1-1-1993

An interactive graphical simulation of CNC milling

Craig T. Muncaster
Iowa State University

An interactive graphical simulation of CNC milling

by

Craig T. Muncaster Jr.

A Thesis Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Mechanical Engineering
Major: Mechanical Engineering

Iowa State University
Ames, Iowa

1993

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

In industry today, efficiency is a major goal for manufacturing of any product. Computer simulation is an increasingly important method of achieving this goal. Using computer simulation, it is possible to predict what will happen in the real world before a prototype is built. Simulation can allow a user to visualize a process before it happens. Simulation, as demonstrated in this thesis, can also create and test device control code for any type of computer controlled machine.

This thesis describes the initial stages of development of an interactive graphical mill simulation program developed for the DynaMyte 4400 computer controlled milling machine. The software uses "solid" models to represent the relevant objects in the milling process, and displays the interaction of these models through animated computer graphics. Several considerations that must be addressed in the development of this software are: 1) modelling of the equipment, 2) display of the models, 3) material removal and collision algorithms, and 4) the user interface design. As a final verification step, the simulation software was tested by running several simulations and then performing the actual milling operation on the DynaMyte 4400.

There are several simulation and verification software packages available for Numerically Controlled (NC) milling operations. This simulation was developed to be used in conjunction with a verification program to allow the user to perform a complete pre-process test of any milling operation. When both programs are used together, the objective is to eliminate verification runs on the mill and eliminate the need to take the milling machine out of production except for tool changes and CNC code down loading. This capability will reduce the overall time required to program the milling machine for a new operation.

2 LITERATURE REVIEW

Originally NC program verification was accomplished by "proofing" runs. The milling machine was programed and set up with a work piece of soft inexpensive material such as wood, plastic or wax. After the milling process the part dimensions were measured to verify the quality of the NC program. Changes to the program would require additional "proofing" runs before an acceptable part was produced. This process was very labor intensive and resulted in non production time for the mill. Thus, there is a growing demand for ways to verify and simulate NC programs without using the milling machine.

2.1 Verification and Simulation

In general, there are two methods to verify and simulate NC programs off line using computers: the constructive solid geometry method (CSG) and the "view-based" method.

Using the constructive solid geometry method, the work piece and tool are modeled using combinations of primitive solids (i.e., blocks, cylinders, etc.). The final production part is then created by a boolean subtraction of the tool from work piece solid for each tool position along the tool path. Fridshal (1982) at General Dynamics modified the TIPS solid modeling package to do NC simulation using the CSG method. This process yields an exact representation of the part being produced but is very computationally expensive. To increase the efficiency of the CSG method, several approximate simulation methods were developed.

Chappel devised a "point-vector" approach to NC simulation [8]. The surface of the work piece was approximated by a set of points with direction vectors normal to the surface associated with them. The vectors extend to the boundary of the stock material or

until they intersect another surface of the finished part. The length of the vector is reduced whenever it intersects the tool envelope.The length of the final vectors correspond to the amount of excess material or the depth of under cutting after the process is over.

Oliver developed a method similar to Chappel except the surface point approximations were developed by projecting each pixel on the screen back onto the work piece [2]. After the surface point approximation mesh was created the process proceeded the same way as Chappel's method did.

Jerard introduced a method similar to the previous two methods except the direction vectors are parallel to the longitudinal axis of the cutting tool [3]. This choice for the direction vectors simplifies the intersection calculations of the tool and work piece and therefore increase the computational efficiency of the process.

The "view-based" method defines vectors (or rays) normal to the computer screen that intersect the work piece. These intersections are calculated and stored as depth values in the z-buffer. The z-buffer is an extended frame buffer that contains the color and z-depth of each pixel on the computer screen [11]. When these screen vectors are intersected by the tool moving along the tool path the color of the pixel is changed depending on the position relationship between the work piece and the tool.

Saito and Takahashi developed the concept of the G-buffer (geometric buffer) [7]. The G-buffer is an array that contains geometric information of the work piece and tool, such as world coordinate z, normal unit vectors, object/patch identifiers and u-v patch coordinates. This method separates geometric procedures (i.e., scan conversion and hidden line removal) from other procedures (i.e., shading, texture mapping and enhancement). This separation of tasks makes it easier for both path generation and process simulation instead of just process simulation as with other "view-based" methods.

Van Hook describes the use of "dexels" or depth elements in verifying N.C. programs [1]. Each pixel is given a near z depth, a far z depth, a color and a pointer. One dimensional boolean operations are then performed between the dexels of the work piece and the dexels of the tool by simple transformations of the entities.

3 SIMULATION SOFTWARE

This chapter presents simulation software developed in this thesis using a discrete solid geometry approach similar to the Jerard's method discussed in the previous chapter [3]. Included in this chapter are: 1) a summary of requirements for viable simulation in a graphical environment, 2) a basic explanation of some computer graphic concepts, 3) a description of the user interface controls, 4) the method used to simulate the material removal, and 5) the algorithm used for the detection of collisions.

3.1 Graphical Simulation Requirements

There are several capabilities needed to enable graphical simulation of a NC milling process. These fall into four major categories: display, user interface, material removal and collision detection. The display of the milling process in a graphical environment allows the user to see what will occur when a given NC code is used. This is critical for the detection of problems that might be in the NC code or in the positioning of fixtures in the work cell without using the mill itself. If a problem exists in either of these areas, the user must be able to correct the problem. The user interface accomplishes this task by allowing the user to change certain milling parameters while the simulation is in progress. The calculation of the amount of material removal and the detection of collisions are important to assure a realistic and accurate simulation of the milling process.

3.2 Display

The display of the machining process is accomplished by defining object models, camera positions, and lighting positions. Object models are three dimensional representations of objects, including the milling machine, work piece, and fixtures. The object models can be represented as polygons(facets), parametric curves and surfaces, or a combination of both. Polygons were used for the representation of the solid objects in the simulator because they are less complex and therefore have higher rendering update speeds than do parametric surfaces. The objects used in the mill simulator were created using Structural Dynamics Research Corporation's I-DEAS, an engineering design and analysis software package, and are represented in files using the polygonal Brigham Young University (BYU) format [10]. The BYU format is a simple listing of three dimensional vertices for each facet and how these vertices are connected. Using the BYU format for the storage and representation of the model data allows for fast calculation and display of the milling process and more compact data representation.

Similar to filming a movie, the positioning of the cameras and lights must be considered to display the solid model data structures. Cameras are defined by a look from point and a look at point which determines the viewing vector. Each viewing vector is represented by a 4x4 viewing matrix. The geometric model information is transformed by the viewing matrix and the perspective matrix, which contains the field of view for each camera. The computer uses the transformed model to generate a graphical image. The user interface enables changing of the camera positions so the user can follow the process from different viewing points. The lights are defined by position and lighting direction which, along with a material property definition for each object, determines how the facets are to be colored and shaded by the computer [12].

3.3 Interface

The graphical user interface consists of input commands from text lines, buttons and sliders that continuously control program parameter. Buttons are areas on the computer screen where the cursor can be positioned and the mouse button depressed to execute a specific command, typically on/off functions or text input. Sliders, as the name implies, enables continuous control of program parameters through use of the cursor and computer screen. Dynamic control of these parameters coupled with real time display allows the user to see the changes as they happen. Text lines are areas on the screen where commands are input via the keyboard.

The graphical user interface for the mill simulator contains four main interface areas, as shown in Figure 3.1. The upper left of the screen contains the viewport where the process is displayed. The upper right of the screen displays and controls the milling parameters, such as the NC file being tested, the size and position of the workpiece and the size of the tool bit. The lower left of the screen contains several sets of sliders to control the camera positions and fixture orientations. The lower right portion of the screen is made up of buttons that allow the user to select which set of sliders are active in the lower left corner, which camera the process is viewed from, and which fixtures are being used in the simulation. A more detailed description of the interface controls can be found in the users manual in the Appendix.

Since there are a the large number of buttons and sliders, there is a need to coordinate the colors of each to allow for easier understanding of which type of input each button requires. Yellow areas of the screen represents toggle buttons that input only two commands, on or off. Purple areas of the screen are made up of textual input buttons that the user must key in the command desired. Black areas are for the display of information only and do not perform any program input or control functions.

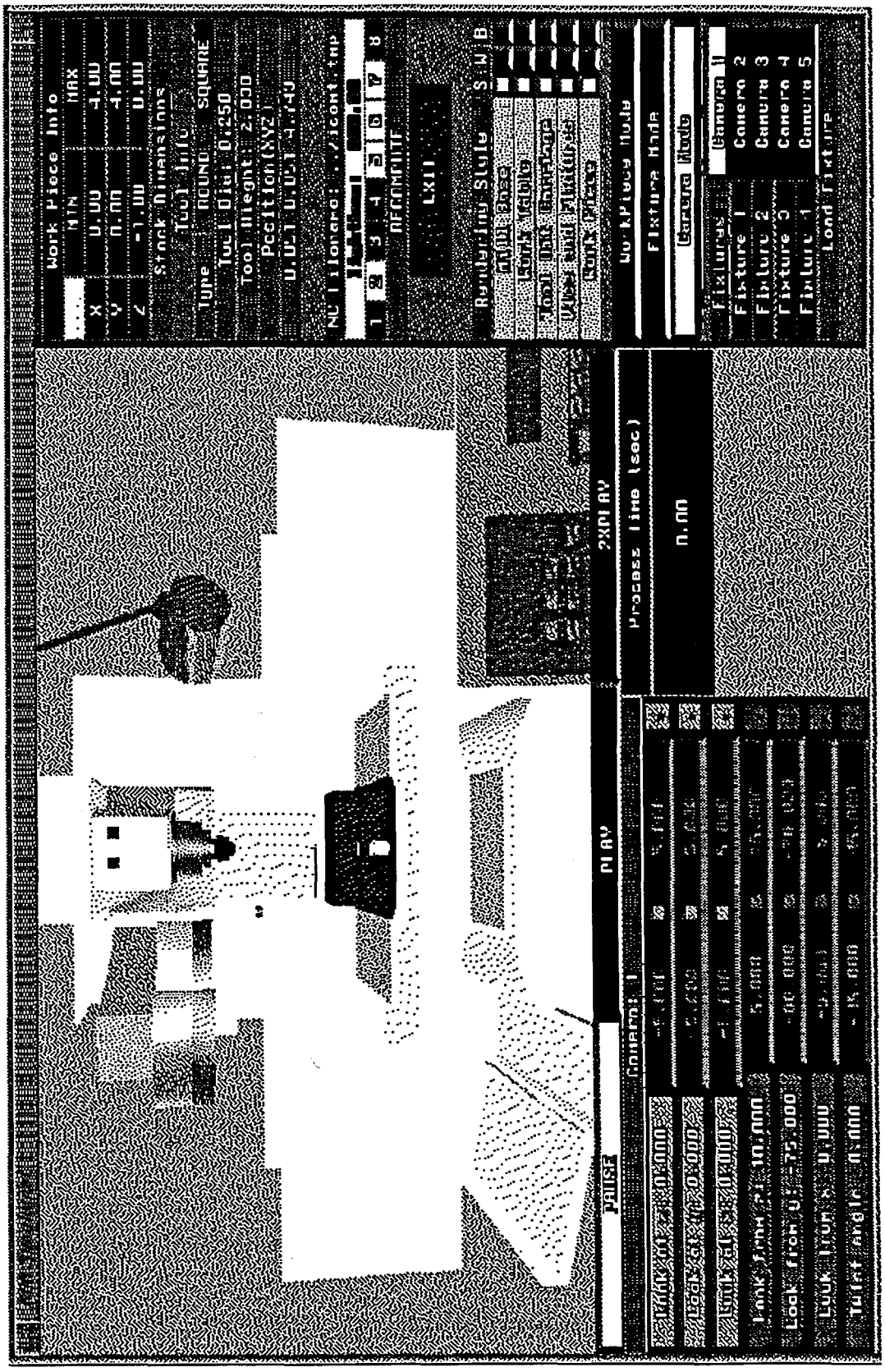


Figure 3.1 Graphical screen interface

3.4 Material Removal

Material removal was simulated using a method similar to the G-buffer method discussed earlier [7]. A two dimensional mesh was defined over the work piece material and the z-value of the workpiece surface was calculated for each node as shown in Figure 3.2. The Z-value is then loaded into a two dimensional z-buffer array where each array index corresponds directly to an x-y position on the surface of the block. This approach minimizes data storage and allows control over the accuracy and computation demands of the material removal process through resolution of the grid. The block mesh can be changed by a load factor that specifies the number of nodes per inch on the block. Verification can be accomplished by setting the load factor to the desired accuracy. This can create memory and display problems for the computer if the work piece is large.

Since the milling machine has only three axis of motion, calculation of the material removal was accomplished by creating a swept area or 2-dimensional tool envelope on the workpiece surface. The tool envelope is the area that the tool will traverse when moving from one control point to the next (Figure 3.3). If a surface node is inside the tool envelope, the nodes z-value is checked to see if it is greater then the z-value of the tool. If it is, a new z-value is calculated for the node and the mesh array is adjusted accordingly. A tolerance can be incorporated in to the calculation of the distance to the tool bit to account for any variation in the tool position. Since the position of the tool is known and the position of each work piece node is dependent on the array indices, there is no need to check every node in the array, rather only the nodes that are in the general area of the tool envelope require checking. This process increases the efficiency of calculating the material removal.

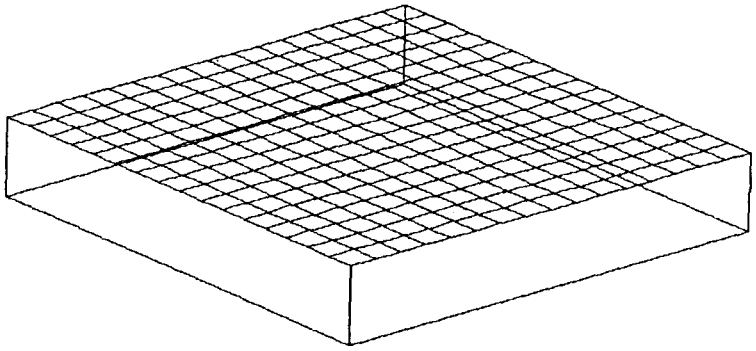


Figure 3.2 Mesh superimposed onto block surface

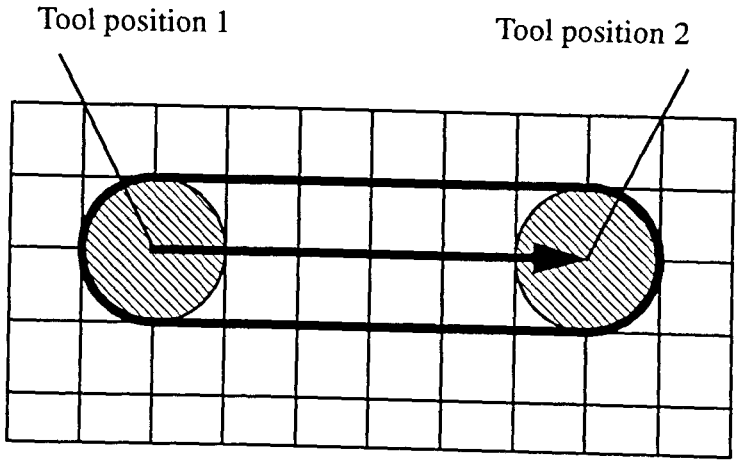


Figure 3.3 2-dimensional envelope created by motion of tool

3.5 Collision Detection

The detection of collisions is accomplished by three dimensional vector analysis. Since all other motions of the mill, such as tool changes, are controlled internally by the machine, collision detection is only performed for the mill fixtures, tool, and tool bit collar. The tool collar is represented by polygons, therefore the edges of each polygon can be represented by sets of parametric vector equations.

$$x = x_i + a_i t \qquad \textbf{(EQ 1)}$$

$$y = y_i + b_i t \qquad \textbf{(EQ 2)}$$

$$z = z_i + c_i t \qquad \textbf{(EQ 3)}$$

with x_i, y_i, z_i representing each vertex of the polygon and a_i, b_i, c_i the x, y, z directional components of each edge. The t is a parametric variable that ranges from zero to one.

Each polygon of the fixtures can be represented by a planar equation of the form,

$$P = N_{xj}(x - x_{pj}) + N_{yj}(y-y_{pj}) + N_{zj}(z-z_{pj}) \qquad \textbf{(EQ 4)}$$

where N_{xj}, N_{yj}, N_{zj} are the components of the normal vector for each polygon, which is determined by the cross product of two edge vectors. The symbols x_{pj}, y_{pj}, z_{pj} represent any vertex of the polygon being investigated.

By substituting equations 1 through 3 into equation 4 and solving for t, an equation results of the form,

$$t = (d - N_x x_i - N_y y_i - N_z z_i) / (a_i N_{xj} + b_i N_{yj} + c_i N_{zj}) \qquad \textbf{(EQ 5)}$$

where

$$d = N_{xj} x_{pj} + N_{yj} y_{pj} + N_{zj} z_{pj} \qquad \textbf{(EQ 6)}$$

Equation 5 is solved for each edge of the collar and tool bit polygons to determine if that edge intersects the planes represented by the fixture polygons. If the value of t is between zero and one then the collar edge intersects a plane representing a fixture polygon (Figure 3.4).

To determine if the collar edge has an intersection inside the polygon representing the fixture, three vectors are calculated from the intersection point to each vertex of the polygon. The angle between each vector is then calculated by taking the dot product between each vector.

$$V_{i1} \text{ dot } V_{i2} = |V_{i1}| |V_{i2}| \cos(\Theta_{12}) \quad \text{(EQ 7)}$$

$$V_{i2} \text{ dot } V_{i3} = |V_{i2}| |V_{i3}| \cos(\Theta_{23}) \quad \text{(EQ 8)}$$

$$V_{i3} \text{ dot } V_{i1} = |V_{i3}| |V_{i1}| \cos(\Theta_{31}) \quad \text{(EQ 9)}$$

The procedure is to solve equations 7-9 for the angles between each vector and add them together, a collision is indicated if the three angles add up to 360 degrees. As shown in Figure 3.5, intersection point 2 lies within the fixture polygon and the angles between the vectors add up to 360 degrees. Intersection point 1 lies outside the fixture polygon and, therefore, the angle between the vectors add up to less than 360 degrees. A tolerance can be included in the intersection calculation to allow for a safety margin between the tool and fixtures.

The collision method can be expanded to check every part in the work area to detect collisions between the mill and other machines, such as robots. In this research, the collision routine has been limited to only the tool, tool collar, and mill fixtures to increase the computational efficiency of the simulator program.

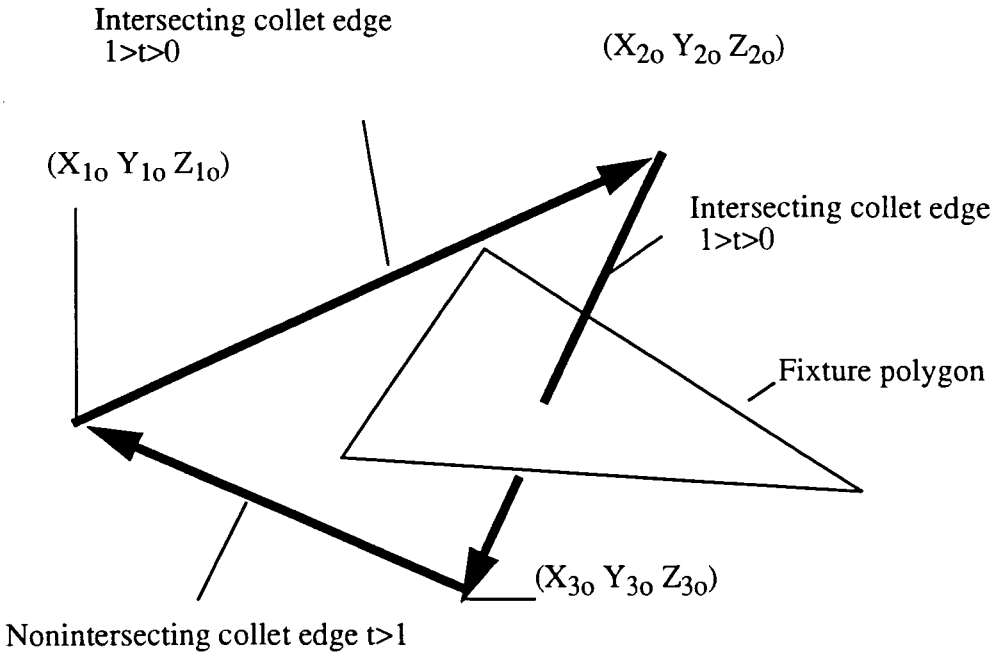


Figure 3.4 Intersection of collar edge with fixture polygon

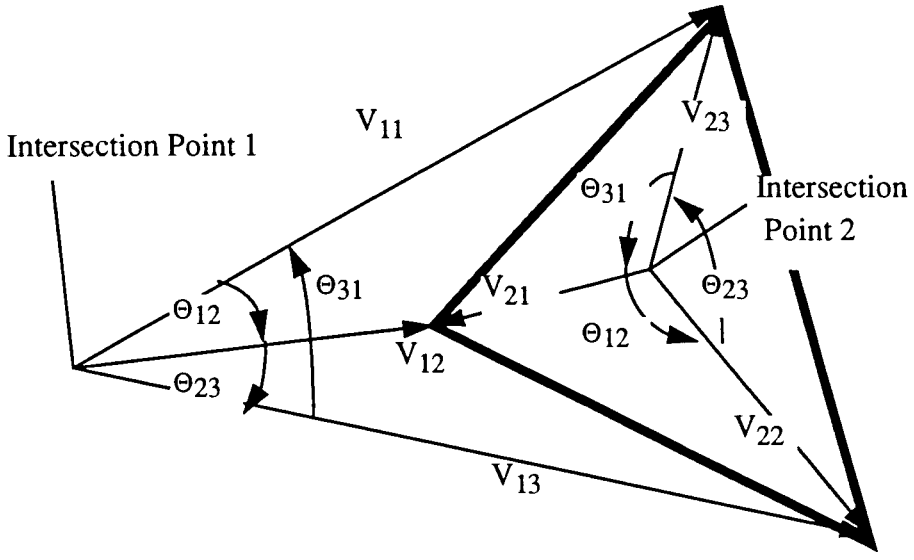


Figure 3.5 Vectors used to determine collision intersection

4 RESULTS

The simulation software was evaluated by simulating two common milling processes. The first process simulated was the milling of a part with several pockets incorporated in its design. The second process simulated involved a machined part with an unusual surface geometry. During both of these simulations, several evaluations were made on the accuracy of the simulation, the efficiency of the software, the quality of the display and the functionality of the interface.

4.1 Pocket Simulation

The first testing procedure involved milling the part shown in Figure 4.1. There are several pockets involved in the design of this part which can often result in fixture placement problems. In the past, the placement of fixtures was left to the experience of the machine operator. The part cannot be machined in a vice due to the channels running from the center to the outer edges. Therefore, two fixtures were positioned at the bottom left and upper right edges of the part. The hold down clamps positioned at opposite sides of the part were bolted down to the base plate of the machine. From an initial inspection it appears that the fixtures would not interfere with the machining process. However, during the machining process there was a collision between the tool and the lower left clamp (Figure 4.2). If the collision was not detected until the part was actually being milled on a machine then the collision could have resulted and damaged both the machine and the part. Another collision resulted between the tool collar and upper fixture during the drilling of the top right hole (Figure 4.3). The simulation software

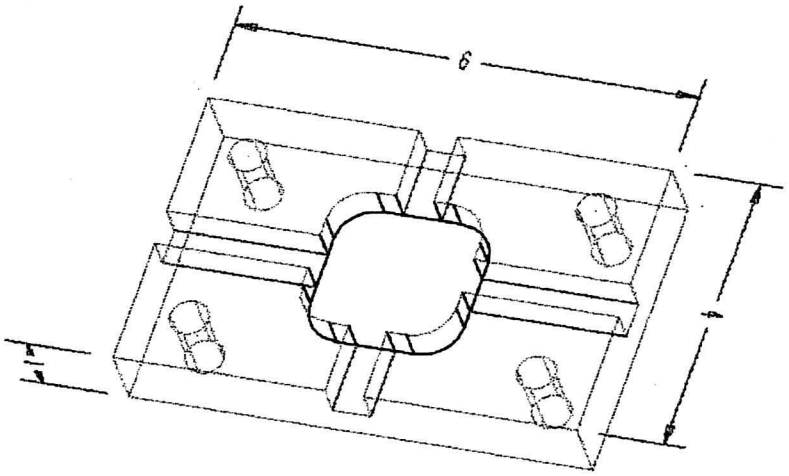


Figure 4.1 Part used in first testing procedure

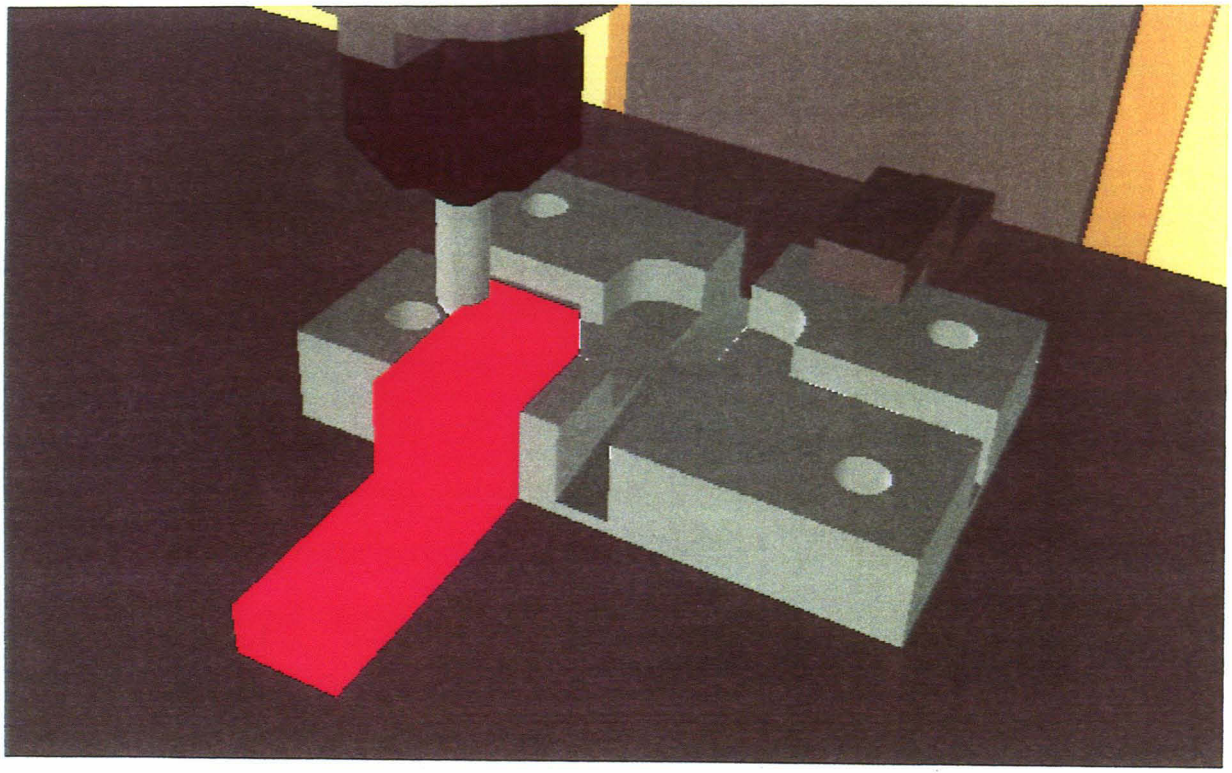


Figure 4.2 Collision between tool bit and fixture on test run #1

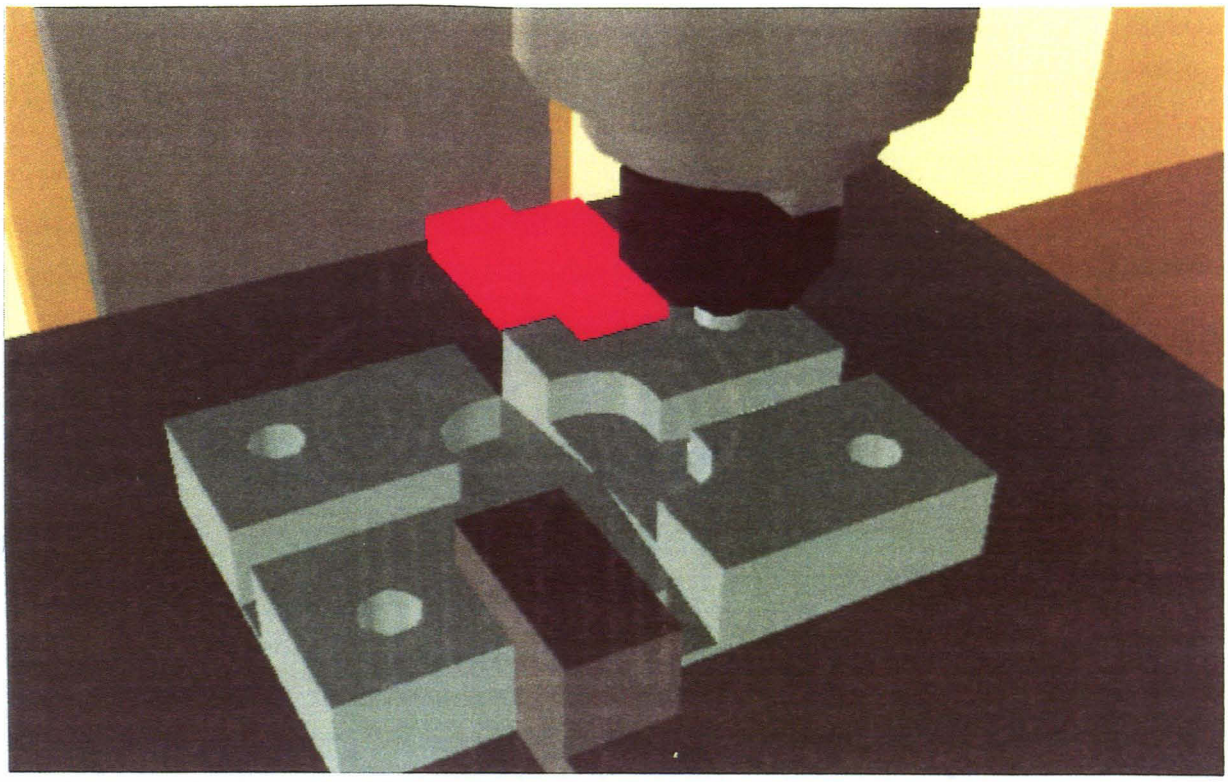


Figure 4.3 Collision between tool collar and fixture on test run #1

enabled the user to correct both problems before taking a milling machine out of production for a test run. There are several approaches that could have been taken to correct the problem. First, the existing fixtures could be repositioned by moving them farther out from the center of the part. A second approach that could be taken is to design a new fixturing assembly. A third approach is to clamp the part in a vice with plates of soft material between the part and the vice jaws. The soft material would act as spacers between the part and the fixture and could be machined along with the original part.The first option of changing the position of the existing fixtures was attempted because this required the least amount of additional time and money. After solving the fixturing problem, the NC code was down loaded to the milling machine and the milling process was successfully completed without any collisions.

4.2 Surface Geometry Simulation

The second testing procedure involved the part shown in Figure 4.4. This part required that it be fixtured in two different ways during the milling process. During the initial phase of milling, the three holes where drilled through the part material while the initial square stock was held down by a vice as shown in Figure 4.5. During the second phase of the operation the outside contour was machined while the part was mounted on a bottom plate by two fasteners as shown in Figure 4.6. The part was mounted on the plate to allow the tool bit to extend below the part while not damaging the mills' table plate. The original NC code used for the machining of this part intentionally contained mistakes to illustrate the capabilities of the software. One of the mistakes incorporated into the NC code occurred during the drilling of the upper right hole. The part was held in place by the vice without a base plate under the part. The tool bit extended below the part enough to collide with the fixturing vice. The second mistake in the NC code involved an excessive material removal rate while machining the lower right contour of the part. The material removal rate was too great because again the tool bit was extending too far below the part and was cutting too much material from the base plate under the part.

These problems are common during machining and were detected by the simulation software. To correct the first problem the operator had to either move the part so the interference did not occur or place a base plate under the part. The second problem was corrected by changing the CNC code so the tool bit does not traverse as much through the part.

4.3 Software Performance

During both of these test runs the software performance was evaluated. The features evaluated were: 1) the accuracy of the simulation, 2) the efficiency of the software, 3)the quality of the display, and 4) the functionality of the interface.

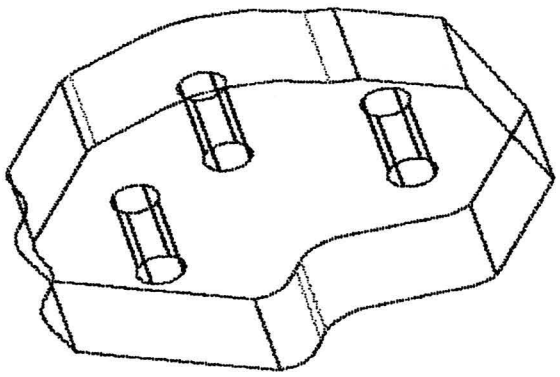


Figure 4.4 Machined part used for test run #2

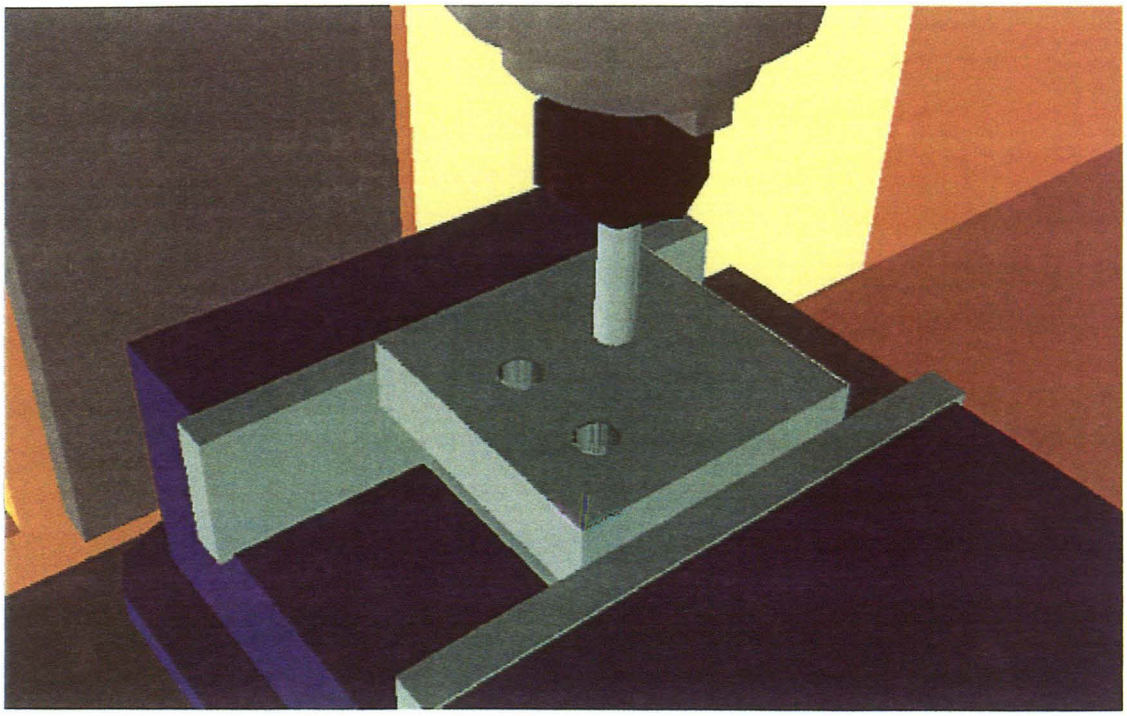


Figure 4.5 Test part #2 with vice fixture during intial phase of operation

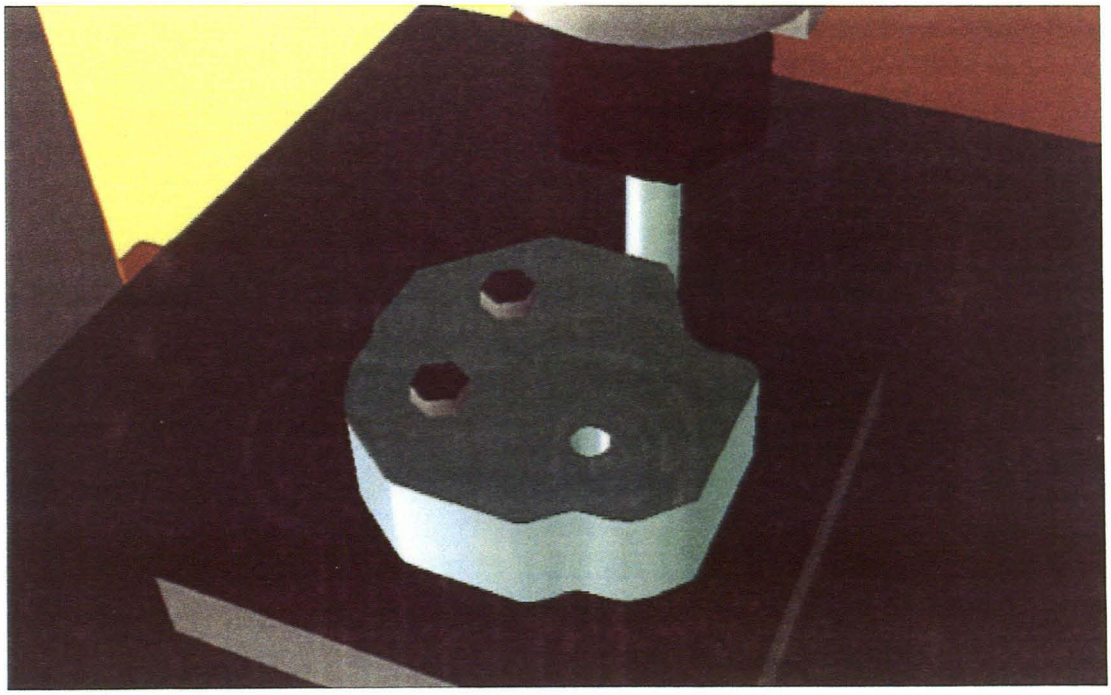


Figure 4.6 Test part #2 with bolt and base plate fixtures during second phase of operation

4.3.1 Dimensional Accuracy

The accuracy of the simulation software was evaluated during both testing procedures. During the first test run the software had errors resulting from an inability to precisely orient the fixtures with the local origin of the part. The software currently does not have the ability to display the exact position of each fixture with respect to the local origin. Instead, the user must guess the position of each fixture. This is currently being modified.

Another problem with the software is its inability to detect violations in the range of motion of the milling machine. Each milling machine has a specific range of motion in the X-Y-Z directions. Sometimes when creating NC code from a commercial CAM software package, the operator might call for a tool motion that exceeds the range of motion of the specific milling machine. Most of the CAM software available today is unable to detect boundary violations because they are required to communicate with several different machines. The ability

to set bounds on the range of motion for a simulation would greatly enhance the users capability to control the process and decrease the chance for mistakes in the NC code. This is also being implemented.

4.3.2 Efficiency

Efficiency is a major concern when dealing with interactive software. Since most of the motions during the simulation were accomplished through view transformations, the only computational load was from collision detection and material removal. The simulation software was efficient enough to allow the user to see the milling process faster than real time. A major source of trouble in this regard is the size and complexity of the fixtures being tested. When there are large fixtures, with a lot of polygons used in their representation, the simulation slows because every polygon is tested to detect collisions. This problem can be solved by localizing the test area on each fixture.

Material removal is another factor affecting the efficiency of the software, especially when a large part is tested or the number of nodes in the mesh is large. There is no way to correct this problem because the trade off is reduced accuracy in the simulation and the display. The user must be aware of this problem and adjust the simulation accordingly. For example, if the user has a verification software package available then the display of the part being machined is of no consequence since the user is only interested in the possibility of collisions. If the user does not have a verification routine than there may be interest in any gross errors in the part being machine and, therefore, must set the load factor to a high value.

4.3.3 Quality of Display

The quality of the display is closely related to the efficiency of the software. If the load factor is set to a low number than the quality of the display will not be as good as if the

load factor were set to a large number. Again, the user must be aware of this problem and adjust the simulation accordingly.

4.3.4 Functionality

The functionality of the user interface is of primary importance for interactive simulation software. By allowing the software to be used during the developmental stages of this research a general guideline of improvements was determined. Overall, the general feeling towards the simulation software was good and the need for such software was expressed by the users.

Some constructive comments expressed by people who have used this simulation software are listed below:

1) The menu layout is somewhat cluttered and difficult to understand.

2) A method of determining distances in the work space from objects on the screen is needed.

3) A status file should be developed to allow for easy and quick uploading and down loading of information.

4) A help menu should be incorporated into the software.

5) A method to load and analyze parts with unusual and complex geometries should be added.

6) The ability to edit the CNC code being tested from the simulation software is needed.

7) A fixture library should be included containing simple and most common type of fixtures

5 CONCLUSIONS AND RECOMMENDATIONS

This thesis presents the initial stages in the development of a graphical simulation program for the DynaMyte 4400 milling machine. Algorithms were developed for the material removal and the detection of collisions during the milling operation. Also, a user interface was developed to allow the user to control certain process parameters and allow them to experiment with different fixturing configurations. Testing of the simulation program was conducted to verify the programs functionality and to suggest possible improvements in the software. This software when used in conjunction with a verification software package could greatly reduce the time required for testing and implementation of new operations on the milling machine.

With the requirement of increased efficiency, the need for simulation and verification is obvious. Computer visualization allows for greater understanding of solutions to many different problems by showing the solution in an easily understood format.

BIBLIOGRAPHY

[1] Van Hook, T., "Real-Time Shaded NC Milling Display," *Computer Graphics, (Proc. SIGGRAPH '86)*, Volume 20, Number 4, pp 15-20, August 1986.

[2] Oliver, J.H. and Goodman, E.D., "Graphical Verification on NC Milling Programs for Sculptured Surface Parts," *10th Annual Automotive Computer Graphics Conference and Exposition*, Engineering Society of Detroit, December 1985.

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

[4] Jerard, R.B., Drysdale, R.L., "Geometric Simulation of Numerical Control Machining," *Proc. ASME Int. Computers in Engineering Conf*, San Francisco 2, pp 129-136, 1988.

[5] Wang, W.P., Wang, K.K., "Real-Time Verification of Multiaxis NC Programs With Raster Graphics," *IEEE Proc. of Int. Conf. on Robotics and Automation*, San Francisco, pp 166-171, April 1986.

[6] Kawashima, Y., Itoh, K., Ishida, T., Nonaka, S., Kazuhiko, E., "A Flexible Quantitative Method for NC Machining Verification Using Space-Division Based Solid Model," *The Visual Computer*, pp 149-157, 1991.

[7] Saito, T. and Takahashi, T., "NC Machining with G-buffer Method," *Computer Graphics, (Proc. SIGGRAPH '91)*, Volume 25, Number 4, pp 207-216, 1991.

[8] Chappel, I.T., "the Use of Vectors to Simulate Material Removed by Numerically Controlled Milling," *Computer Aided Design*, Volume 15, Number 3, pp 156-158, 1983.

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

[10] Christiansen, H.N., *Movie.BYU Training Text*, Community Press, Provo, UT, 1986.

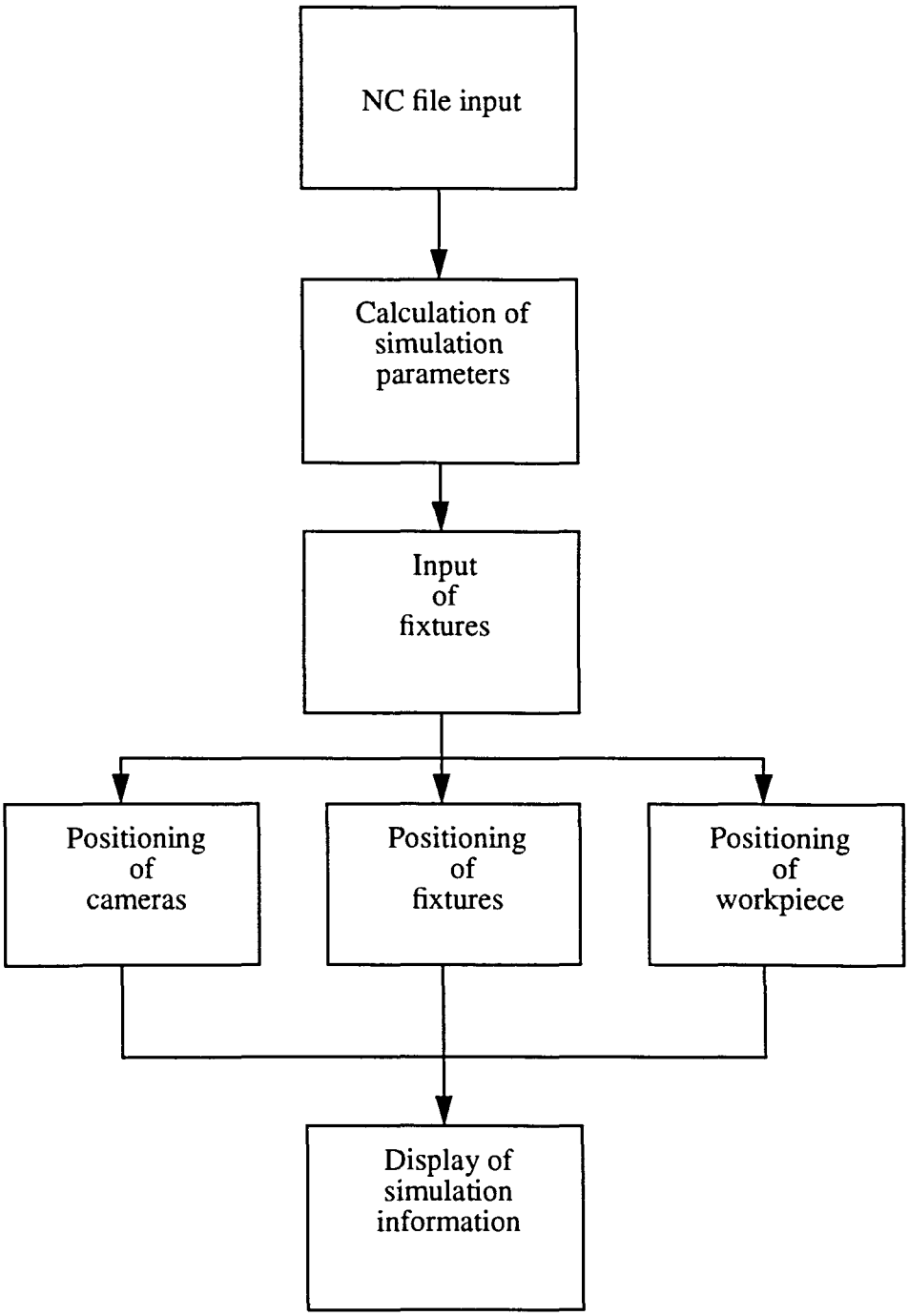
[11] Foley, J.D., VanDam, A., Feiner, S. and Hughes, J.D., *Computer Graphics Principles and Practice*, 2nd ed. Addison Wesley, Menlo Park, CA, 1990.

[12] Silicon Graphics Inc., *Graphics Library Programming Guide*, Mountain View, CA, 1991.

APPENDIX

- Simulation Software Flow Chart
- Simulation User's Manual

Simulation Software Flow Chart



Simulation Users Manual

This manual is intended to give the first time user the basic information necessary to use the mill simulator to create a milling simulation.

Starting The Mill Simulator

The mill simulator is started by typing *mill_gl*. Once the simulator has started up, notice that the display is divided into four main areas as shown in figure A1. The majority of the screen is taken up by the viewing window which displays the workcell of the milling machine and the simulation. The NC controls in the upper right of the screen control the milling parameters such as which type of NC file being tested as well as the dimensions of the workpiece and the cutting tool. The simulation controls in the lower right of the screen control which fixtures are used in the simulation and from which camera the simulation is being viewed from. The positioning controls in the lower left of the screen contain a bank of sliders that positions the fixtures, the cameras, and the workpiece.

Positioning Cameras

The cameras can be positioned by first selecting the *camera mode* button in the simulation control area of the screen. Next, select the camera that is to be changed from the camera buttons in the simulation control area of the screen. The position of the camera can now be changed by the current set of sliders in the positioning area of the screen.

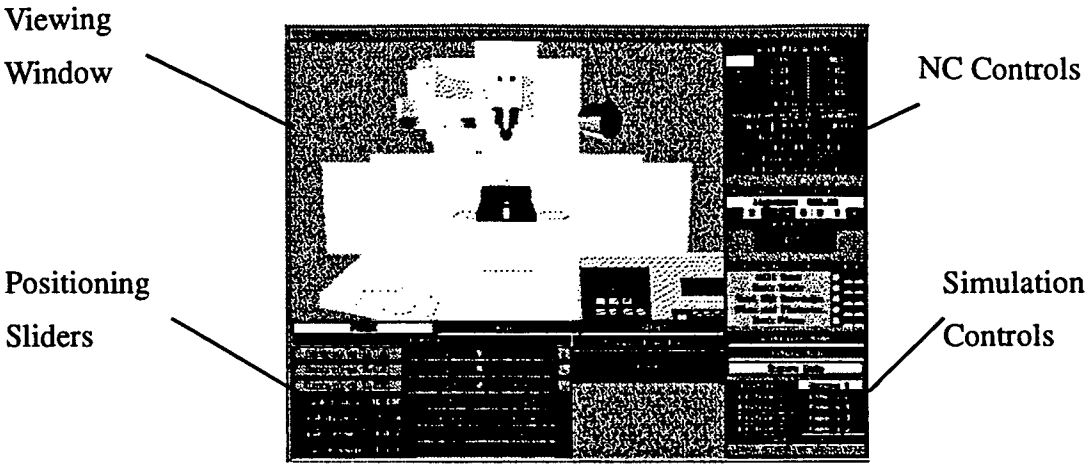


Figure A1 Mill simulator screen layout.

Loading and Positioning Fixtures

Fixtures can be loaded by using the *load fixture* button in the simulation control area of the screen. The fixtures must be in BYU format and the polygons triangularized when a fixture is loaded in. The fixtures can then be positioned by first selecting the *fixture mode* button. Then the fixture number must be given at the top of the set of sliders in the area marked *fixture number*. The fixtures are then positioned by using the active sliders. Activating which fixtures are to be used in the simulation is done by selecting the fixture number from the bank of buttons in the simulation control area of the screen.

Positioning the Workpiece

The workpiece is positioned in the same way as the cameras. First select the *workpiece mode* button in the simulation control area of the screen then use the active sliders in the lower left of the screen.

Milling Parameters

The milling parameters are controlled in the upper right of the screen. The initial stock dimensions are given by using the *stock dimensions* button and specifying the width, height, and depth of the object. The cutting tool parameters are controlled by using the *tool type*, *tool diameter*, and *tool height* buttons. Using the *NC filename* button allows the user to load in different NC control files.

Miscellaneous Parameters

This section describes miscellaneous buttons not described previously. The *recompute* button resets all the simulation parameters to their intial values so the simulation can be viewed from the beginning. The *reset alarm* button reset the collision alarm associated with each fixture to allow the simulation to proceed from the point where there was a collision. The set of rendering style buttons allows the user to set how each object in the viewing window is displayed. The three choices are solid, wireframe and blank. The *pause*, *play*, and *2xplay* buttons start and pause the simulation.