Regional search of projection function

The formulation of projection function, perpendicular or minimum norm projection, includes a $(\boldsymbol{P}\text{-}\boldsymbol{Q})$ term, where $\boldsymbol{P}$ is the space point to be projected and $\boldsymbol{Q}$ is the surface projection. This term can be relatively large if $\boldsymbol{P}$ is far from the design surfaces, in which case the refinement step $(\Delta u, \Delta v)$ will be correspondingly large and the Newton's method may fail to find the correct projection point. To overcome the problem, a regional search algorithm is implemented to dynamically adjust the refinement step.

The regional search algorithm utilizes the knot property of NURBS surfaces to localize the search in a knot span region. Denoting the u-knot vector of a surface by $\boldsymbol{U}$ and the v-knot vector by $\boldsymbol{V}$, for a given surface point at (u_0, v_0), the corresponding knot span region is defined at $\boldsymbol{U}_i \leq u_0 < \boldsymbol{U}_{i+1}$ and $\boldsymbol{V}_j \leq v_0 < \boldsymbol{V}_{j+1}$, as illustrated in Figure 6.5. Denoting the knot span ranges in u- and v-parameter by $\delta u = \boldsymbol{U}_{i+1} - \boldsymbol{U}_i$ and $\delta v = \boldsymbol{V}_{j+1} - \boldsymbol{V}_j$, respectively, the refinement step in the Newton's method is limited by,

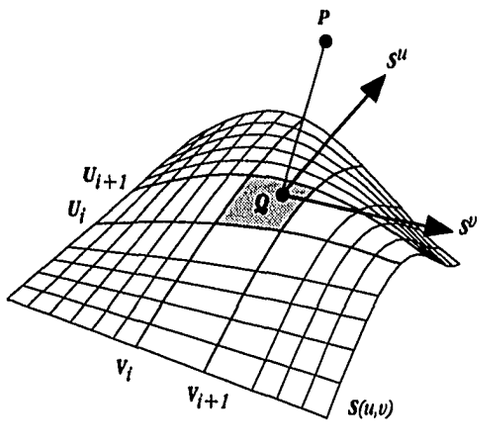


Figure 6.5 Regional update of projection function

$$\left(-\frac{\delta u}{r}, -\frac{\delta v}{r}\right) \leq (\Delta u, \Delta v) \leq \left(\frac{\delta u}{r}, \frac{\delta v}{r}\right) \tag{6.6}$$

where $r \geq 1$, is called a refinement factor, and is used to further reduce the step size of a search. Since the knot span region corresponds roughly to the complexity of the surface shape, Equation (6.6) slows down the search at narrow knot spans but has less effect on wide ones. The overall convergence speed is, for refinement factor $r=2$, slightly reduced but the accuracy and robustness is enhanced. The refinement factor can also be dynamically determined with respected to the local surface curvature, however, such computation reduces the efficiency and hence is not considered in this software implementation.

To further enhance the accuracy of the projection function, each search step in the Newton's method is restricted not to cross the current knot span region in a single search step, i.e., the next search point can at most move to the boundary of the knot span then restart search from the next knot span area. Since adjacent knot spans can vary widely in the span range, this restriction ensures no knot span region is missed with large step size.

6.4 An Example of Milling Verification and Path Correction

A five-axis finish cutting example utilizing the perpendicular projection method and the distance square color coding scheme is demonstrated in Figure 6.6, in which sequential images depict the milling simulation and in-

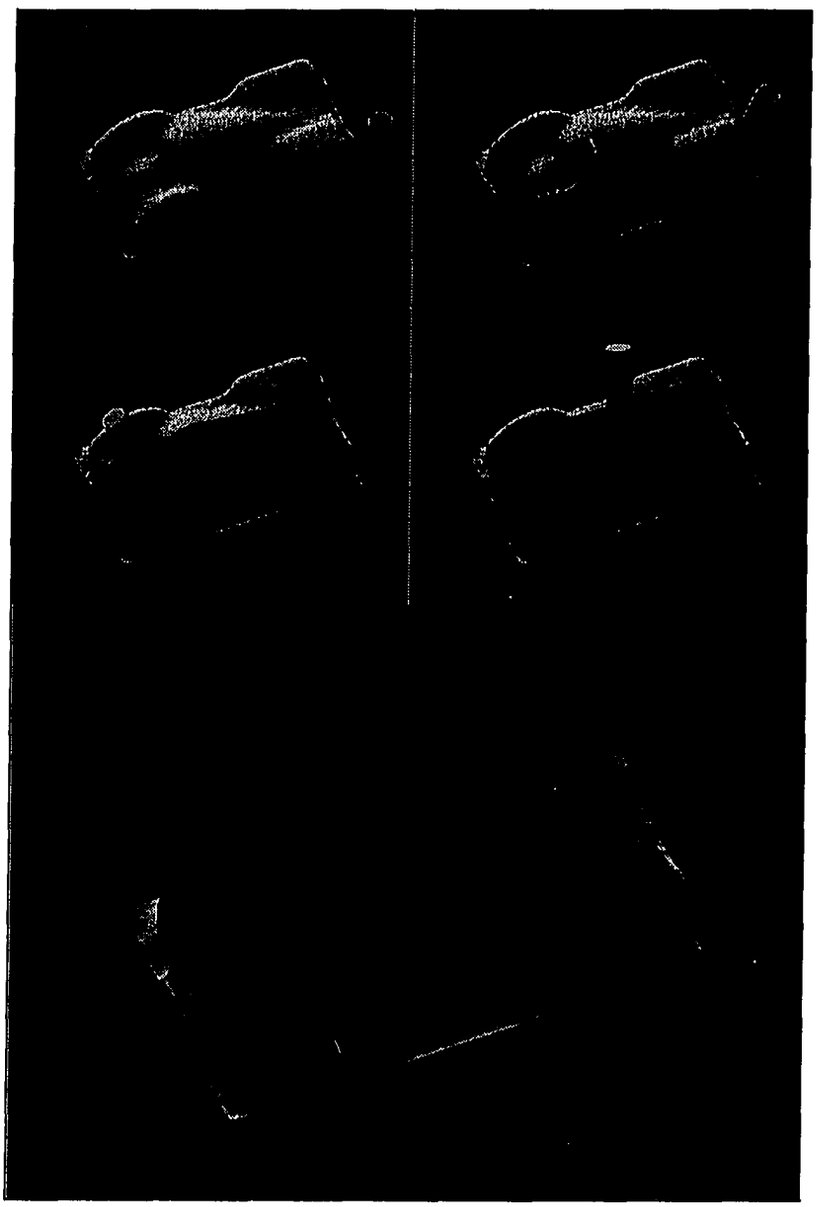


Figure 6.6 Finish cutting and verification process of the car model

process verification. The running time for this example is 2,825 seconds for 7,713 instances of motion, or 2.73 instances per second. Path correction of the model is demonstrated in Figure 6.7, which took 5,041 seconds for 7,713 instances of motion, or 1.53 instances per second.

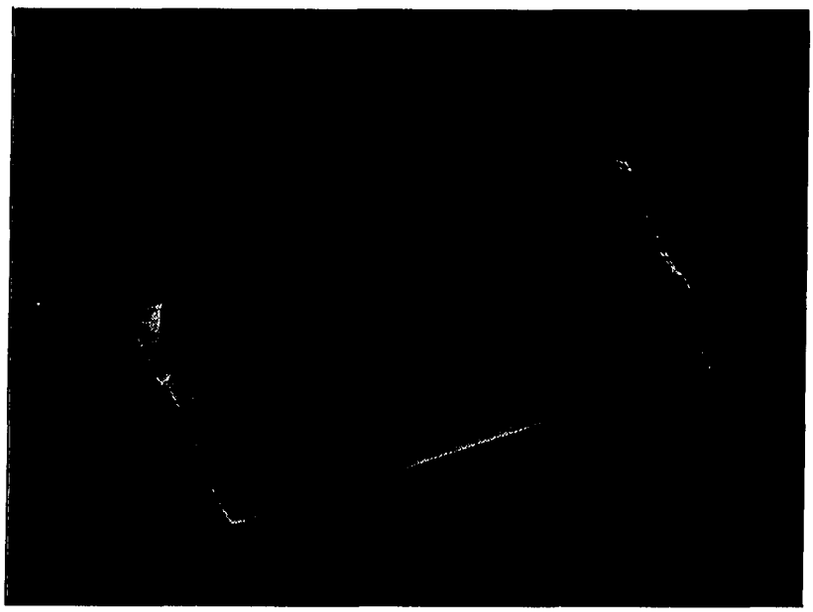


Figure 6.7 Gouge elimination of the car model

7. CONCLUSION AND FUTURE WORK

A discrete dexel NC verification system is presented in the dissertation. The verification algorithm combines the advantages of the discrete vector intersection approach and the spatial-partitioning representation method, and thus is capable of efficiently simulating and precisely verifying NC milling tool paths. Furthermore, tool path correction algorithms are developed based on the verification results.

The dexel representation of solid geometry developed in this dissertation is not merely an extension of the data structure proposed in Van Hook [1986]. A unique construction of the dexel coordination system which is independent of the screen coordinate system is introduced. The geometric definition of design surfaces, workpiece, fixtures, and milling tools are transformed into the dexel coordinate system. Workpiece, fixtures, and cutting tool are further decomposed into sets of dexels thus enabling fast shape manipulation via regularized Boolean set operations. The implementation of contour generation algorithm described in Chapter 3 further distinguishes this dexel representation as view independent. The contour generation algorithm exploits the highly organized data structure of dexel representation to efficiently connect all dexel points to form constant x- or y-contours. The algorithm is also capable of generating multiple contours, if they exist, at any constant x- or y-plane. Therefore, the contour display method effectively transforms dexels into a set of parallel planar contours that can be freely translated and rotated.

At current implementation, the contour display method can not display object at the quality level of the image display method. The gaps between contours become obvious when an object is viewed closely. However, this drawback can be eliminated by generating a triangular mesh among the contours [Meyers and Skinner, 1992, Ekoule et al., 1991]. Again, the property of highly organized dexel representation can be exploited to simply the triangulation procedure for display efficiency. One opportunity for future research is to implement a triangulation algorithm to generate polyhedral model of the dexel representation, thus providing realistic five-axis NC milling simulation which can be viewed realistically from any direction while the process is taking place.

The discrete dexel NC milling verification algorithm presents a major breakthrough in the field of dimensional milling verification: it integrates the advantages of spatial partitioning approach and discrete vector intersection approach and solves major drawbacks in both methods. Therefore, very efficient NC simulation and high precision dimensional verification are achieved. This algorithm exploits the fact that all dexel points of any dexel-based object lie exactly on the object surface, thus the distance between a dexel point and design surfaces is exactly the milling error. Color-coded milling errors are displayed on the milled part and hence are very realistic and useful for further finishing processes.

The objective of an NC verification system is to depict potential problems of tool paths. However, tool paths, especially for complicated parts and five-axis motions, are difficult to manually modify. To overcome this problem, tool path correction algorithms are implemented to assist NC programmers in tool path modification. The automatic tool path correction is achieved via the reduction of intersection volume algorithm and the tool assembly sensing algorithms. The former eliminates detected gouge by computing a guide vector to translate the tool location so that the cut depth is reduced to meet a specified tolerance limit. The latter introduces a sensor-equipped tool assembly model to the NC milling simulation process. Thus collision is detected by checking the location of each sensor against the workpiece model. Collisions are eliminated by two types of remedial actions: offsetting tool location and pivoting tool axis. The integration of the gouge- and collision-elimination algorithms in an iterative combination provides a comprehensive automatic tool path correction system.

The path correction algorithms also present a unique method for five-axis tool path generation: generating initial five-axis tool paths then modifying the paths to be gouge- and collision free. The CL data of the initial five-axis tool path is generated from general three-axis tool path generation algorithms[Huang and Oliver, 1992] and the tool axis vectors are obtained from the surface normal vectors at the cutter contact points. The initial tool paths are then modified by using the tool path correction algorithm to eliminate gouge and collision. The advantage of this tool path generation approach is that the tool path modification can be performed after a the workpiece setup is determined, i.e., the size and shape of the workpiece and the types and locations of fixtures is decided. Therefore, no additional check is necessary and the tool paths are ready for immediate application.

In summary, the dissertation presents a complete system for NC milling tool path simulation, verification, correction, and generation that no previous research or commercial software has achieved. Furthermore, the contour display method solves a major problem in the spatial partitioning approach, i.e., view dependency. Thus this approach is very suitable for other manufacturing and general engineering models.

In addition to the generation of triangulation mesh over dexel-based models, future research will be conducted in machine dynamic analysis. This research will be based on the volume and geometry of the material removed from the workpiece at each instance of tool motion, which is computed by using the dexel-based Boolean intersection operation.

REFERENCES

Atherton, P. R., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry," *Computer Graphics, Proceeding of SIGGRAPH '83*, Volume 17, Number 3, July, pp. 73-82

Barnhill, R. E. and Kersey, S. N., 1990, "A Marching Method for Parametric Surface/Surface Intersection," *Computer Aided Geometric Design*, No. 7, pp. 257-280

Baron, R. J. and Higbie, L., 1992, *Computer Architecture*, Addison Wesley Publishing Company, New York, NY

Chang K. Y. and Goodman E. D., 1991, "A Method for NC Tool Path Interference Detection for A Multi-Axis Milling System," *ASME Control of Manufacturing Process*, DSC-Vol. 28/PED-Vol. 52, pp. 23-30

Drysdale, R. L. and Jerard R. B., 1987, "Discrete Simulation of NC Machining," *Proceedings of 3rd ACM Annual Symp. on Computational Geometry*, pp. 126-135

Ekoule, A. B., Peyrin, F. C., and Odet, C. L., 1991, "A Triangulation Algorithm From Arbitrary Shaped Multiple Planar Contours," *ACM Transactions on Graphics*, Vol. 10, No. 2, April, pp. 182-199

Foley, J. D., van Dam, A., Feiner, S. K. and Hughes, J. F., 1990, *Computer Graphics Principles and Practice*, Addison-Wesley Publishing Company, New York, NY

Hoffmann, C. M., 1989, *Geometric and Solid Modeling, An Introduction*, Morgan Kaufmann Publishers, Inc., San Mateo, CA

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W., 1992, "Surface Reconstruction from Unorganized Points," *Computer Graphics, Proceedings of SIGGRAPH '92*, Vol. 26, No. 2, July, pp. 71-78

Huang Y. and Oliver J. H., 1992, "Non-Constant Parameter NC Tool Path Generation on Sculptured Surfaces," *Proceedings of ASME International Computers in Engineering*, August, Vol 1, G.A. Gabriele, ed., pp. 411-419

Jerard, R. B., Hussaini, S. Z., Drysdale, R. L., and Schaudt, B., 1989, "Approximate Methods for Simulation and Verification of Numerically Controlled Machining Programs," *The Visual Computer*, Vol. 4, pp. 329-348

Kawashima, Y., Itoh, K., Ishida, T, Nonaka, S., and Ejiri, K, 1991, "A Flexible Quantitative Method for NC Machining Verification Using a Space-Division Based Solid Model," *The Visual Computer*, Vol. 7, pp. 149-157

Kincaid, D. and Cheney, W., 1991, *Numerical Analysis, Mathematics of Scientific Computing*, Brooks/Cole Publishing Company, Pacific Grove, CA

Leighton, F. T., 1992, *Introduction to Parallel Algorithm and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, San Mateo, CA

Manber, U., 1989, *Introduction to Algorithms A Creative Approach*, Addison-Wesley Publishing Company Inc. New York, NY

Menon, J. P. and Robinson, D. M., 1992, "High Performance NC Verification via Massively Parallel Raycasting: Extension to New Phenomena and Geometric Domains," PED-Vol. 59, *ASME Concurrent Engineering*, pp. 179-194

Meyers, D., Skinner, S., and Sloan, K., 1992, "Surfaces from Contours," *ACM Transactions on Graphics*, Vol. 11, No. 3, July pp. 228-258

Mortenson, M. E., 1987, *Geometric Modeling*, John Wiley & Sons Inc., New York, NY

Narvekar, A., Huang, Y. and Oliver, J. H., 1992, "Intersection of Rays with Parametric Envelope Surfaces Representing Five-Axis NC Milling Tool Swept Volumes," *Proceedings of ASME Advances in Design Automation* Vol 2, D. A. Hoeltzel, ed., pp. 223-230

Oliver, J. H., 1986, Ph. D. Dissertation, *Graphical Verification of Numerically Controlled Milling Programs for Sculptured Surface Parts*, Michigan State University, MI

Oliver, J. H., 1992, "Efficient Intersection of Surface Normals With Milling Tool Swept Volumes for Discrete Three-Axis NC Verification," *Journal of Mechanical Design*, June Vol. 114, pp. 283-287

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

Piegl, L., 1987, "Curve and Surface Constructions Using Rational B-Splines," *Computer-Aided Design*, Vol. 19, No. 9, November pp.485-498

Piegl, L., 1991, "On NURBS: A Survey," *IEEE Computer Graphics and Applications*, January, pp. 55-71

Pegna, J. and Wolter, F.-E., 1990, "Designing and Mapping Trimming Curves on Surfaces Using Orthogonal Projection," *Proceedings of ASME Advances in Design Automation*, Vol. 1, Computer Aided and Computational Design, DE-Vol. 23-1, Sept., pp. 235-245

Saito, T. and Takahashi, T., 1991, "NC Machining with G-buffer Method," *Computer Graphics, Proceedings of SIGGRAPH '91*, Volume 25, Number 4, July, pp. 207-216

Sambandan K. and Wang, K. K., 1989, "Five-Axis Swept Volumes for Graphic NC Simulation and Verification," ASME Advances in Design Automation, DE-Vol 19-1, Sept.

Samet, H. 1989, "Implementing Ray Tracing with Octrees and Neighbor Finding," *CGVIP*, 46(3), June, pp. 367-386

Sungurtekin, U. A. and Voelcker, H. B., 1986, "Graphical Simulation & Automatic Verification of NC Machining Programs," *Proceedings of 1986 IEEE International Conference of Robotics and Automation*, April pp. 156-165

van Hook, T., 1986, "Real-Time Shaded NC Milling Display," *Computer Graphics, Proceedings of SIGGRAPH '86*, Volume 20, Number 4, August, pp. 15-20

Voelcker, H. B. and Hunt, W.A., 1981, "The Role of Solid Modeling in Machine-Process Modelling and NC Verification," *SAE Technical Paper* No 810195, Feb.

Wang, W. P., and Wang, K. K., 1986, "Geometric Modeling for Swept Volume of Moving Solids," *IEEE Computer Graphics and Applications*, December, Vol.6, No. 12, pp. 8-17