# COMPUTER GRAPHIC REPRESENTATION OF REMOTE ENVIRONMENTS USING POSITION TACTILE SENSORS

by

DONALD CHARLES FYLER

B S M E , University of Massachusetts, Amherst
( 1978 )

SUBMITTED TO THE DEPARTMENT OF
MECHANICAL ENGINEERING IN PARTIAL
FULFILLMENT OF THE
REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1981

Signature of Author [illegible]
Department of Mechanical Engineering
August 4,1981

Certified by [illegible]
Thomas B Sheridan
Thesis Supervisor

Accepted by
W M Rohsenow
Chairman, Department Committee

# COMPUTER GRAPHIC REPRESENTATION OF REMOTE ENVIRONMENTS USING POSITION TACTILE SENSORS

by

DONALD CHARLES FYLER

Submitted to the Department of Mechanical Engineering on August 4, 1981 in partial fulfillment of the requirements for the Degree of Masters of Science in Mechanical Engineering

## ABSTRACT

The usefulness of remotely controlled manipulators is increasing as the need grows to accomplish complex tasks in hazerdous environments such as the deep ocean

The best sensory input currently availiable to the operator of a remote supervisory controlled manipulator is a television picture of the manipulator and its surroundings Very often, though, optical opacity due to suspended particles in the water can make television impractical or impossible to use This report investigates the use of touch sensors to construct a picture of the manipulator surroundings One method studied was to find 3-dimensional surface points and show them on a computer graphic display An extension of this was to reconstruct the surface of these points with the aid of a computer

It was found to be possible to quickly construct a reasonable picture with a position touch sensor by showing 3-D surface points on the graphic display and then having them rotate about an arbitrary center A better picture could be made by reconstructing the actual surface, but this took more computer time

An informal evaluation by observers suggests that this method offers practical advantages for "seeing" objects in environments where vision is impossible

Thesis Supervisor Thomas B Sheridan
Title Professor of Mechanical Engineering

## ACKNOWLEDGEMENTS

I would like to thank Professor Thomas Sheridan for his support and for the opportunity to work in the Man-Machine Systems Laboratory Special thanks to Calvin Winey for advice and help in maintaining lab equipment His research in the field of manipulator simulation provided a basis for much of this report I would like to thank Ahmet Buharali and Dana Yoeger for support of the computer system and for the help in understanding how it worked I would also like to thank Ryang Lee for his help proof-reading and organizing this report Finally I wish to thank all the Man-Machine Systems Lab members for making it an enjoyable place to work

This research was supported in part by Contract N00014-77-C-256 with the Office of Naval Research Arlington VA 22217 G Malecki was contract monitor

TABLE OF CONTENT

ABSTRACT 2

ACKNOWLEDGMENTS 3

LIST OF FIGURES 6

CHAPTER 1 INTRODUCTION 8

CHAPTER 2 PROPOSED SOLUTION 11
2 1 Construction of a Picture with Touch Sensors 11
2 2 Construction of a Surface from Points 14

CHAPTER 3 EQUIPMENT 16
3 1 Manipulator 16
3 2 Computer 16
3 3 Vector Graphics System 16
3 4 Analog to Digital Converter 18
3 5 Trackball 18
3 6 Raster Graphics System 18

CHAPTER 4 TOUCH SENSORS 19
4 1 Best Configuration 19
4 1 1 To Sense Touch Direction or Surface Direction 19
4 1 2 Rigid or Flexible Base 21
4 1 3 Where to Mount the Sensor 23
4 2 Touch Sensors Used in Experiments 26

CHAPTER 5 CALCULATION OF POINT COORDINATES 30
5 1 Description 30
5 2 Problems with the Manipulator 30
5 2 1 Cables and Gears 32
5 2 2 Pushrod 33

CHAPTER 6 POLYHEDRON CONSTRUCTION 38
6 1 Introduction 38
6 2 2-Dimensional Solution 39
6 3 3-Dimensional Solution 45
6 3 1 Polyhedron Description 45
6 3 2 Initializing the Polyhedron 53
6 3 3 Checking Facet Pairs 56
6 3 4 Quality of Polyhedron Shapes 62

CHAPTER 7 METHODS OF DISPLAY 63
7 1 Introduction 63
7 2 Problems with Polyhedron Displays 63
7 3 1 Rotating the Picture 65
7 3 2 Types of Rotation 70
7 3 3 Oscillating the Picture 70

7 3 4 Rotation with a Joystick or Trackball 71
7 3 5 Position Control 71
7 3 6 Velocity Control 72
7 4 Improving the Display for Polyhedra 74
7 4 1 Showing All Edges 74
7 4 2 Drawing Contour Lines 75
7 4 3 Raster Graphic Display 76

CHAPTER 8 EVALUATION 81
8 1 Number of Points to Make a Picture 81
8 2 Speed of Picture Construction 88
8 2 1 Construction Time for Points 88
8 2 2 Construction Time for Polyhedra 90
8 3 Raster Display 92

CHAPTER 9 CONCLUSIONS AND RECOMMENDATIONS 93
9 1 Conclusions 93
9 2 Recommendations 93

REFERENCES 95

APPENDIX A Computer Program Discription 96
APPENDIX B Computer Program for Display of Manipulator and Surface Points 101
APPENDIX C Subroutines to Construct Polyhedron from Surface Point Data 107
APPENDIX D Subroutines for Display Management 118
APPENDIX E Program for Raster Display of Polyhedron 127

## LIST OF FIGURES

2 1 Position Touch Sensors Used for Graphic Display 12

2 2 Vector Graphic Display of Manipulator 15

3 1 E-2 Manipulator 17

4 1 Touch Sensor with Surface Normal Touch Sensing Capability 22

4 2 Flexible Base Touch Sensor 22

4 3 Touch Sensor with Extra Two Degrees of Freedom 24

4 4 3 Degree of Freedom Manipulator with Interference Problem 25

4 5 Possible Solution to Interferance Problem 25

4 6 Brush Touch Sensor 28

4 7 Single Switch Touch Sensor 28

4 8 Switch Mechanism that is Sensitive to Touch from all Angles 29

5 1 Manipulator Coordinate System 31

5 2 Nonlinear Pushrod 34

5 3a Grid Errors Due to Pushrod Nonlinearity 37

5 3b Grid Errors Reduced with Compensation 37

6 1 Connecting 2-D Surface Points Into a Polygon 41

6 2 Definition of Lines and Polygons 41

6 3 Concave Polygons 43

6 4 Definition of Touch Vectors 43

6 5 Constructing Concave Polygons with Touch Vectors 44

6 6 Examples of Points That Cannot be Attached 46

6 7 Example of an Erroneously Attached Point 47

6 8 3-D Definition of Facets 49

| | | |
|---|---|---|
| 6 9 | Example of Complete 3-D Polyhedron | 49 |
| 6 10 | Addition of New Points to the Polyhedron | 52 |
| 6 11 | Possible Errors from Attaching Distant Points | 54 |
| 6 12 | Attaching Interior Points | 55 |
| 6 13 | Initialization of Polyhedron | 55 |
| 6 14 | Need for Smoothing of Polyhedron | 57 |
| 6 15 | Example of a Facet Pair and Its Compliment | 59 |
| 7 1 | Different Displays for One Set of Points | 64 |
| 7 2 | Screen and Object Coordinates | 73 |
| 7 3 | Example of Polyhedron Snown on Raster Display | 80 |
| 8 1 | Cubes Described by Randomly Distributed Points | 83 |
| 8 2 | Random Cube Points Made Into Polynedron | 84 |
| 8 3 | Spheres Described by Randomly Distributed Points | 85 |
| 8 4 | Random Sphere Points Made Into Polyhedron | 86 |
| 8 5 | Graph of Computation Time Versus Polyhedron Size | 91 |

# CHAPTER 1 INTRODUCTION

Remotely controlled manipulators make it possible to perform tasks in hostile environments that would be impossible or very dangerous for humans to perform It is very difficult and expensive to send a man down into the deep ocean to do a task But tasks such as exploration, salvage, and maintenance of oil rigs must be done Because the technology is not yet available to make a completely autonomous robot, some compromises must be made A robot can be made as self sufficient as the technology allows and the higher order thinking can be left to a human controller This robot-human system is called Supervisory Control and is meant to relieve the human of as much direct control as possible to minimize the amount of required transmitted data and perhaps even allow the robot to continue working during breaks in transmission

In human-manipulator control systems, it is very important that the human have as much feedback as possible about what is happening at the manipulator Sight is considered to be the most important source of feedback because it can be readily understood by the operator If the operator cannot directly see the manipulator and manipulated object, (which is often the case), some sort of artificial vision must be provided This is most often a television picture of the manipulator work area Television provides the best picture available but there are some problems that can make television hard to work with Some of these problems are

1) Television cannot give a reliable sense of depth because it is only displayed on a 2-dimensional screen This can slow the operator's reaction time because he can never be sure if the manipulator arm or its surroundings are really in the place he thinks they are It is possible to use two cameras to get a stereo picture but this kind of display requires undivided attention and the operator can become fatigued very quickly

2) The raster picture on the television screen requires a massive data flow rate to refresh the screen in a reasonable amount of time If the operator is trying to control a manipulator working on the bottom of the ocean or in deep space, the data flow rate can be very restricted by transmission problems This means the operator will have to live with a fuzzy picture or a slow frame rate or both

3) A television camera must have a clear view of the manipulator It cannot see anything in turbid water and the television must always be located so obstructions do not block the view

4) In modern types of supervisory control systems, a computer works intimately with the operator to control the manipulator The computer should have as much feedback as possible made available to it While a television picture is easily understood by a human, it is meaningless to a computer unless it has extensive, time-consuming processing A computer of any control system is essentially blind to a television picture

These problems show the need for investigating new, types of viewing systems for use in supervisory control A system using touch sensors to construct a simulation of the surroundings of a manipulator is investigated in this report This kind of simulation can be used to draw a picture to be viewed by a human or can be used to provide 3-dimensional information to a computer about the surroundings of the manipulator

# CHAPTER 2 PROPOSED SOLUTION

## 2 1 Constuction of a Picture with Touch Sensors

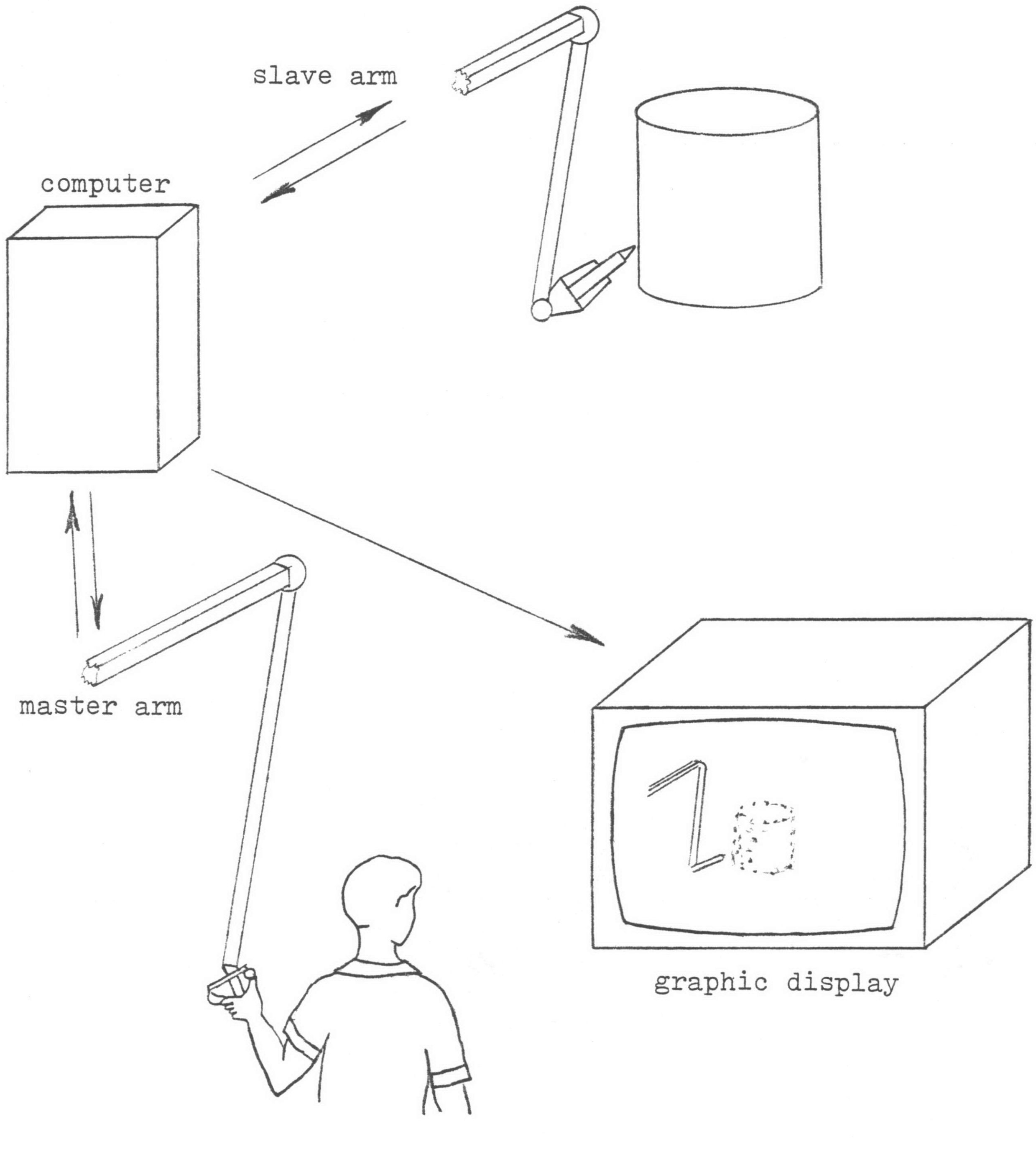
A method is needed to improve visual feedback using touch sensors for a human operating a remote supervisory controlled manipulator One way is to find the coordinates of a large number of points on all the solid surfaces within reach of the manipulator A picture of the manipulator surroundings can then be constructed with computer graphics by drawing a dot at each location where a solid surface is hit

Points and their coordinates can be found by using touch sensors mounted on the manipulator Whenever a sensor comes in contact with a surface it could send a signal to the computer to record the coordinates of the point touched The computer can accurately calculate point coordinates if it is given the exact angles of the manipulator joints the instant the sensor is tripped, see Fig 2 1

A dynamic simulation of the manipulator itself can also be added to the display as a reference if these angles are known, [1] This means an entire picture of the of the manipulator surroundings plus a moving picture of the manipulator can be made with just information on the values of the joint angles and indications of when sensors are tripped

Very little transmitted data is required to describe

Fig. 2.1 Position Touch Sensors Used for Graphic Display



3-dimensional points as opposed to a television picture Assuming the joint angles are to be transmitted anyway, all that is needed to describe a dot is an indication of which touch sensor had just been triggered Its coordinates can then be calculated from the joint angles given at that instant

A picture on a 3-dimensional graphic display has the same disadvantage as a television picture in that it can only be shown on a 2-dimensional screen But a graphic display picture can be viewed from any angle, something a television cannot do without having the camera moved An obstacle blocking a clear view of the manipulator on the display could be ignored by simply looking around it

Also, because the data of the graphic display picture is stored in three dimensions, the picture can be modified to bring out it's depth of field Showing shadows, orthographic views, and perspective will bring out three dimensionality, [1] Dynamic pictures also bring out depth The three dimensions of the picture become very apparent when it is slowly rotating on the screen

An advantage of having the surroundings of the manipulator mapped out as discrete points is that it can be quickly interpreted by a computer Say a task given to a computer is to move a manipulator arm from one spot to another without hitting any obstacles If the computer is given enough information about 3-D point locations on the obstacles, then it could be programmed to keep the the manipulator away from the surface points This would be easier for the computer to solve than trying to interpret a flat television picture

## 2 2 Construction of a Surface From Points

A problem with surface points shown on a graphic display is that they give a somewhat ambiguous indication as to what the surface is like between them Without the surface, there is no way to calculate volume, surface area, or decide when something should be hidden from view

A method was found to reconstruct the surface described by a given set of points with the aid of a computer This method will be covered in some detail, as it provides a solution to the above problems and also can significantly improve the quality of the graphic display used in supervisory control

Computer graphics can never replace television as a sense of sight in supervisory control but it could be a very useful aid to television or even an alternative in situations where television is impossible to use

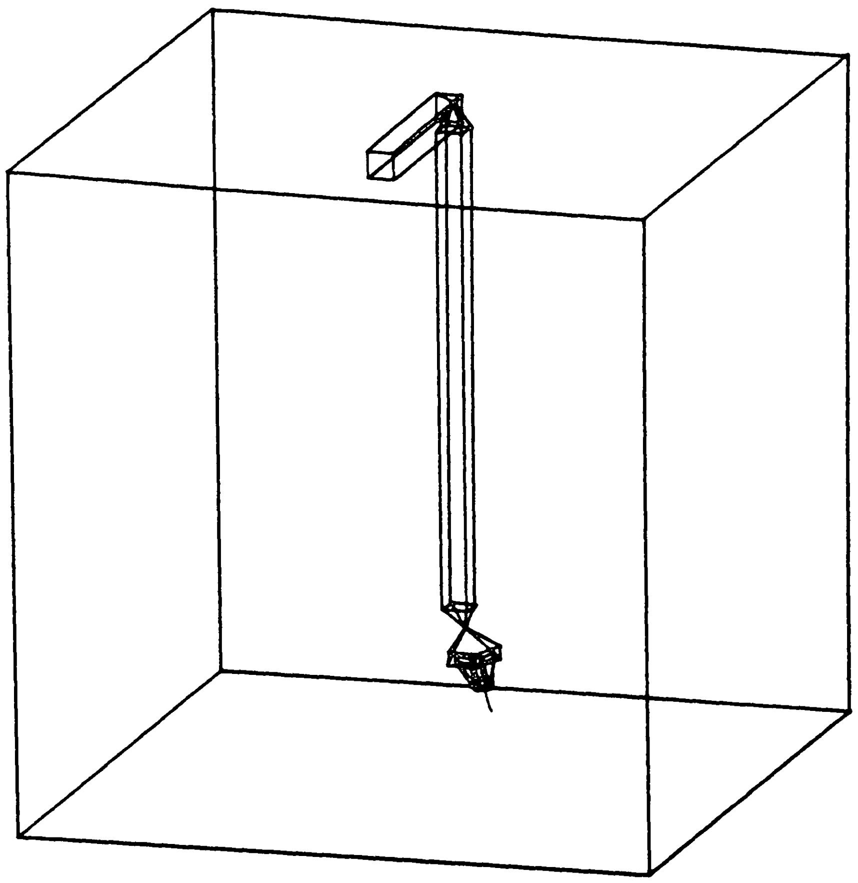


Fig 2 2 Vector Graphic Display of Manipulator

# CHAPTER 3 EQUIPMENT

## 3 1 Manipulator

The manipulator used in this project was a master-slave E-2 built by the Argonne National Laboratories for use in radioactive environments, see Fig 3 1 The control system used in experiments was analog with full force feedback Control potentiometers installed at the servos provided a signal for determining manipulator joint angles Interfaces between the manipulator and the A/D converter were installed by K Tani [2]

## 3 2 Computer

A PDP 11/34 with a RSX-11M timesharing operating system was used for all computation There was a FP11-A floating point processor installed to speed the fractional multiplication and division required for real time graphic transformations and simulation

## 3 3 Vector Graphic System

All vector graphics were done on a Megatek 7000 System It had a resolution of 4096 x 4096 on the approximatly 12 x 12 screen There was room in the display list for 8000 3-dimensional points or lines This system was capable of hardware rotations to speed the cycle time for dynamic display

Interface between the Megatek and computer was done through a user common where all display information could be

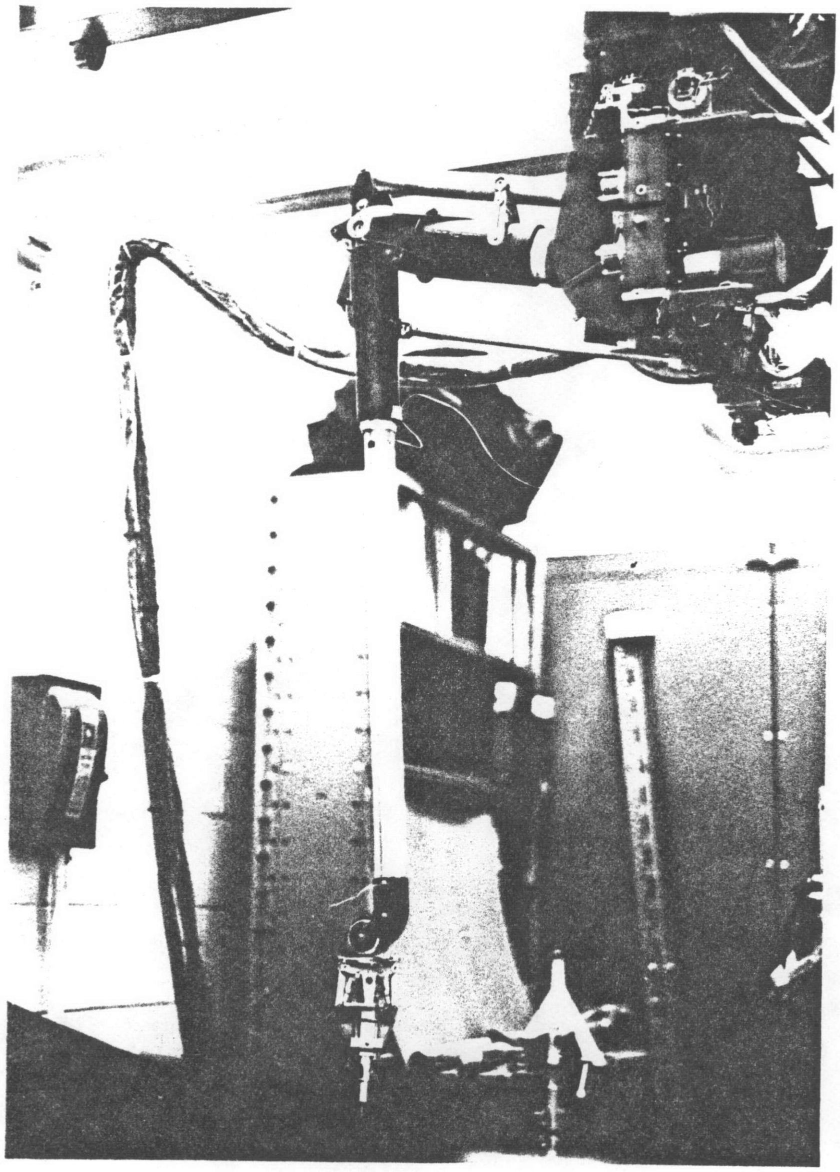


Fig. 3.1: E-2 Manipulator

stored until a command was called to send the information to the Megatek all at once

## 3 4 Analog to Digital Converter

An Analogic 5400 Series was used to convert analog signals to digital for computer input It also had inputs that could convert simple on-off signals to digital numbers The six analog channels giving the joint angles of the manipulator could be read in about 300 microseconds on the parallel interface

## 4 5 Trackball

The Measurement Systems Inc Trackball was connected to the computer through a serial interface It had a resolution of 512 for 360 degrees of ball travel and would output the number of units travelled between each send to the computer The send rate was set by a baud rate of 9600 The Trackball was sensitive to motion around both x and y axes but not around the z axis

## 4 7 Raster Display

A Lexidata 3400 Vidio Processor was used for raster display It had a resolution of 640 x 512 pixels with each pixel having 256 possible shades The shades were stored in a lookup table where they could rapidly be changed

# CHAPTER 4 TOUCH SENSORS

A touch sensor is required that will respond when it comes in contact with a solid surface and has to be configured in such a way that the exact location of the contact point can be determined Many different types of sensor switching devices can be imagined Switches based on pneumatics, stess, strain, or electical inductance might have good applications in different environments but for experimental purposes simple electical switches were used Whatever the sensing device, it must be converted into an electical signal for the computer The configuration of the touch sensor was found to be much more important than the actual sensing mechanism

## 4 1 Best Configuration

### 4 1 1 To Sense Touch Direction or Surface Direction

When reading the three dimensional coordinates of a point on a surface it is also useful to find a vector pointing the direction of the surface normal at that point This would give valuable information about how the surface is structured The problem is that two degrees of freedom will have to be added to the touch sensor to enable it to read a surface normal, see Fig 4 1 Adding more degrees of freedom significantly increases mechanical complexity, the amount of data that must be transmitted to the computer, and computation time There is another problem in that the surface normal would be be found for only one small spot The surface normal in the immediate neighborhood of the point would only be implied The average surface normal over a larger area could be found but would be at the expense of resolution of the point location The surface normal could be found more accuratly if the points touched were densely packed, but then the points themselves describe the surface normal

Although the ability to sense the direction of the surface normal would increase the surface description capabilities of a touch sensor, it was decided that it was not worth adding two more degrees of freedom Since three adjacent surface points describe an average surface normal, it was felt there was no need to find it for every single point

It was found to be useful, though, to record the touch sensor direction for each point This was actually the center line of the touch sensor at the instant a point was touched The touch direction was easily found because it had to be known to calculate the coordinates of the point anyway The touch direction was useful because it defined a line that could not pass through the surface This helped to define inside from outside A series of points on a plane can describe a surface normal but cannot, by themselves, describe which side of the plane is the outer side

### 4 1 2 Rigid or Flexible Base

A touch sensor mounted rigidly to the manipulator would be more reliable and accurate than one mounted on a flexible base The mathematics required to find its coordinates would be simpler and so would its mechanical complexity It might seem that a rigid mounted sensor would be the best But there are some advantages to a flexibly mounted sensor that may outweigh its disadvantages One advantage of a flexibly mounted sensor is that the manipulator would not have to come to a complete stop when a point was touched The sensor could just bend out of the way and not impede the continuous motion of the manipulator This would allow faster motion of the manipulator and would reduce the risk of damage to the manipulator, sensor, or the object to be touched Another advantage would be that many sensors could be used at once if they were all on flexible mounts When one sensor hit a surface, it could respond and then bend out of the way to let the next sensor touch, see Fig 4 2

Flexible-base touch sensors could be constructed with or without degrees of freedom The type without degrees of freedom would only work when straight, then simply shut off when bent over so as not to register any erroneous points If the sensor had one or two degrees of freedom, it could

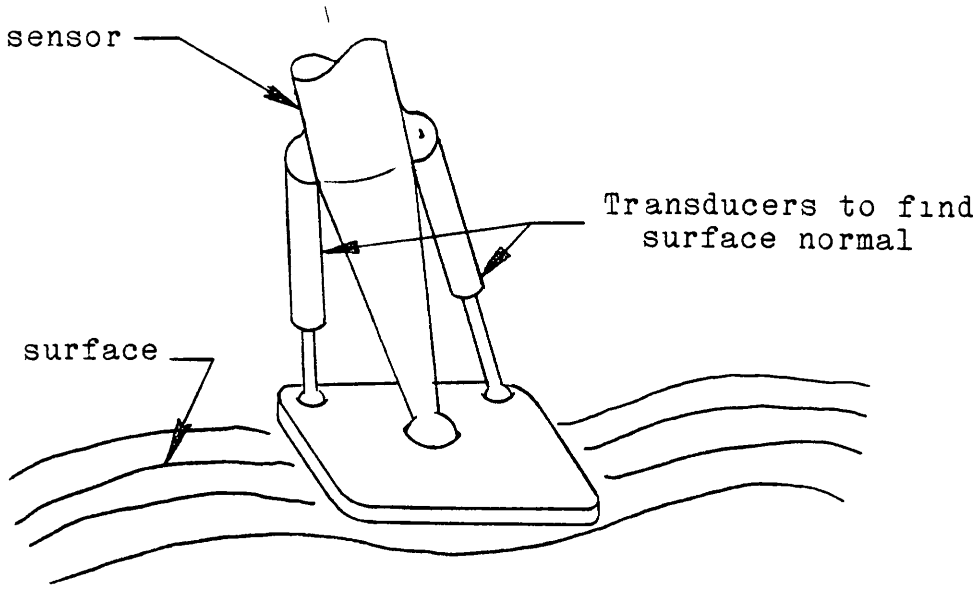


Fig 4 1 Touch Sensor with Surface Normal Touch Sensing Capability

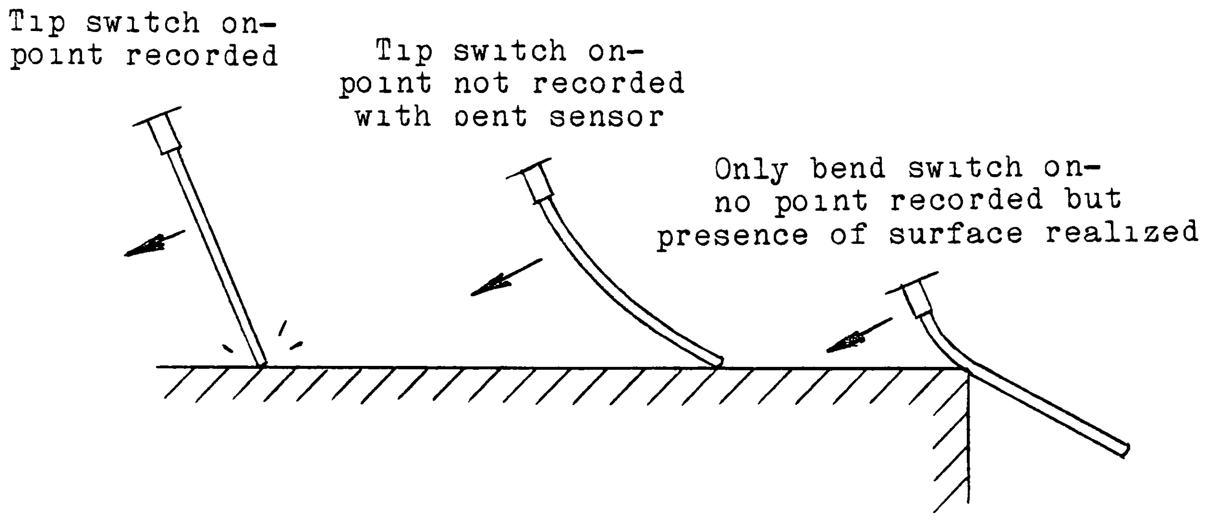


Fig 4 2 Flexible Base Touch Sensor

still register points even when bent over, see Fig 4 3 This way, a continuous stream of points could be read in one motion The trade off would come when deciding whether it is more important to have fewer degrees of freedom or the capability to read many points with one sensor

## 4 1 3 <u>Where to Mount the Sensor</u>

Since the sensor is to work with a manipulator, the most likely place to mount the sensor would be on the manipulator itself If the sensor were mounted at the wrist of the manipulator, the sensor would have six degrees of freedom and be most maneuverable If it were too awkward to use the wrist, the next best mount would be the forearm of the manipulator This would reduce the number of calculations required to locate the sensor in space but would still leave three degrees of freedom

The sensor could theoreticly reach any point in front of the manipulator but the sensor would only be able to approach any one point from one direction The sensor would not be able to reach around an object in the way, see Fig 4 4 A solution might be to install many sensors on the arm protruding in all different directions so as to be able to reach all points with at least one sensor, see Fig 4 5

The very best mounting location would be to have the sensor mounted on its own arm This could run completely independent of the manipulator and be controlled by a

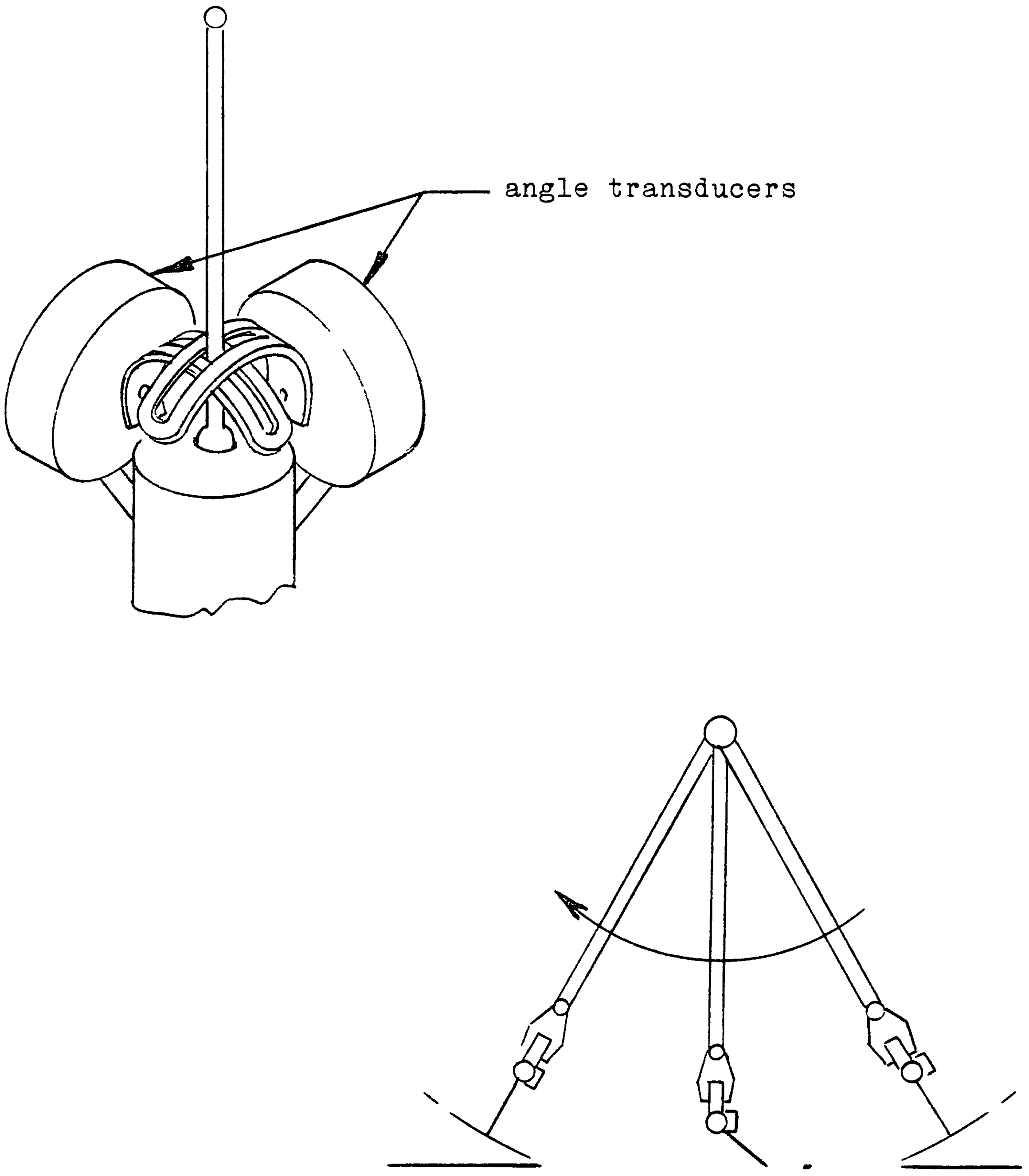


Fig 4 3 Touch Sensor with Extra Two Degrees of Freedom

A long string of points could be recorded with one sweep of the manipulator

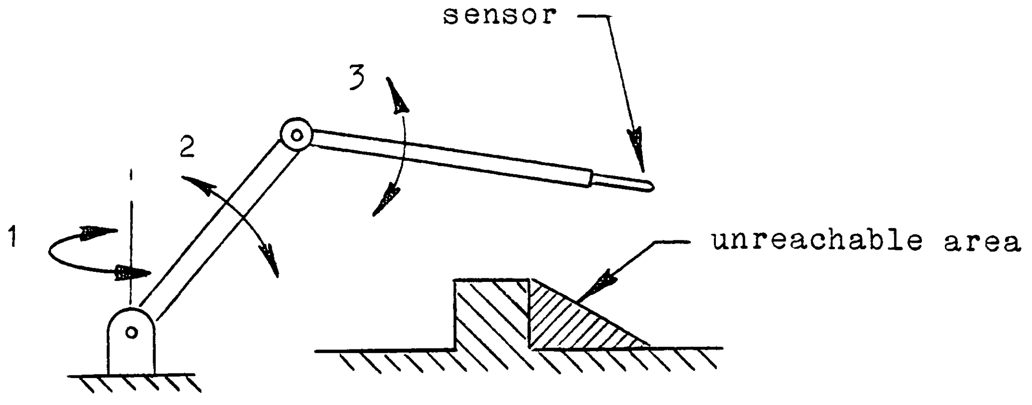


Fig 4 4 3 Degree of Freedom Manipulator with Interference Problem

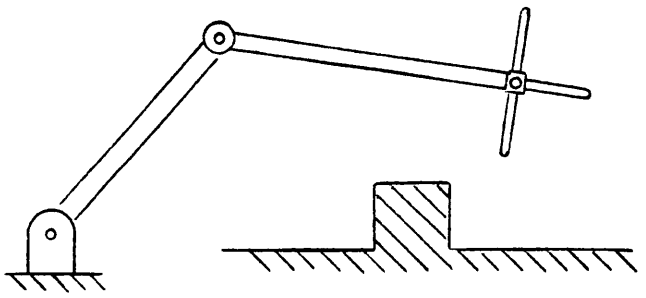


Fig 4 5 Solution to Interference Problem

Several mounted touch sensors could reach more areas and would not increase the degrees of freedom of the manipulator different operator or perhaps be completely controlled by computer A computer could be programmed to randomly sweep the sensor around and to concentrate on relatively untouched areas

## 4 2 Touch Sensors Used in Experiments

The touch sensors that were built for experiments were designed solely to get surface points into the computer as efficiently as possible The touch sensors were always mounted firmly in the jaws of the manipulator and only on-off electrical switches were used to send signals to the computer

The first sensor built had 10 switches on it and each was connected seperately to digital inputs on the analog to digital converter, see Fig 4 6 The switches were mounted on somewhat flexible stems and were arranged like a brush It was found that a shorter stem provided the most accurate point coordinates and a slight convex curve to the profile of the endpoints of the stems allowed the sensor to be rocked across a surface to collect a maximum amount of points

This brush sensor had some problems that made it difficult to use The biggest problem was that the switches worked only when pressed from one direction When a switch was hit from the side nothing would happen This meant the sensors always had to be pointed in the direction the manipulator was being moved to make sure the switches would be hit straight-on Another problem was the sensors were too far from the base of the manipulator wrist It turned out that the joint angles of the wrist could not be calculated accuratly and errors multiplied the farther the sensors were from the base of the wrist

The second sensor built had only one switch on it, see Fig 4 7 This was because in later experiments it was desirable to be able to select individual points on a surface Also, the second touch sensor was located such that one degree of freedom of the wrist was not needed to calculate the sensor's coordinates

Although the brush sensor had many more switches on it, the second sensor could collect points just about as fast This was because the second sensor was made to be sensitive when approaching a surface from any direction, see Fig 4 8 Besides being easier to maneuver than the brush sensor it could also be moved faster because the manipulator only had to move at the wrist to trigger the switch The brush sensor required that the entire manipulator be moved to get the switches to approach the surface from the correct direction

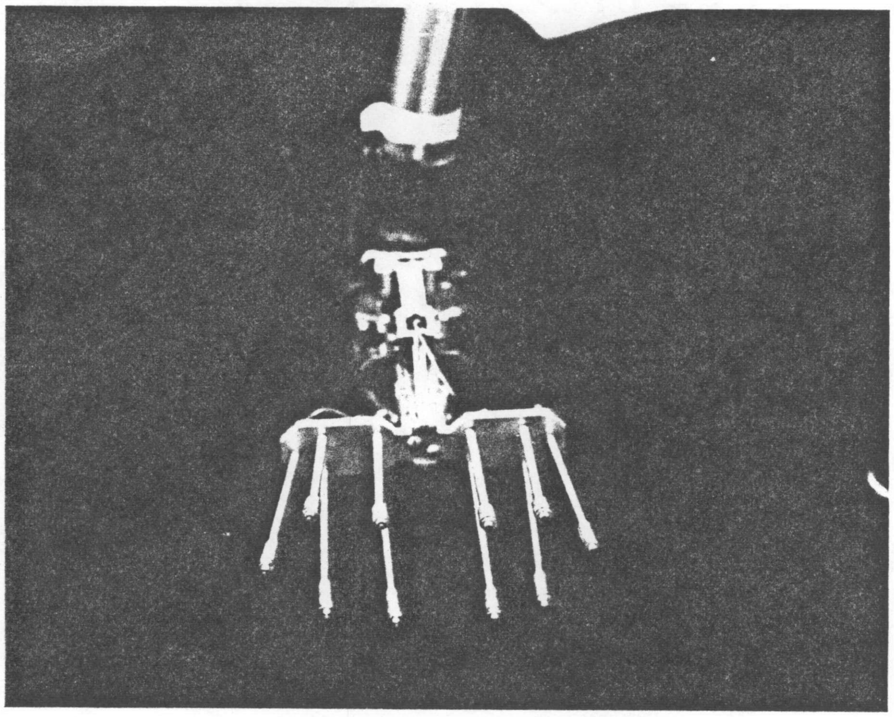


Fig. 4.6: Brush Touch Sensor

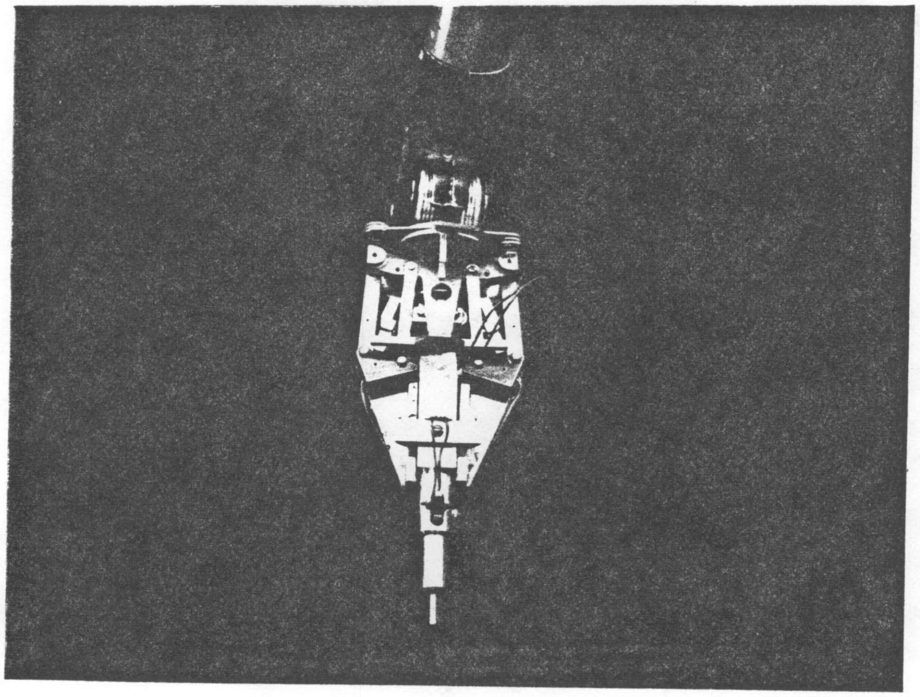
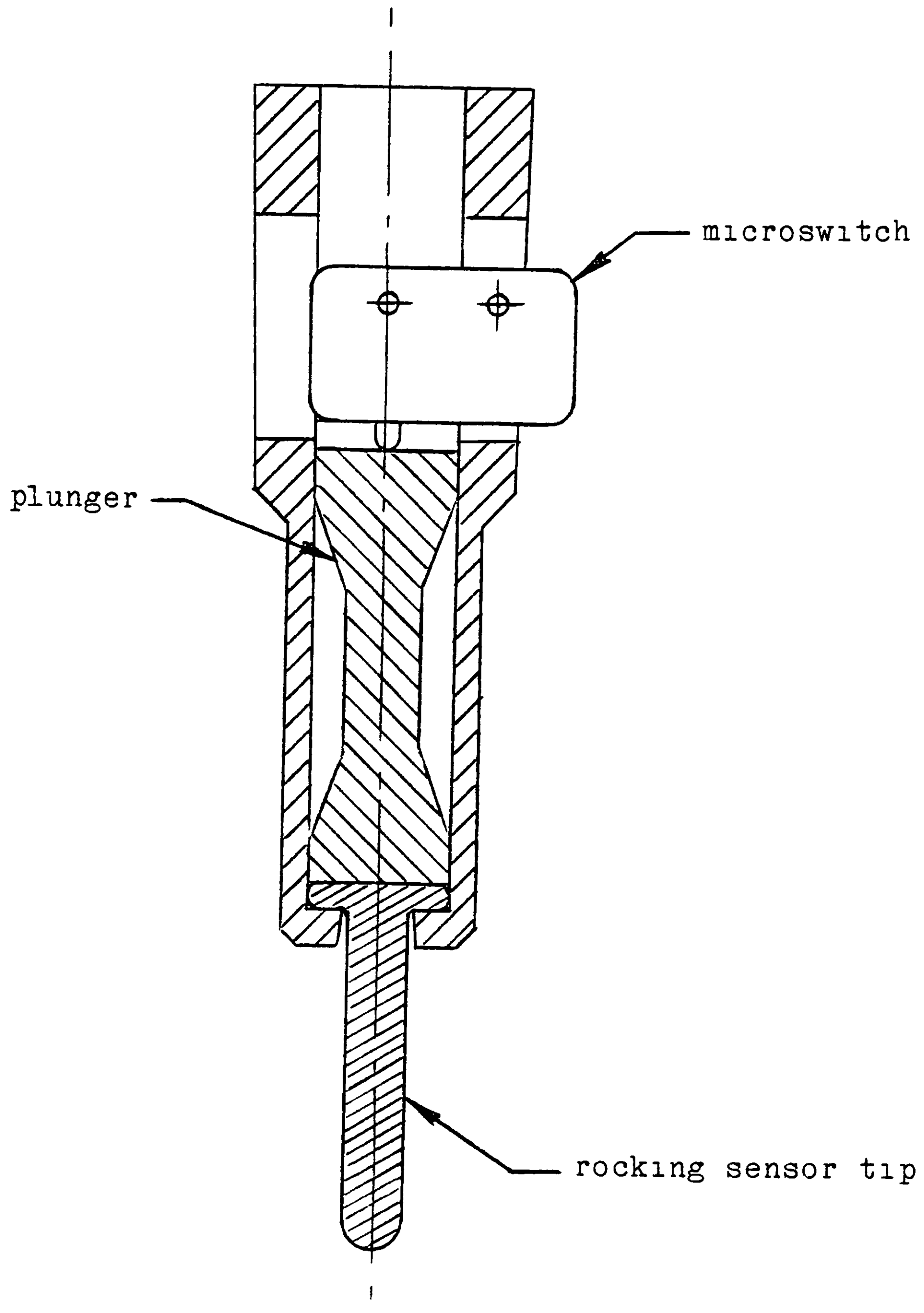


Fig. 4.7: Single Switch Touch Sensor



SCALE 2/1

Fig 4 8 Switch Mechanism that is Sensitive to Touch from All Angles

# CHAPTER 5 CALCULATION OF POINT COORDINATES

## 5 1 Description

The picture of the manipulator was refreshed about every 20 milliseconds while the touch sensor program was running To do this, the new angles of the manipulator had to be read every cycle The coordinates of a touch point would be calculated during the cycle also whenever a touch sensor was activated This was done by computing the sequential angular transformations from the base of the manipulator to the touch sensor tip Intermediate transformations from each manipulator link were saved so the manipulator itself could be drawn on the graphic display The coordinates of touch points were calculated and stored using the manipulator base as a relative origin and the x, y, and z axes were as shown in Fig 5 1 Only integer values could be sent to the display processor so length units were chosen such that there were 40 units per inch These units were chosen to minimize round off error and at the same time not overrun the display processor maximum length values, (plus or minus 2048) The basis for the dynamic display of this manipulator was developed by C Winey and is explained in some detail in Ref [1]

## 5 2 Problems with the Manipulator

The manipulator that was used to maneuver the touch

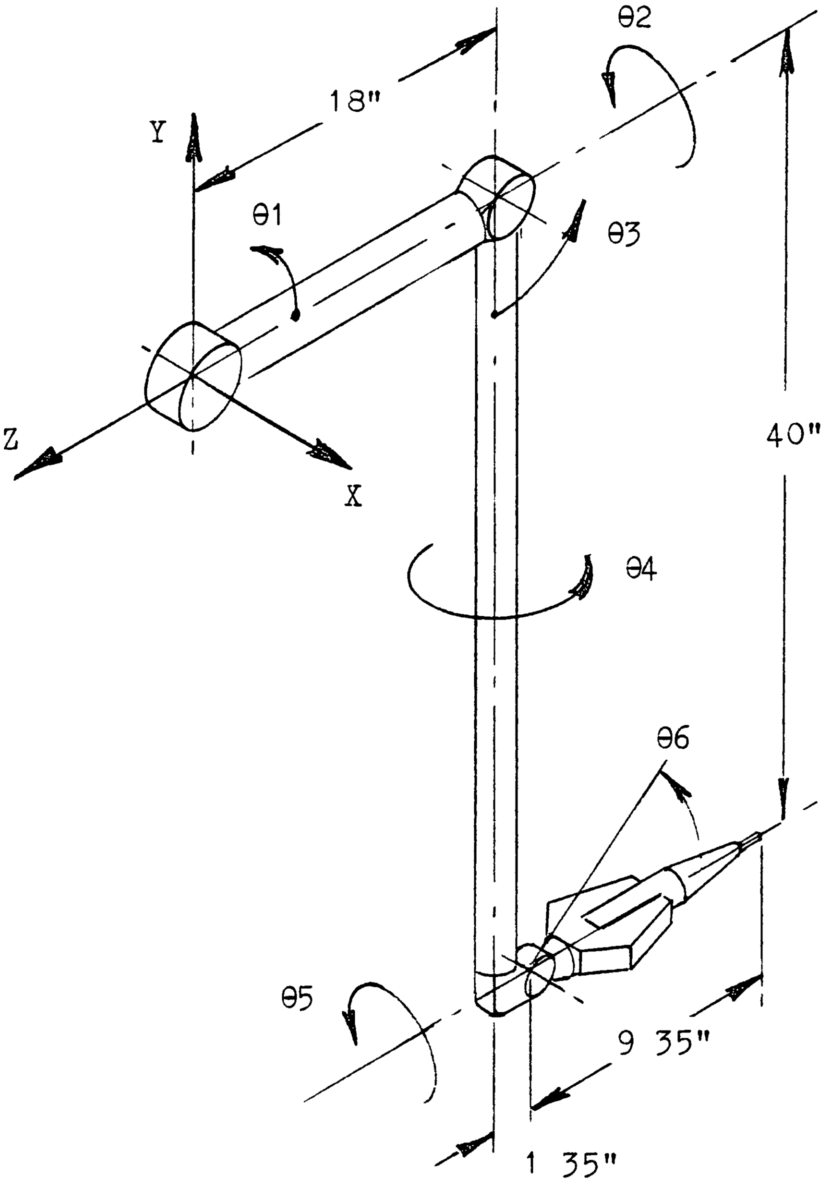


Fig 5 1 Manipulator Coordinate System

sensor was built to be controlled by a human who would have direct visual feedback as to where he was moving it This type of control system did not require accurate positioning because it was assumed the operator would compensate for errors Consequently the manipulator was not very good for finding absolute point locations This posed some unique problems to getting accurate point angles The problem could be rectified by using a more rigid manipulator with less elasticity and "free play"

### 5 2 1 Cables and Gears

The joints of the manipulator were connected to the servos and position transducers by a series of cables and gears This allowed for much backlash and flexibility which translated into errors for recorded joint angles Any error in joint angles in turn translated into larger errors in calculated point coordinates One way these errors were minimized was to make the touch sensor sensitive to very light pressure to reduce the strain on the cables Another solution was to minimize the effect joint angle errors had on point coordinates The wrist joints were most prone to errors because they were connected with the longest cables Their effect was minimized by keeping the touch sensor as close to the base of the wrist as possible

5 2 2 Pushrod

The elbow joint of the manipulator was connected to its servo and transducer by the pushrod arraingement shown in Fig 5 2 At first it was thought that tne gear angle Ag would respond very much the same as the elbow angle Ae and that they could be considered as equivalent For relative motions this worked well enough but for calculating absolute point locations, the long forearm length multiplied a small angle error into a large position error Fig 5 3a shows the calculated locations of points on a flat square grid when it was assumed that Ag and A3 were the same Clearly this assumption is invalid for absolute positioning

An equation had to be developed to calculate the elbow joint angle A3 from the two angles it was dependent on, Ag and the X motion angle A2 A closed solution for A3 would be very long because the linkage was 3-dimensional and relatively complex This was to be avoided if the calculations were to be done in real-time Since the angle A3 was to be calculated for small incremental cnanges on each cycle it was decided to use the previous value of A3 on some preliminary calculations when figuring the new A3 Guessing the new value of A3 could eliminate some long calculations that really did not nave much effect on the final answer The metnod used was to calculate the x, y, and z locations at each end of the pushrod using Eq 5 1

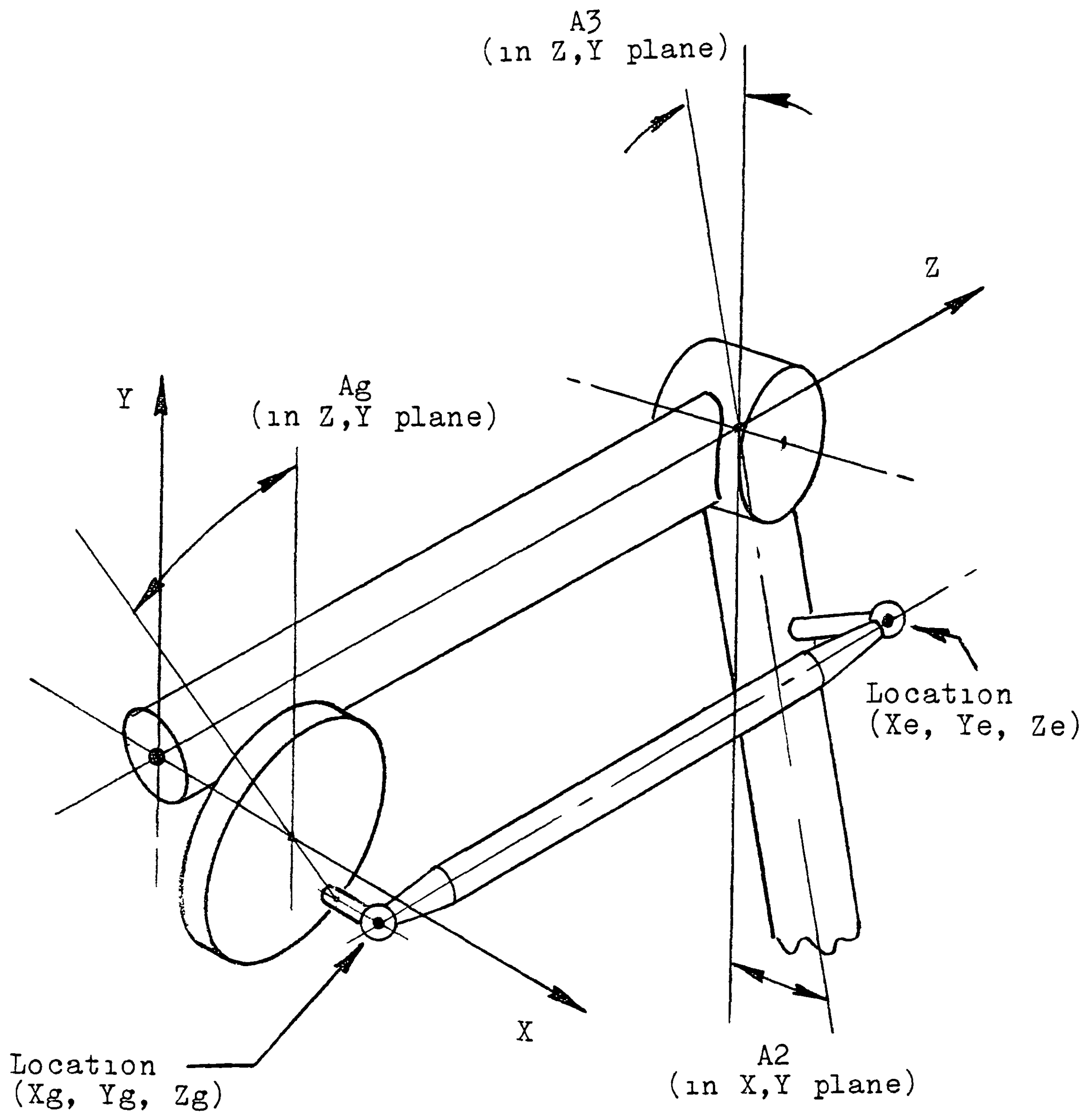


Fig 5 2 Nonlinear Pushrod

( 5 1 )

a ) $Xg = 3\ 10$

b ) $Yg = -4\ 5 \sin( Ag )$

c ) $Zg = 4\ 5 \cos( Ag )$

d ) $X3 = 2\ 25 \cos( A2 ) - 4\ 5 \sin( A2 ) \cos( A3 )$

e ) $Y3 = 18 + 4\ 5 \sin( A3 )$

f ) $Z3 = 2\ 25 \sin( A2 ) + 4\ 5 \cos( A2 ) \cos( A3 )$

The pushrod length was known to be 18 02 inches and could also be defined in Eq 5 2

( 5 2 ) $$18\ 02 = \sqrt{(X3 - Xg)^2 + (Y3 - Yg)^2 + (Z3 - Zg)^2}$$

Between Equations 5 1 and 5 2 there are 7 equations and 9 variables The two variables A2 and Ag are known so all the others should be defined if the equations are all linearly independent The problem is that A3 appears 3 times, once in a sine function in Eq 5 1e and twice in a cosine funtion in Eqs 5 1d and 5 1f This makes the problem of calculating A3 very nonlinear and makes it useful to do some guessing If it is assumed that A3 is usually near zero, then small errors in A3 will have little effect on cos(A3) That means it should not make much difference if the value of A3 from the previous cycle is used to calculate cos(A3) in Eqs 5 1d and 5 1f If this is done then it is a straight foward problem to calculate the new value of A3 from Eq 5 1e Equation 5 2 can be converted to

$$( 5\ 3 ) \qquad Y3 = Yg + \sqrt{18\ 02^2 - (X3 - Xg)^2 - (Z3 - Zg)^2}$$

And from Equation 5 1e ,

$$( 5\ 4 ) \qquad A3 = \arcsin((\ Y3 - 18\ )/4\ 5)$$

This method of calculating A3 worked very well even when the angle of A3 went up to 60 degrees A value of A3 was converged upon fast enough that only one iteration per cycle was required Figure 5 3b shows how points were located on a square grid with the angle A3 computed with the above routine

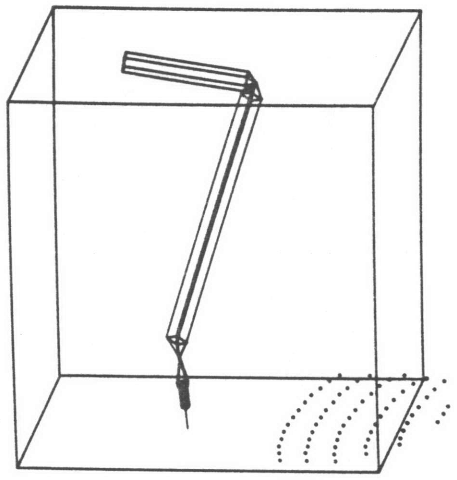


Fig. 5.3a: Grid Errors Due to Pushrod Nonlinearity

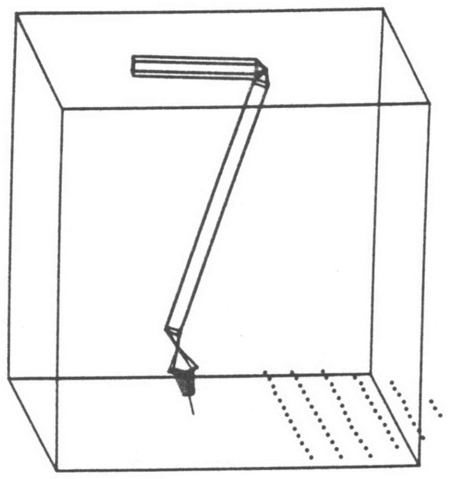


Fig. 5.3b: Grid Errors Reduced with Compensation

# CHAPTER 6 POLYHEDRA CONSTRUCTION

## 6 1 Introduction

The previous chapter described a method of finding 3-dimensional point locations on a surface It oecame apparent later that it would be very useful to have a way of describing the surface the points where found on To have a geometric description of the surface would make it feasible to delete hidden lines and surfaces because a definite edge would be defined It would also provide a basis for deciding inside from outside and make it possible to calculate volume and surface area

First, simply connecting each point to its three nearest neighbors on the graphic display was tried This had disappointing results because the lines tended to cluster in small bunches and didn't interconnect very much The approach was discarded because it didn't give any semblance of a closed object and was no better than bare dots for making a recognizable picture

It is a trivial problem for human to connect a given set of points with lines to make a closed shape so it would seem that a solution solvaole by a computer would be possible The problem is a human can make a judgement based on the whole set of points at once while a computer can only operate on a very small portion at a time This means an iterative process must be found to construct the surface with the aid of a computer

It was decided to treat the surface as a geometric polyhedron (this is what the surface would come out as anyway if the surface is constucted properly) Also, a constraint was imposed that the polyhedron surface be made up entirely of triangular facets This was done because it provides the computer the simplest possible surface segments to process Also, triangular facets give the greatest resolution for a given number of points A four sided facet connecting four dots would be the same as two triangular facets without the cross line

## 6 2 2-Dimensional Solution

The 2-dimensional solution to the problem will be shown first because it has many analogies to the 3-dimensional solution but is much easier to explain In this case there are points scattered randomly on the edges of a flat area in two-space The problem consists of finding the best way of connecting the points to enclose the area and describe its edge, see Fig 6 1

The problem is fairly trivial if the area in question is completely convex The correct way to connect any combination of edge points will always come out a convex polygon and any wrong solution will have some lines that cross over one another This suggests an algorithm where a computer could try every possible line connection combination until it came across a solution where there were no crossing lines The trouble is that the number of required trials would go up exponentially with the number of points to connect

The solution to this problem that is most similar to the one used to solve the 3-D problem is an iterative approach First, any 3 points are connected with lines to form a triangle Now if the area is still convex then all the other points lie outside this triangle

It is important at this stage to define inside from outside for each line because the computer will only consider one line at a time It can be seen from Fig 6 2 that the three lines of the triangle can be defined as 1-2, 2-3, and 3-1 assuming that the x-y locations of points 1, 2, and 3 are known The line 1-2 can be thought of as a vector with base 1 and end 2 Now the outer side this vector can be defined arbitrarily as its right side

After the initial triangle is made and inside and outside defined, it is a straightfoward problem to add each point onto the existing polygon An example is shown in Fig 6 2 Point 4 is to be added to polygon 1-2, 2-3, 3-1 It is apparent that line 1-2 is the only one that faces out toward point 4, (there will always be just one such line if the area is convex) Now the line 1-2 can be deleted and the lines 1-4 and 4-2 added to make a new polygon 1-4, 4-2, 2-3, 3-1 The only decisive task for the computer is to

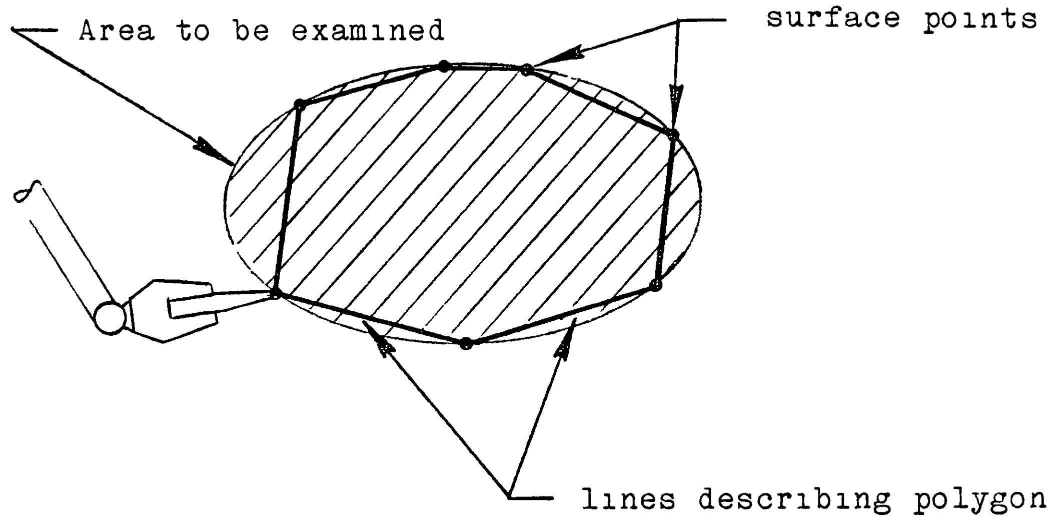


Fig 6 1 Connecting 2-D Surface Points into Polygon

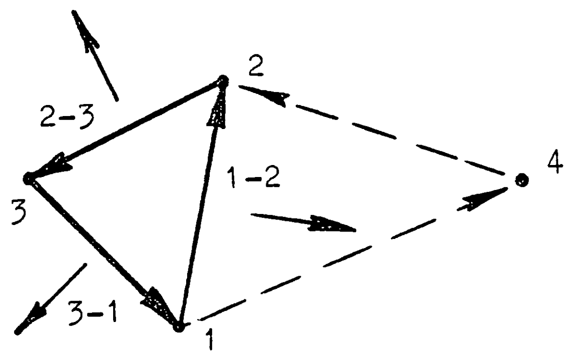

2
2-3
4
1-2
3
3-1
1
sequence specifies point connection and outer side of each line

Fig 6 2 Definition of Lines and Polygons find the line which is best to attach the point

The problem becomes more complex if areas with concave edges are allowed Many different polygons can be made from a given set of points if there are concave edges, see Fig 6 3 What can be done to limit the number of possible polygons to one?

If it can be assumed that touch sensors were used to find the points, then data about the direction from which the point was approached will be available A "touch vector" can be associated with each point to indicate its outer side, see Fig 6 4 Note that the touch vector does not necessarily have to be at right angles to the edge touched It is only the centerline of the touch sensor at the instant the point is touched Now a single polygon solution is again possible if the constraint is imposed that the touch vectors cannot pass through the polygon, see Fig 6 5 Also, for computer control, there will only be one line on the polygon available to attach a new point to, (if any) If the new point is found to be inside the existing polygon then the correct line to attach it to is the one the touch vector passes through

Some problems can occur with convex polygons It is possible to come across a point that has no line on the polygon that it can attach to without violating a rule, see Fig 6 6a In these situations, the point must be thrown out or set aside until the polygon is developed enough to

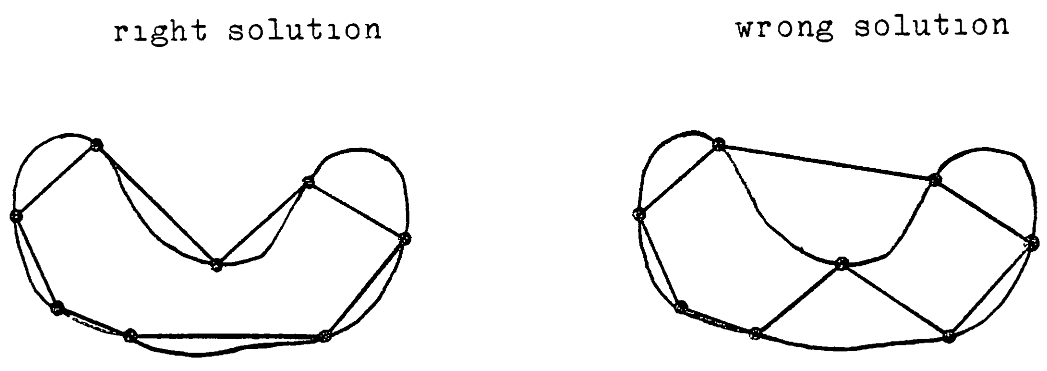


Fig 6 3 Concave Polygons

In general, there are many ways to connect points found on a concave area and still get a closed polygon

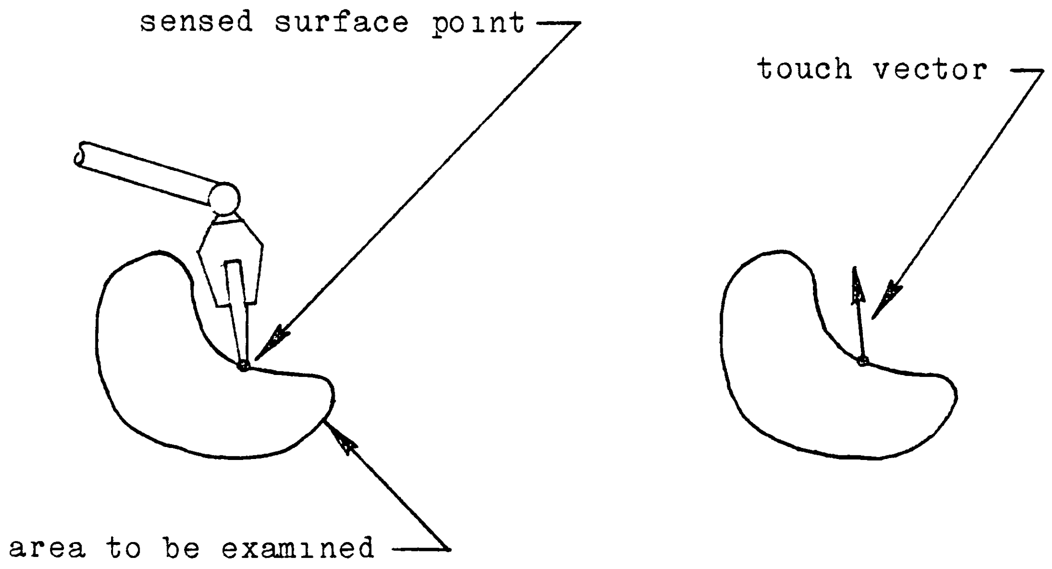
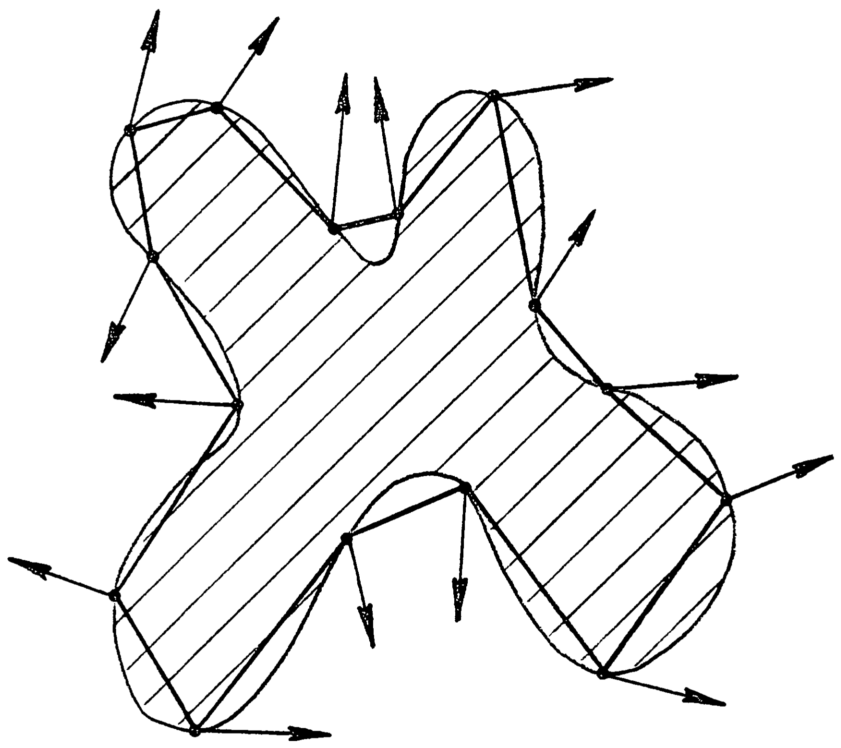


Fig 6 4 Definition of Touch Vectors



touch vectors must always point
away from the polygon

Fig 6 5 Constructing Concave Polygons with Touch Vectors

There will only be one polygon solution if touch vectors are considered accept the point Another problem with convex areas is that a folded polygon can be constructed by the computer, see Fig 6 6b The solution to this problem is to ignore any point that has a touch vector that goes through any line on the polyhedron from its outer side

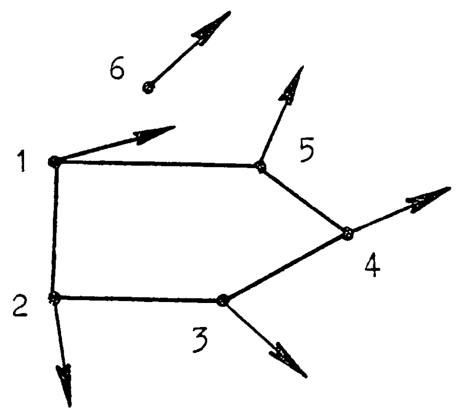
It is also possible to attach a new point to a completely erroneous line if a finite length touch sensor is used on an extremely convoluted polygon, see Fig 6 7 This problem could be solved by putting a bend in the touch vector to more accurately simulate the touch sensor and its arm An easier solution is to ignore points found to be over a certain depth inside the polygon

## 6 3 3-Dimensional Solution

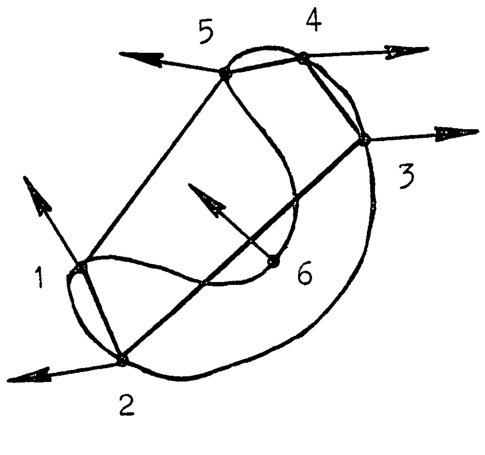
The problem here is to find a way to connect 3-D surface points with lines to make a polyhedron that closely resembles the surface the on which points were found It turned out that the best way to solve the problem was not by analyzing the connecting lines but by analyzing the facets of the polyhedron If the facets on a set of points is known then the edges are also known Triangular facets were used as stated earlier

### 6 3 1 Polyhedron Description

A method is required to store the facets in computer memory It was decided to describe the facets as a sequence



a Point 6 cannot be attached to the existing polygon without causing a touch vector to pierce through It is not likely that point 6 is even from the same area as points 1 - 5



b Point 6 cannot be attached to the polygon without turning it inside-out Point 6 will have to be ignored or saved until the polygon is further developed

Fig 6 6 Examples of Points That Cannot be Attached

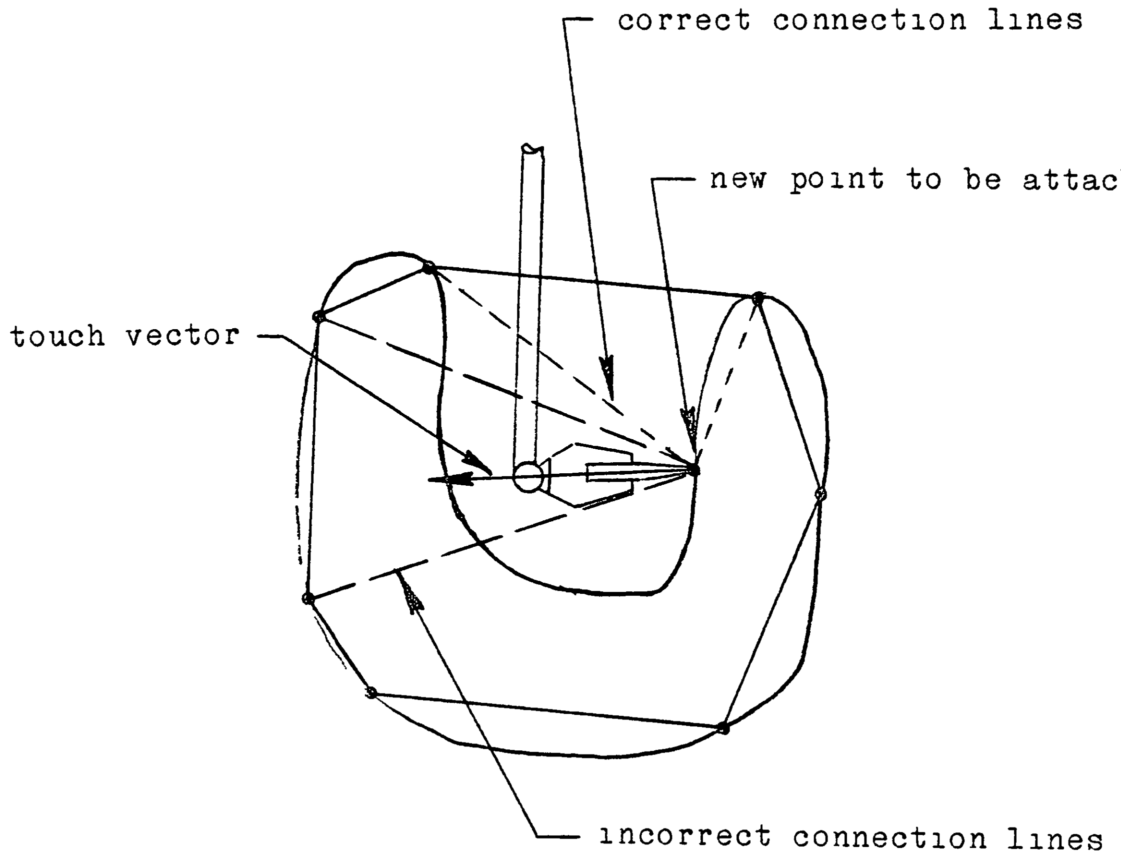
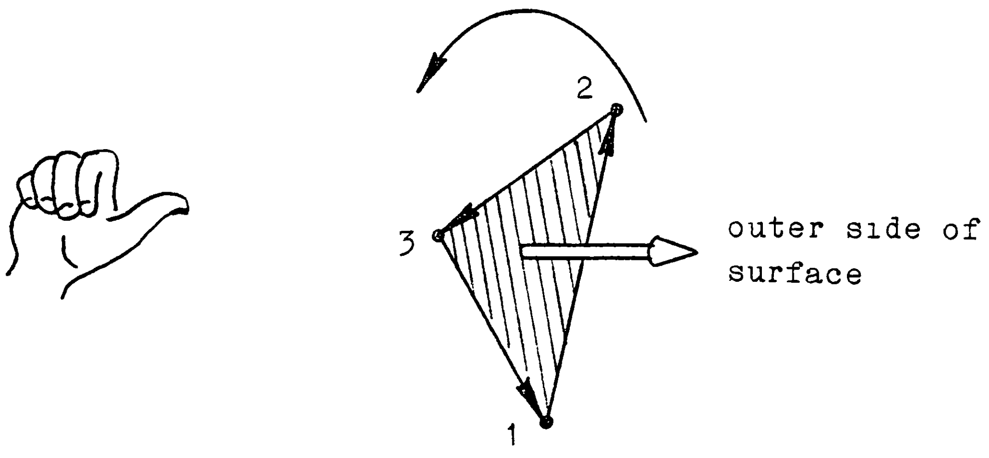


Fig 6 7 Example of an Incorrectly Attached Point

To keep the touch vector on the outside of the polyhedron, the touch point will have to attach to the wrong line This problem stems from the fact that touch vectors are considered to be infinitly long while the actual touch sensor is very short The simplest solution to this problem is to ignore or save points that are found to be deeper in into the polyhedron than the length of the touch sensor of points, because the points and their coordinates would be already known The facet 1-2-3 would be a facet with edges connecting the points 1 to 2, 2 to 3, and 3 to 1 Also the inside and outside of the facet could be defined with this number sequence using the right-hand-rule, see Fig 6 8 It can be seen that the facets 1-2-3, 3-1-2, and 2-3-1 all describe the same facet because the sequence always goes in the same direction around the triangle The facets 3-2-1, 2-1-3, and 1-3-2 describe the same facet as above but with the opposite outside surface

The computer description of a tetrahedron is shown in Fig 6 9 Note that each line on a polyhedron is given twice in the facet data, once on two different facets and always in opposite sequence It might seem easier to describe the polyhedron by storing the lines as two-number sequences rather than the apparently redundant method of storing facets as three-number sequences But it turns out to be very important to know the complete facets and this data would not be readily available with line information

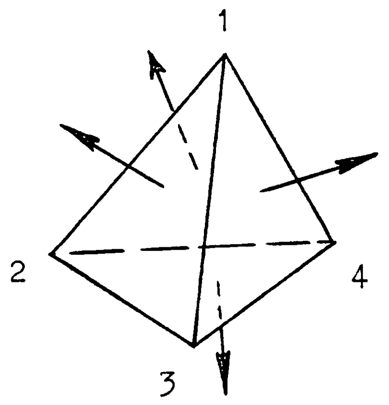
Like the 2-D solution, restraints were imposed that restricted the configuration of the polyhedron No surfaces were allowed to stick through one another and no touch vector could be allowed to exist on the inside of the polyhedron Also, like the 2-D solution, an iterative approach was used where each point was added onto an existing polyhedron one at a time



Facet 1 - 2 - 3

Fig 6 8 3-D Definition of Facets

Number sequence defines point connection and outer side of facet using the right-hand-rule



Polyhedron described by facet data
1 - 2 - 3
1 - 3 - 4
4 - 2 - 1
3 - 2 - 4

Fig 6 9 Example of Complete 3-D Polyhedron

How can a new point be added onto a polyhedron? First, it is helpfull to exploit some of the useful properties of polyhedrons as described by Euler's formula for polyhedrons, where F is the number of faces on a polyhedron, E is the number of edges or connecting lines, and V is the number of vertices or points

( 6 1 ) $$F = E - V + 2$$

This equation holds for any ordinary 3-D polyhedron that does not have any holes passing through it

Only polyhedrons with triangular facets will be considered so another defining equation is given On a polyhedron with triangular facets it can be seen that each facet has exactly 3 edges and that each edge seperates exactly two facets Thus

( 6 2 ) $$3F = 2E \quad \text{(for triangle faceted polyhedrons)}$$

Combining Eqs 6 1 and 6 2 gives two relations

( 6 3 ) $$F = 2V - 4$$

( 6 4 ) $$E = 3V - 6$$

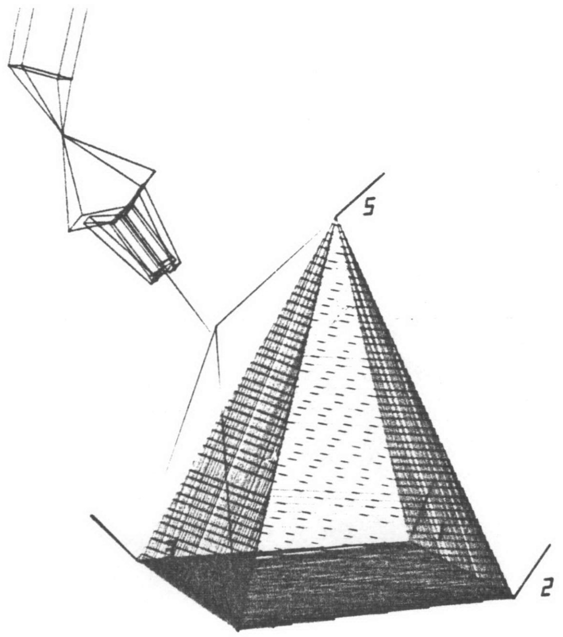
Equations 6 3 and 6 4 show that for each new point added to a triangular polyhedron there will have to be 2 more facets and 3 more lines

For the 2-D solution a point was added onto the existing polygon by deleting one chosen line and adding 2 more In effect, the point was attached to the place were one line used to be In the 3-D solution a facet must be chosen on the existing polyhedron on which attach the new point That facet is then deleted and the resulting hole is closed by adding 3 new adjacent facets that reached out to the new point, see Fig 6 10 This procedure satisfies Equation 6 3, in the total number of facets added to the polyhedron for each new point It is also apparent from Fig 6 10 that Equation 6 4 is satisfied because exactly 3 new lines are added

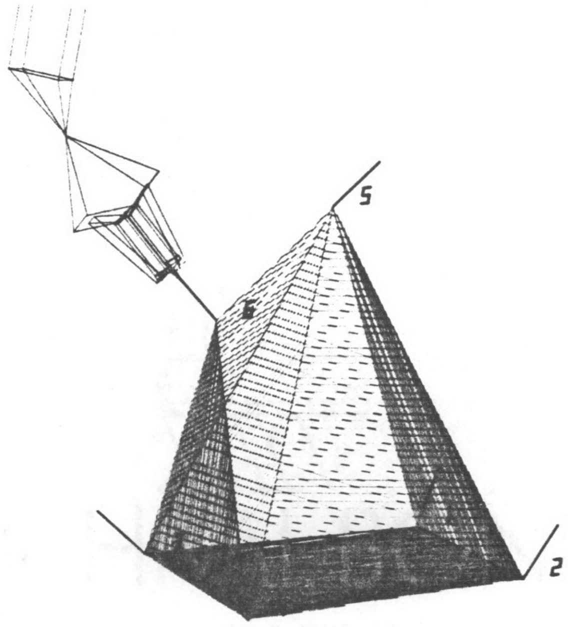
One of biggest problems was deciding which facet to attach the point to Unlike the 2-D problem there was not always a single answer, even when touch vectors were considered In general there could be several facets that a point could be attached to that would produce a closed polyhedron and would not cause any touch vectors to stick through any surface More restraints had to be incorporated to make the computer converge on a single facet

One restraint added to the program was that if a facet was pierced from the negative side of the touch vector of a new point, then that point must attach to that facet, assuming all the other restraints are satisfied This restraint worked very well in situations where the new point was close to the polyhedron and the touch vector most likely passed through the best facet

Sometimes, though, the new point was so far away that its touch vector did not pass through the polyhedron at all and if it did, the facet it pierced through was not likely to be the best To cover these situations a secondary



a. Point 6 shown with chosen facet for attachment



b. Completed attachment

Fig. 6.10: Addition of new Points to the Polyhedron

restraint was added which required that the new point attach to the facet with the nearest centroid

If the new point was very far away from the polyhedron, there would be very little chance the new point would attach to a good facet, see Fig 6 11 The solution to this problem was to ignore points over a specified distance away Taken together, these restraints caused the computer to converge on a single facet and usually it was the best one Even when the chosen facet did not look like the best, the next step of processing usually converged on a better solution for the polyhedron

Many times the new point was found to be on the inside of the polyhedron In these cases there was at least one facet that could be found which the point's touch vector pierced from the inside This was the only facet the interior point could attach to and keep its touch vector on the outside of the polyhedron, see Fig 6 12

## 6 3 2 Initializing the Polyhedron

The above procedure worked only at adding points to an existing polyhedron A seperate algorithim was required to create a starting polyhedron from a set of initially unconnected points The method used only required 3 points to make an imaginary two sided polyhedron The computer was simply instructed that there were two facets, one on each side of the triangle defined by the 3 new points, see Fig

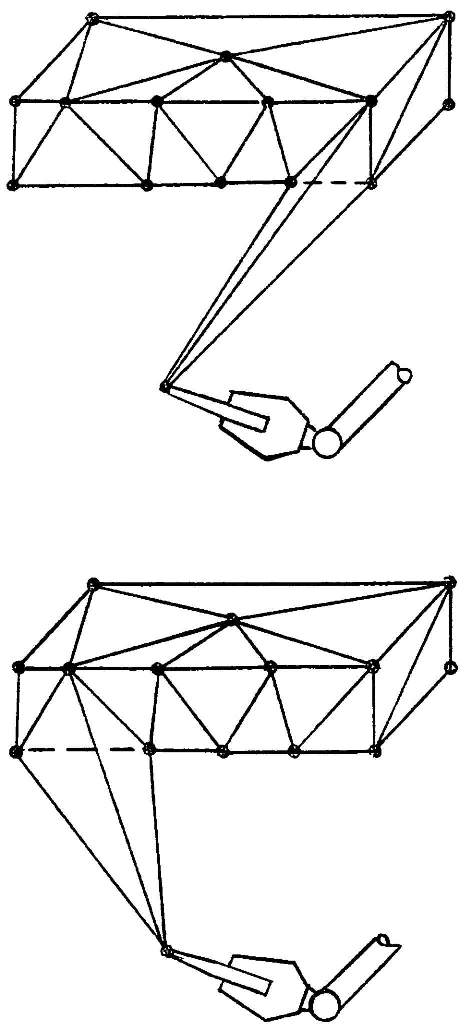


Fig 6 11 Possible Errors from Attachment of Distant Points

In general, it is very difficult to make a rational decision on which facet to attach a distant point to The choice, though, can have a drastic effect on the resulting shape of the polyhedron The easiest solution to this problem is to ignore points that are over a certain distance from the polyhedron

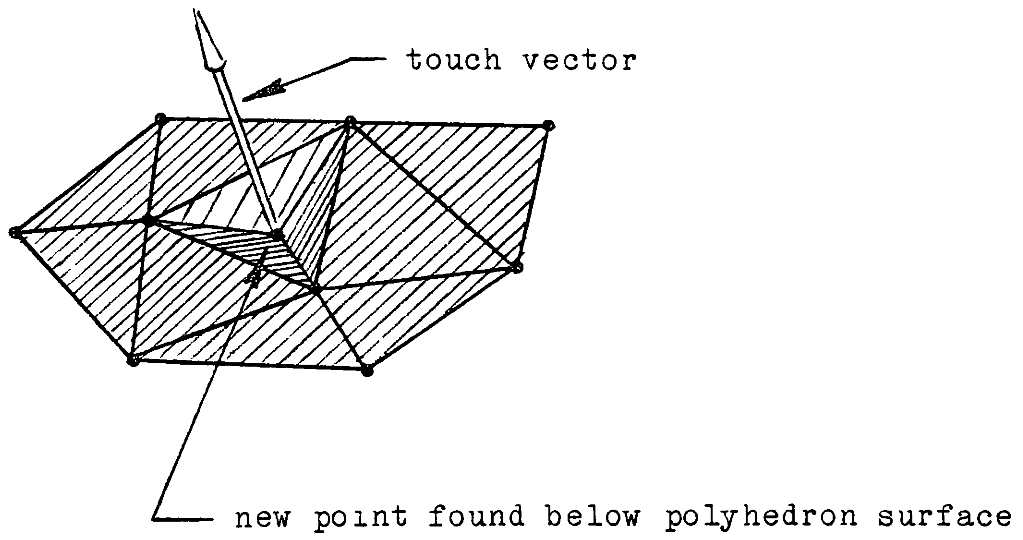
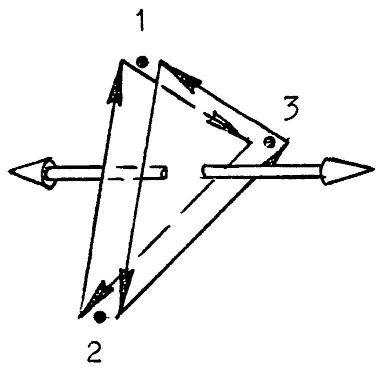


Fig 6 12 Attachement of Interior Points

Interior points must always attach to the facet that the touch vector pierces through



initialization facets
1 - 2 - 3
3 - 2 - 1

Fig 6 13 Initialization of Polyhedron

First 3 points are connected with 2 facets to make psuedo-closed polyhedron

6 13 The computer had no capacity to reject such an impossible polyhedron once it had been installed Any 3 noncolinear points in space can be connected this way and will not technically violate any of the stated polyhedron rules This entity also satisfied Equations 6 3 and 6 4 which specify the correct number of verticies, edges, and faces for a real polyhedron

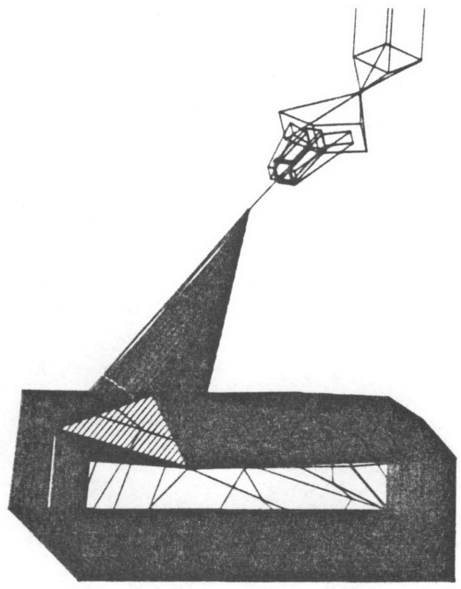
When the 4th point is added on, the computer will use the usual algorithm to erase one of the coplaner facets and add 3 more to make a tetrahedron The reason that a tetrahedron was not used for initialization is that too much programing space would be required make sure the shape was not inside out and also that none of the touch vectors where piercing through

## 6 3 3 Checking Facet Pairs

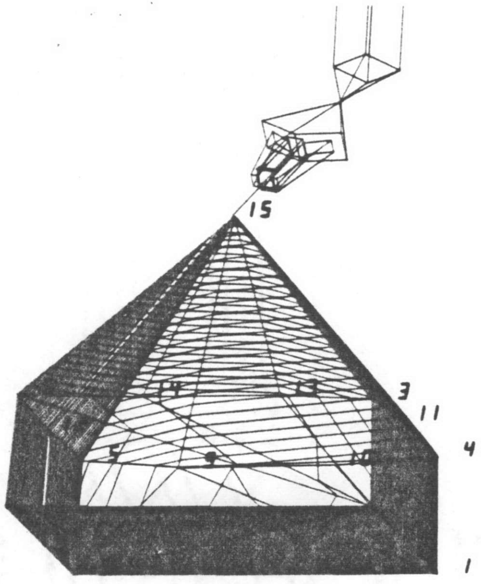
After a new point had been attached to the polyhedron, the facets were not usually in the best configuration The new point could be sitting on the top of a long spike or otherwise looking as though it was stuck on as an afterthought, see Fig 6 14

Since there were usually many possible polyhedron configurations that a given set of points could be built into, some new critera had to be used to make sure that one polyhedron solution was decided upon

The method chosen to modify the polyhedron was to check



a. New point attached to polyhedron without smoothing.



b. After smoothing.

Fig. 6.14: Need for Smoothing of Polyhedron

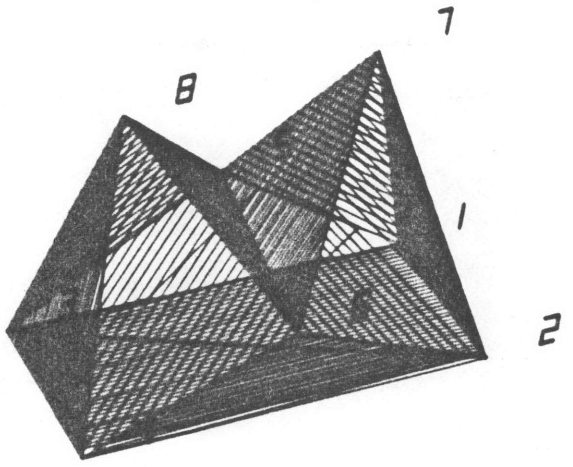
adjacent pairs of facets and, if required, replace them with compliment facets Figure 6 15 shows how the four corner points connected by any two adjacent triangles could also be the corner points of two other completely different triangles The facets 8-6-5 and 5-6-7 are the starting facets and 8-6-7 and 8-7-5 are the compliment facets An entire polyhedron could be modified bit by bit by changing facet pairs and the polyhedron would never have to be considered as a whole

The primary criterion used for deciding if a pair of facets should changed was based on the idea that a polyhedron with the smoothest surface will be the best In other words a polyhedron would be seached for that had a minimum average angle between facets This was done by comparing the pair of facets, considered for changing, to their four neighboring facets The algorithm checked the angular difference between

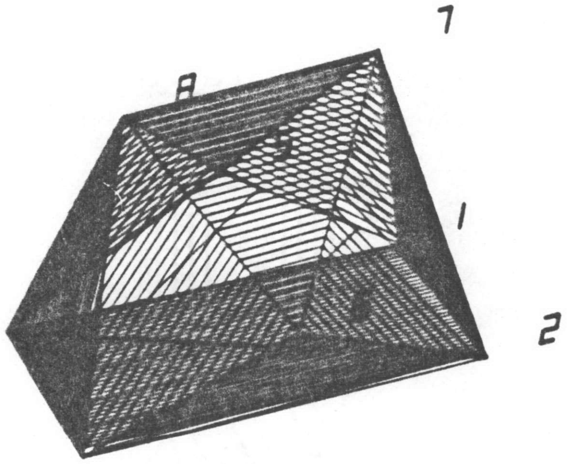
1) the original facets
2) the compliment facets
3) the neighboring facets and the pair to be checked
4) the neighbooring facets with the compliment facets

This gave 5 angular differences to average for each of two polyhedron surfaces If the complimentary facet arrangement was found to have less average angular difference, then the facets would be changed

Several checks had to be performed when it was decided



Facet Pair 8-6-5, 5-6-7



Compliment Facet Pair 8-6-7, 8-7-5

Fig. 6.15 Example of a Facet Pair and Its Compliment

to change a pair of facets  New facets could not be allowed to stick through another surface of the polyhedron  Also, a check had to be made that none of the touch vectors of the points on the polyhedron pierced through the new facets  The change in facets would be stopped if any of the above happened

It was possible to come across a pair of facets that had no reasonable compliment  These facet pairs were not considered changable and were found by checking to see if any of the compliment cross-lines were already occupied by other facets

It would not be expected to find a touch vector that lay at an angle of greater than 90 degrees to the surface normal of an adjacent polyhedron facet  The computer, though, would construct a polyhedron this way if not instructed to consider touch vector angles  Therefore, another restraint was added that any facet pair had to be made convex if it had a corner point with a touch vector that pointed away from its surface normal at greater than 90 degrees

The above requirements had to have certain priorities because they very often conflicted with one another  The order of priority was

1) The polyhedron must remain a closed object and cannot be allowed to fold on itself or wrap inside out  Also all touch vectors must exist on the outside of the

polyhedron and cannot be allowed to stick through

2) Any facet pair with a touch vector that pointed away at greater than 90 degrees from its surface normal had to be convex

3) The facet pair that had the least average difference between themselves and their four neighbors had to be chosen

When one pair of facets were converted, it affected all the neighboring facets as to whether they still followed the above requirements This meant all these facets had to be rechecked

The routine used to decide which facets to check was fairly simple First all the facets were checked around the spot where a new touch point was added to the polyhedron Then, if one of these facets was converted, all its neighboring facets were put in a list of facets to be checked The routine stopped when the list was empty Sometimes a pair of facets to be changed could get skipped over because the list was limited to 30 points These facets would be found by using an operator controlled option that checked every facet pair on the polyhedron to catch any that were incorrect

There was some concern that a polyhedron might be formed that would have a chain of mutually dependent facet pairs In other words each facet change would cause the neighboring facets to change and an endless loop of changing facets would be formed  The existence of such a polyhedron has not been proven but it was never observed to occur  The computer program would always converge on a polyhedron where all the facet pairs satisfied the requirements

## 6 3 4 Quality of the Polyhedron Shapes

It might seem that there would always be one solution that the computer would converge upon  This was not always true  Sometimes the polyhedron would get into a bad shape the computer algorithm could not get it out of  This due to the fact that the computer algorithim based its decisions on only one pair of facets at a time  There was no way for the computer to get to better facet configuration if the first facet change meant putting the polyhedron in an impossible shape

The method used to keep the polyhedron from locking into bad shapes was to make sure that new points were not added an unreasonable distance away from the existing polyhedron  If the maximum distance was held to within the general feature dimensions of the object being touched then the points would attach onto reasonable areas  It would be very difficult to attach a new point to a developed polyhedron in the right place if the polyhedron was roughly one foot across and the new point was more than two feet away, see Fig  6 12

# CHAPTER 7 METHODS OF DISPLAY

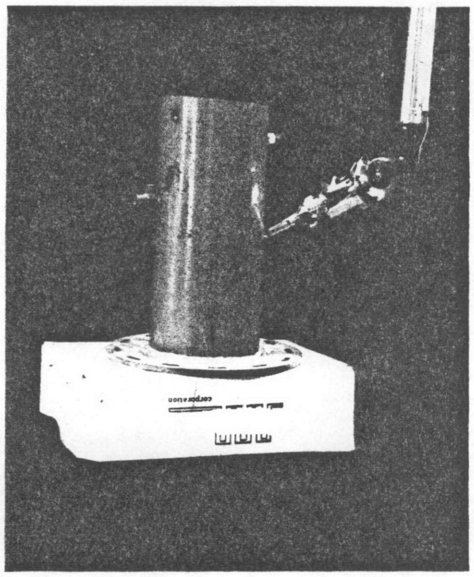
## 7 1 Introduction

The 3-dimensional information needed to completely describe points and polyhedrons in space can be easily stored as data in a computer But if these data are just displayed as lists of numbers, it will be absolutely meaningless to a human A graphic display can show 3-dimensional data much better but suffers from the fact that it can only display a 2-dimensional picture This chapter will consider different methods of bringing out 3-dimensionality for data to be shown on a graphic display

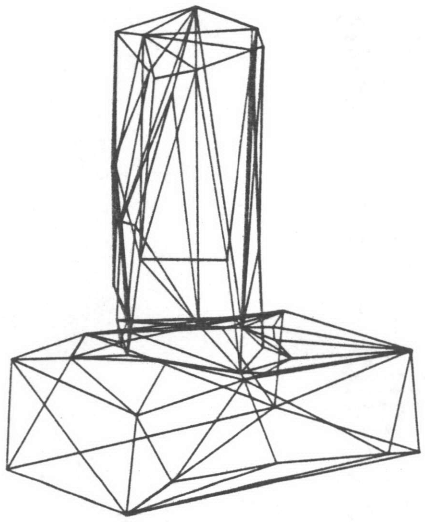
## 7 2 Problems with Polyhedra Displays

Most of the methods used to display 3-dimensionality described here were developed long before it was possible to create polyhedra from point data It would have been very difficult to understand what was happening in the program without it This was because it was impossible to tell what the computer was constructing in 3-D, without a good method of viewing it A polyhedron drawn on a vector graphic display just looked like a mass of connected lines if hidden segments were not removed There was no way to tell if one triangle was sticking though another triangle in 3-space when only one flat view was available, see Fig 7 1

There are several ways to improve the depth of a flat


a) Photograph

b) Points Only


c) Polyhedron

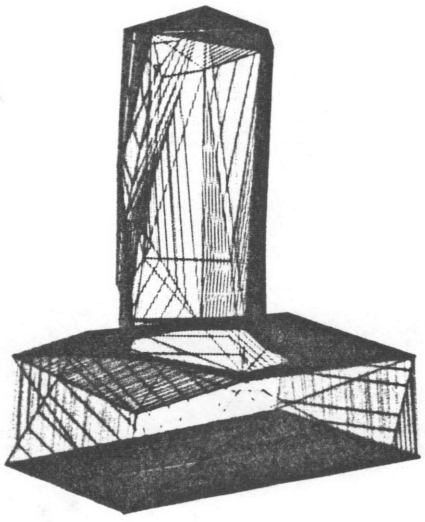

d) Polyhedron with Contours

Fig. 7.1 Different Displays for One Set of Facets

picture  Showing perspective is one way out it is best suited to rectangular shapes  Triangles shown in perspective just look like slightly different triangles  Deleting hidden lines and providing shading are methods that bring out depth for a human but can be very slow to process in real time  A method using raster graphics to remove hidden surfaces will be shown later in this chapter but was only good at getting a static picture  C Winey [ 1 ] did studies on showing two orthogonal pictures on the screen at once and displaying a shadow to help define 3-dimensionality  These methods worked well for displays where related features could be distinguished in each view and were used successfully for maneuvering the touch sensor on the screen  It was difficult, though, to distinguish related points on a complex polyhedron shown in dual views

## 7 3 1 <u>Rotating the Picture</u>

It was found that rotating the polyhedron on the screen helped to bring out its 3-dimensionality  Features in the back of the picture moved one way and features in front of the picture moved the other way  Specific details could be seen also if the picture was rotated a full 360 degrees  For example, it could be seen whether or not a line was piercing a triangle if the picture was turned completely around  If a line was not piercing a facet, then there has to be at least one place in the rotations on the screen where the line does not lay across the facet

To give the appearance of a rotating picture, the object coordinates were calculated for a small incremental angle change and the picture was redrawn on the display It was possible to redraw the picture rapidly enough to give the illusion of smooth rotation The object could be viewed from any angle if it was first rotated about an axis This could be done by multiplying the X, Y, and Z coordinates of the object by a rotation matrix [ T ] to get the new coordinates X', Y', and Z'

( 7 1 ) $$[\ X', Y', Z', 1\ ] = [\ X, Y, Z, 1\ ][\ T\ ]$$

where

( 7 2 ) $$[\ T\ ] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(A) & -\sin(A) & 0 \\ 0 & \sin(A) & \cos(A) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

( for rotation around the x axis )

( 7 3 ) $$[\ T\ ] = \begin{bmatrix} \cos(A) & 0 & \sin(A) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(A) & 0 & \cos(A) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

( for rotation around the y axis )

( 7 4 ) $$[\ T\ ] = \begin{bmatrix} \cos(A) & -\sin(A) & 0 & 0 \\ \sin(A) & \cos(A & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

( for rotation around the z axis )

The orientations of the display coordinates and the object coordinates used for the above equations are shown in

Fig 7 2a A positive rotation is defined as counter-clockwise when looking down that rotation axis

Since the display consisted of points in space either connected or disconnected, all that was required to be transformed was the coordinates of the points The "connectivity" would not change no matter what the angle of view

A combination of rotations could be made by multiplying the rotation matrices together The equation,

( 7 5 ) $$[\ T\ ] = [\ Tz\ ][\ Ty\ ][\ Tx\ ]$$

is equivalent to a rotation around the z axis, then around the y axis, and then around the x axis It is important to keep the order of multiplication straignt or different views will result

It is convenient to describe all the terms of the transformation matix as shown in Equation 7 6

( 7 6 )
$$[\ T\ ] = \begin{bmatrix} XX & YX & ZX & 0 \\ XY & YY & ZY & 0 \\ XZ & YZ & ZZ & 0 \\ XT & YT & ZT & 1 \end{bmatrix}$$

The terms XX thru ZZ handle rotations and their values are usually determined by equations 7 2, 7 3, and 7 4 TX, TY, and TZ are translational values that define the position of the object relative to its own coordinate system These are important if it is desired to zoom in on a small section of the object A zoom effect is possible by multipling all terms of the transformation matrix by a size factor

The Megatek Display Processor had the capability to do rotations in hardware The 3 coordinates of all the points defining the features of the object were first stored in Megatek memory Then it was given the required rotation terms and the Megatek would take care of calculating the transformations for each point This saved having to do the calculations for each point in software and also reduced the amount of data that had to be sent to the Megatek Very fast and smooth rotations were possible regardless of the complexity of the display The transformation terms required by the Megatek were XX, XY, XZ, XT, YX, YY, YZ, and YT The rest of the terms only affect the z plane of the display which cannot be seen on a 2-D screen

The Megatek rotations always occured around the origin of the object as it was installed in display memory This was inconvenient because very often a small portion of the display would be zoomed in on and would also need to be rotated With the object rotating about its center the small portion would generally rotate right out of view The cure for this was to cause the object to always rotate around in screen origin The XT and YT terms sent to the Megatek affected the x-y positions of points in screen coordinates These terms could be altered each time the picture was rotated to keep the object on screen center To do this, the translations in object coordinates had to be specified, $(X_o, Y_o, Z_o)$ This was done by manuvering the desired rotational base to the center of the screen by viewing two orthogonal views Now the picture would always rotate about that base if XT and YT were recalculated every iteration by the equations,

( 7 7 ) $$XT = X_oXX + Y_oXY + Z_oXZ$$

( 7 8 ) $$YT = X_oYX + Y_oYY + Z_oYZ$$

XX, XY, XZ, YX, YY, and YZ had to be calculated first for that rotation

There is a problem with dynamic pictures that are rotated with the above transformations There will be no indication which is front and which is back on an object when no hidden lines are removed and no perspective is shown An object can be rotating on the screen and some people viewing it will say its rotating to the left while others say its rotating to the right The mind tends to lock on one rotation and can be difficult to change One way found to remedy the problem was to memorize the correct rotation for each input but it was too easy to forget The most useful method was to have a known zero position the the picture could be put in, where front and back were known Another solution might be to have a coordinate indicator on the screen consisting of writing Front and back are easily distinguished with writing because it cannot be read when it is shown reversed

7 3 2 Types of Rotation

Some methods of rotating the picture were better than others at showing depth qualities If the picture was rotated about the z axis of the screen, there would be no changes made to the picture due to its depth The same rotation could be done with a flat picture Rotation on the x axis or y axis were better because points at different depth locations would move at different speeds It was best to have the center of rotation somewhere near the middle of the object to get maximum contrast of motion due to depth

Rotation around the x axis could be very disorienting because the picture goes upside down once every oscillation This left rotation about the y axis as the best choice of the three

Simply rotating about the y axis on the screen moved each point on the screen back and forth in the x plane It was found to be helpful to tilt the entire coordinate system on the x axis first before rotating in the y axis This caused descrete points making up the picture to move in ellipses on the screen Ellipses gave a better indication as to exactly where each point was in the picture A tilt downward around the x axis of about 15 degrees produced the most natural looking and informative picture

7 3 3 Oscillating the Picture

Rotating the picture completely around gave the best overall description of the object but when the display was being used to control the manipulator, it was hard to distinguish between front, back, and sideways, as they where always changing A better way of moving the picture was found for situations where picture orientation had to be known Instead of rotating completely around, the picture was just rotated back and forth with a sine wave controlling the y angle This way the orientation was not disturbed much and the 3-dimensions were still apparent An amplitude of 10 degrees with a period of 2 seconds produced a useful picture The problem with this display was that the operator sometimes had to wait for the full cycle to finish before getting his bearings and making another move

## 7 3 4 Rotation with a Joystick or Trackball

All the rotations done previously were controlled by the keyboard and did not require ,or allow, much direct attention Sometimes it could be very useful to be able to position the picture in any view very rapidly There was available a trackball and joystick that were built to provide this type of control They could be wired to the computer to control the display angles

## 7 3 5 Position Control

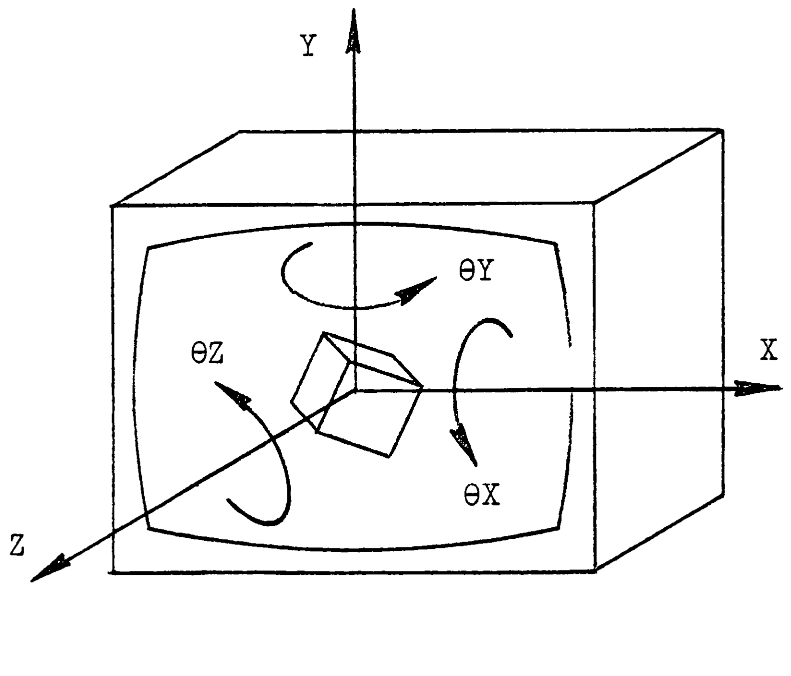
The 3-degree of freedom joystick that was used put out a voltage related to the position of the joystick This voltage was used to control the 3 angles fed into the display transformations matrix, as shown in Fig 7 2a The order of angle transformation used was, first rotation around the z axis, then the y axis, and then the x axis It was important to transform the z axis first because that made the display move most similar to the joystick A potentiometer was mounted on the joystick box to control the maximum allowable angles that the display could be put through The display could be viewed from any angle but would bounce back to zero when the joystick was let go

## 7 3 6 Velocity Control

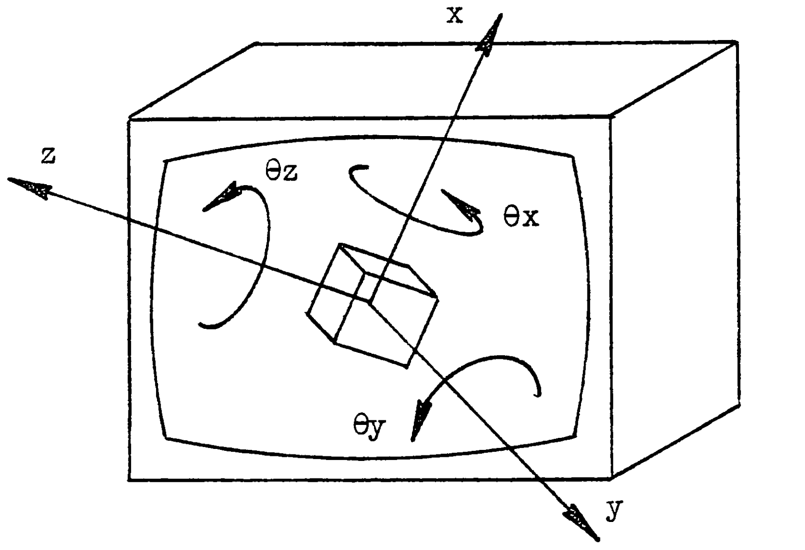
In velocity control it was most convenient to rotate the display in screen coordinates, as shown in Fig 7 2b This allowed the picture to rotate independently of of the orientation of the object coordinates When angles are changed with respect to object coordinates it was not always apparent which way the picture would turn for a given input if the object was already rotated through some other angles

The x-y inputs available from the trackball were sufficient to position the display because only two angular veloceties were required to maneuver the display when using screen rotations With the joystick, all 3 inputs were used even though they were redundant This allowed somewhat faster control of the picture position

Rotation in screen coordinates was done with the same



a Screen Coordinates and Rotations



b Object Coordinates and Rotations

Fig 7 2 Screen and Object Coordinates

transformation matrix as rotation in object coordinates The difference is that the transformation was completely recalculated each cycle when rolling in object coordinates and was only modified for screen coordinates The display transformation matrix was saved from the last iteration and multiplied by an incremental transformation matrix that was the same as the object transformation, but which reflected a very small angle change It did not matter in which order the x, y, and z rotations were multiplied by the transformation because it made little difference for small angle changes It would seem that the transformation matrix would degenerate from floating point round-off when it was continually remultiplied by another matrix but this was not observed to happen and the display did not seem to lose integrity even after many rotations

For versatility, rotations in screen coordinates were found to be the best Also, the capability to rotate directly on the screen z axis in addition to the x and y axes was useful and time-saving even though it was redundant

## 7 4 Improving the Display for Polyhedra

### 7 4 1 Showing All Edges

The construction of a polyhedron out of a set of points offered several methods of improving display quality The obvious way to display a polyhedron constructed in the method shown in Chapter 6 was to just show all edge lines The edge lines could be constructed from the facet data because each edge was defined twice, once in two different facets and always directed in opposite directions The algorithm used to connect the points on the display simply went through the data and drew a line when two connected points on a facet were found in increasing order A complete picture could be made of a polyhedron consisting of 40 facets in about 50 milliseconds

This particular type of polyhedron display was used most frequently because it was so fast to construct In fact this display was completely reconstructed every time a facet was changed It did not produce an especially clear picture but rotating it did help No attempt was made to remove hidden lines from this display because it would take too much computing time

## 7 4 2 Drawing Contour Lines

It was a straightfoward problem to draw contour lines around the outside surface of a polyhedron, because the data for each of its facets were stored in memory The only outside information required by the computer was the number of contour sections to draw The gap between sections was automaticly figured from the overall size of the polyhedron

The contours were all made on the z plane and each contour was calculated and drawn in sequence For each contour plane every facet in the polyhedron was checked to see if it passed through that plane When one did pass through then the endpoint coordinates of the line segment defining the facet cut were calculated by interpolation If the polyhedron was without holes or folded surfaces then the contour drawn at any section would be a closed polygon

Drawing contours was found to be the best way display a polyhedron on a vector graphic display The shape of the object was well defined by two aspects of the contours One was that the directions that the contour lines went in gave an indication of the angle the facets had relative to the z axis The other aspect was that the density of contour lines on one facet indicated the angle the facet had with respect to the z plane The line density of the facets also produced a sort of shading effect that gave an immediate sense of 3-dimensionality When the polyhedron with contours was rotated the picture became very well defined Any errors in the polyhedron became painfully obvious because any facets sticking through other facets could be readily seen Also if any facets folded over on top of each other the picture became very bright in that area

### 7 4 3 Raster Graphic Display

Raster graphics was experimented with to see how well a 3-D polyhedron could be displayed It was also used to show

how easily polyhedron data as described in Chaper 6 could be processed by a computer

The difference between raster graphics and vector graphics is that the raster graphics beam sweeps out the entire screen and its picture is changed by variations in intensity like a television picture The vector graphics beam traces out each of the lines and points individually An advantage of raster graphics is that surfaces can be simulated better because shading is possible and it is also easier to delete hidden lines and surfaces

The primary disadvantage of the raster display used in the experiments was that it was much slower at drawing pictures than the vector display This made real-time rotations impossible so the raster graphics was used primarily to make static copies of polyhedra

To draw a polyhedron on the raster display it first had to be constructed with the vector display The polyhedron was then framed in the vector screen to the view desired to come out on the raster display When this was done all the polyhedron data was stored in a data file The 3-D point locations were stored in screen coordinates to preserve the view chosen for the display This data was then read by a second program that put the polyhedron on the raster display The triangles of the polyhedron were drawn one at a time on the display according to their x-y coordinates The shade of each triangle was determined by comparing the angle of the surface normal to a space vector simulating a light source direction  Triangles facing away from the screen were not drawn at all  It had to be known how the facets lay in 3-space and which side of each facet faced out to accomplish shading  This was another advantage of storing facets as described in Chapter 6

If the polyhedron had any concave areas, it was likely that there were several facets partially hidden by other facets  By its nature, raster graphics will automaticly draw a new triangle right over an old one so all that is required is that hidden triangles be drawn before the non-hidden ones  The method used to draw the facets in the correct sequence was very simple  The point on each facet with the maximum z value (nearest point) was the only one considered to decide facet order  The facets were ordered such that the ones with a minimum value for this point were drawn first and ones with higher values were drawn last  There were some situations where this algorithm would give wrong answers but so long as the object to be displayed was not a radical shape and there were a reasonable number of facets defining each feature there would be no overlapping facets drawn in the wrong order  This type of shading display made the best picture when there were smooth transitions between facets

It was found to be advantageous to be able to interactively change the location of the light source to a position where the 3-dimensionality of the polyhedron was most apparent Due to the nature of the raster graphics hardware used it was very slow while drawing the polyhedron but once drawn the shades of the individual triangles could be changed very fast The trackball was used to input changes in x-y angles for the location of the light source from the center of the screen The apparent light source on the polyhedron could be changed rapidly by recalculating all the new shades for each triangle and sending them to the display The shades could be changed fast enough that the light source could be moved almost in real time, ( about 200 milliseconds to change a polyhedron with 50 exposed facets) This progressively changing light source brought out 3-dimensionality very well

Using raster graphics to display polyhedra can make them look very natural from a human point of view They can even be made somewhat dynamic by moving the light source However it was impractical to rotate the picture in real time with the equipment available

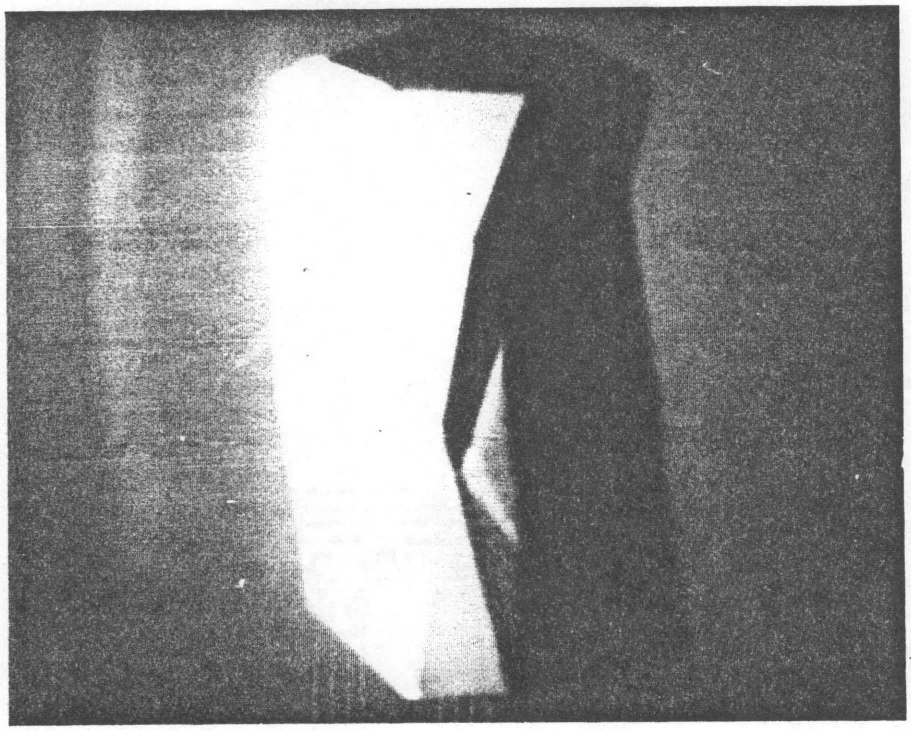


Fig. 7.3 Example of Polyhedron Shown on Raster Display

# CHAPTER 8 EVALUATION

## 8 1 Number of Points to Make a Picture

The quality of a picture consisting of points in space depends very much on the density of points in the picture If too many points were snown, the picture would be white and nothing could be seen Too few points, and the picture would convey nothing Somewhere in between is a region where there are just enough points to describe what is required to be seen

Presumably the minimum number of points is dependent on the number of distinguishing features to be shown in the picture A distinguishing feature could be any simple surface section of the object to be investigated These features would have somewhat rounded profiles and would be either flat planes or slightly curved planes Any features with sharp edges would have to be broken into smaller more-rounded features As an example, a cube could consist of six distinguishing features, one for each of its sides A sphere could consist of just one curved feature, or perhaps it should consist of several features to reduce the total angular change per feature There is no correct answer, but it is required that a degree of magnitude be found for the amount of points required to describe an object As a test, the number of points needed to describe one side of a cube and the number needed to describe the surface of a sphere were estimated and compared to get an upper and lower bound for the number of points required to describe a "feature"

Figures 8 1 thru 8 4 show how recognizable a cube and a sphere can be made with different point densities for dot and polyhedron displays It can be seen that a cube described by points does not become recognizable until there are at least 500 points on the cube Although it cannot be shown here, the cube became recognizable with only 200 points if it was rotated on the screen The sphere became apparent with only 100 points rotated or not Perhaps this was because a sphere looks the same from any view A cube shown with the points connected into a polyhedron became fairly recognizable with only 50 random points It must be kept in mind though that 8 well placed points can perfectly describe a cube The sphere still needed about 100 points to look like a sphere even when the points were connected This may be because the curved lines of a sphere are not suited for description by the straight edges of a polyhedron

Since a cube requires 500 random points to describe its surface, then 85 points are required to describe one of its six distinguishing features A sphere still requires 100 points, assuming it consists of only one feature For polyhedrons, a cube feature needs about 20 points and a sphere requires 100 It will be assumed here that all

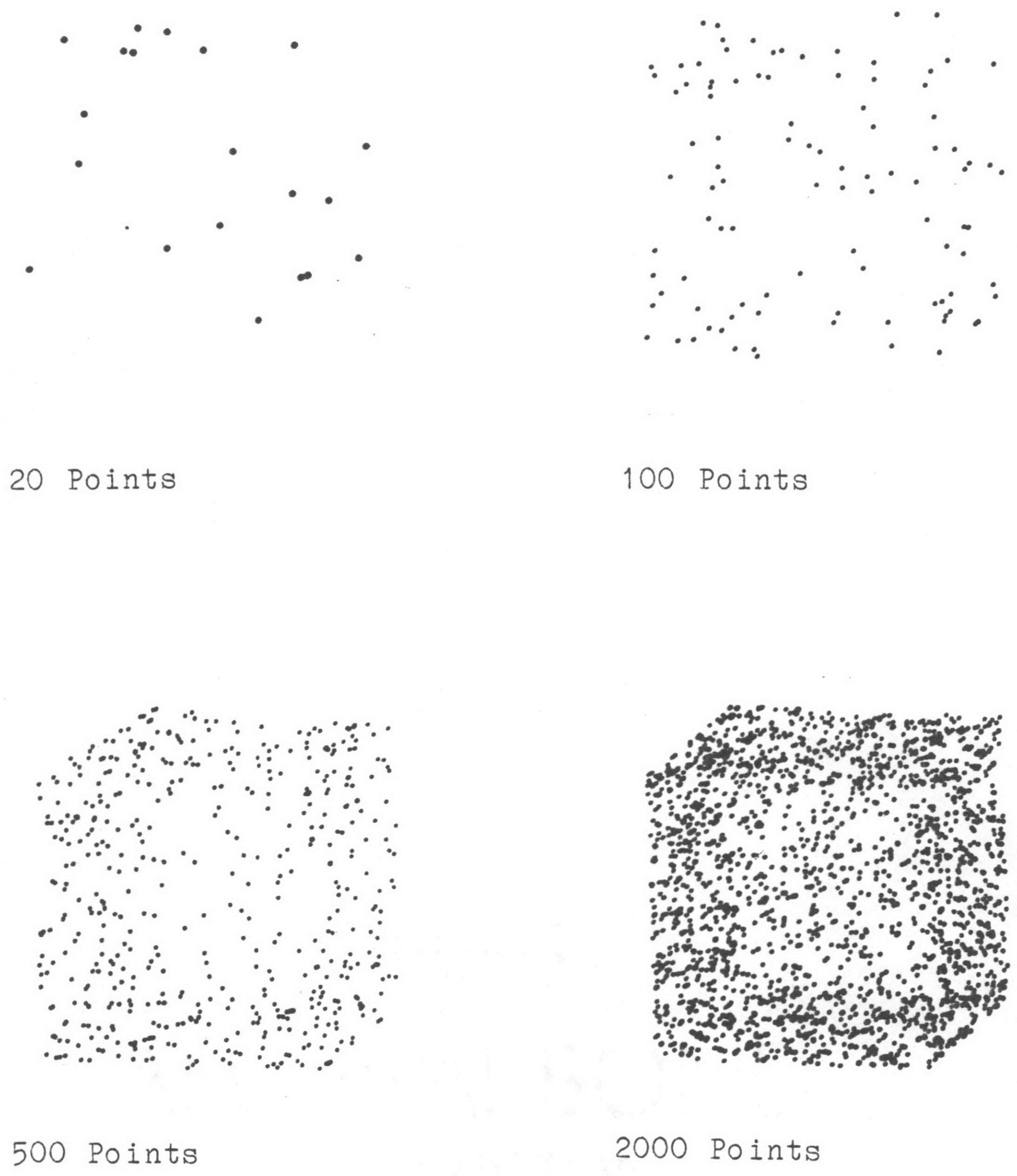
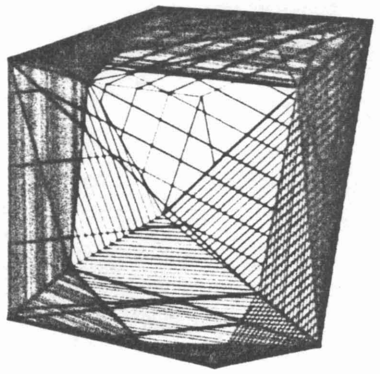
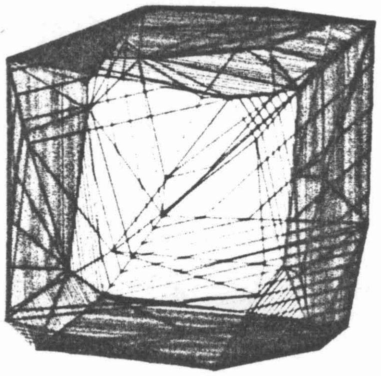


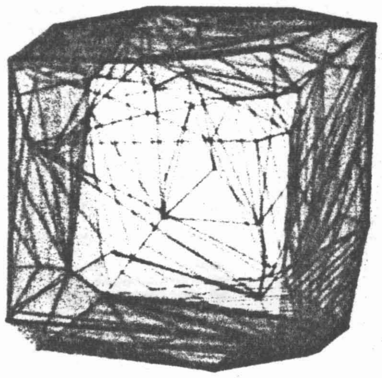
Fig. 8.1 Cubes Described by Randomly Distributed Points



20 Points



50 Points



100 Points

Fig. 8.2 Random Cube Points Made Into Polyhedron

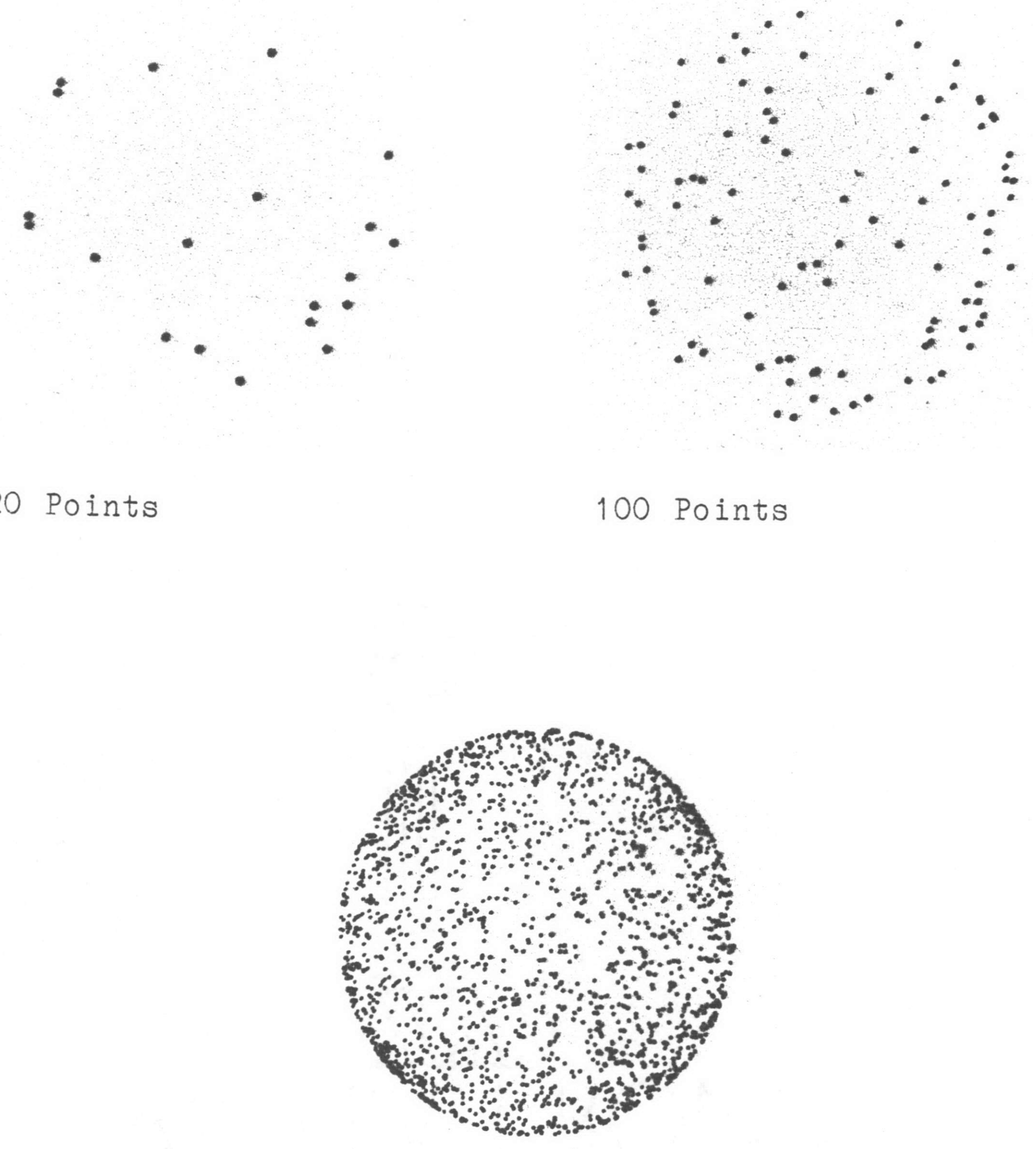
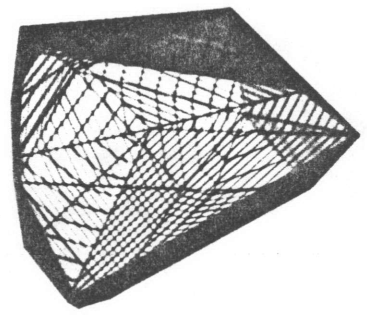
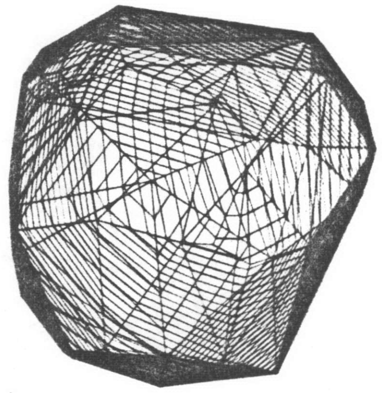


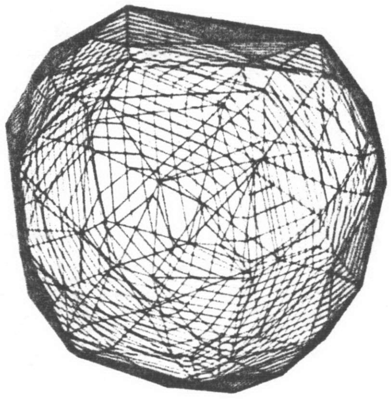
Fig. 8.3 Spheres Described by Randomly Distributed Points



20 Points



50 Points



100 Points

Fig. 8.4 Random Sphere Points Made Into Polyhedron

distinguishing features on any object require about the same amount of randomly distributed points to define its shape for a human Different sized features would also require the same amount of points, they would just have different point densities

Any object can be broken into arbitrarily small features depending on the degree of detail required Say an area in front of a manipulator must be completely described by touch points and it is necessary that all features down to three inches across must be recognizable This means the entire area must be covered with a point density sufficient to describe a 3 inch feature If the area to be investigated is 20 square feet and a suface feature is assumed to require 100 points to be well described, then the entire area would have to be covered with 32000 points to describe all features down to 3 inches across If the points are to be connected into polyhedrons, then it can be assumed that only 20 points are needed per feature, 6400 points will be required to cover the entire area

The above figures are probably exaggerated because the manipulator operator is allowed to choose where he wants to put a high concentration of points He can leave some areas with very few points if he decides they are unimportant Also, if the picture can be rotated, the number of required points can be greatly reduced

A problem unique to points that were connected into polyhedrons was that the surface of the polyhedron could become degraded if the points were too densily packed together That is, if the points were closer together than the positioning error of the manipulator, then lines connected between them would not likely lay parallel to the actual surface These points would make a very jagged surface on a polyhedron One solution would be to delete points that are to close to other points This will not reduce the resolution because it is already limited by the manipulator accuracy

## 8 2 Speed of Picture Construction

### 8 2 1 Constuction Time for Points

The amount of time required to read points from the touch sensor and then draw them on the display was very short When using a single touch sensor switch, one point could be read in at every cycle of the program One cycle took about 20 milliseconds so conceivably 50 points could be read within one second The computer could read points even faster with the brush sensor because it had 10 switches The limiting factor was not how fast computer could read points but the speed the touch sensor could respond The single touch sensor could not be moved fast enough to read more than 2 or 3 points per second and the brush sensor was not much faster because, although it could read many points at once, it was more cumbersome to maneuver

Clearly, a touch sensor is required that can read points very rapidly if a picture of a manipulator's surroundings is to be made in a reasonable amount of time A fast touch sensor could be made if it had many switches and if it was set up such that the switches did not interfere with one another, (see Chap 2) This type of sensor would be considerably more expensive than the ones used in this project but would probably be worth it for the amount of time that would be saved Another way to increase speed would be to make a sensor that could stream points in without having to lift off the surface for every point A streaming sensor would work best if it was non-rigidly mounted to the manipulator That way the manipulator would not have to follow every bend and corner encountered on the surface

As an example, assume the maximum point coordinate reading rate of the computer is 200 points per second If a touch sensor was built with 20 switches on it, then the computer would be capable of reading 10 points per second per switch This rate would not be unreasonable if the switches were made to stream points in A touch sensor capable of reading points at 200 per second could essentially cover any surface encountered with a thick mat of points in a very short time

## 8 2 2 Construction Time for Polyhedra

The speed of the computer was the limiting factor for construction of polyhedra The time period required to attach a new point went up with the number of facets on the polyhedron Fig 8 5 shows a graph of average time required to attach a new point versus the number of points in the polyhedron for the computer program in Appendix B

There are many areas of this program that could be made to run much faster at the expense of more program complexity To attach a new point, the program had to test every facet of the polyhedron for suitability This was very time consuming For this reason a condition was added that the computer only make complete tests on the five facets with nearest centroids to the new point This condition increased the speed of the program by a factor of two Other parts of the program could have used this same kind of selectivity For example, after a facet was chosen for attachment, all the other facets had to be checked to see that they did not get in the way Also, all the facets and all the touch vectors had to be checked for interference before a pair of facets could be changed These checks significantly slowed computation

Perhaps the thing that contributed most to slowing the program was the basic philosophy that points should be attached to the polyhedron in the order they were found by the operator If all the points could be known at the start

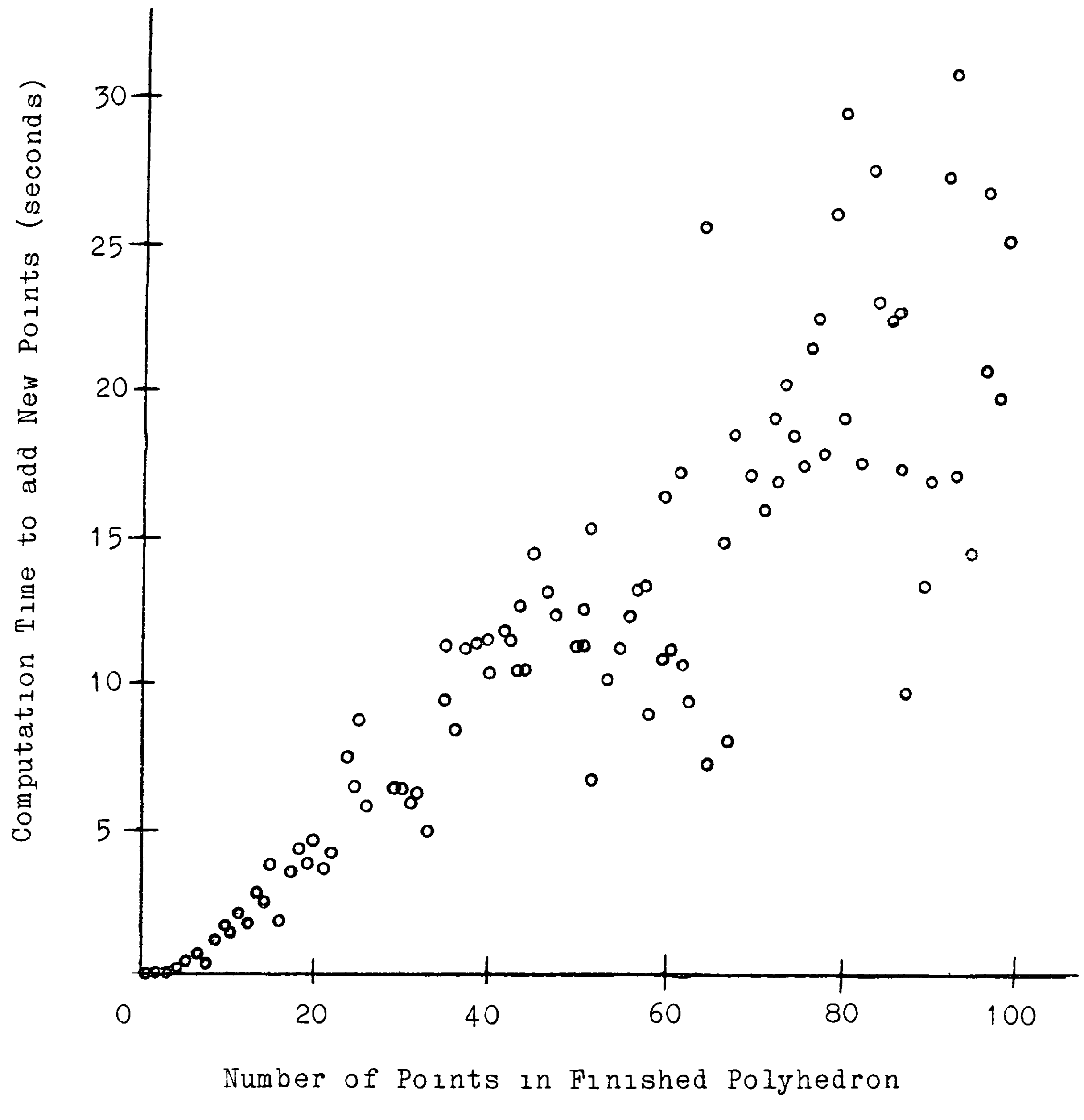


Fig 8 5 Graph of Computation Time Versus Polyhedron Size and arranged in the best sequence for attachment, many of these extensive comparisions and checks might be eliminated This might also allow the points to be seperated into small groups and connected together in patches to further increase speed

## 8 3 Raster Display

Drawing a picture of the polyhedron on the raster display was much slower than any other method tested One facet of the polyhedron could be drawn on the display in about half a second so real-time rotation of the picture was impossible Raster graphic hardware is available on the market that will draw a picture much faster but can be very complex The raster display was best used for making permanent pictures because it was capable of making them look very realistic

# CHAPTER 9 CONCLUSIONS AND RECOMENDATIONS

## 9 1 Conclusions

This project has shown that a supervisory controlled manipulator can be used to construct an understandable 3-dimensional picture of its surroundings with just the sense of touch The picture can consist simply of surface points shown on a computer graphic display It is also shown how a more sophisticated picture can be made by reconstructing a surface from these points Not only can a picture be made that is recognizable to a human, 3-D surface data that is easily digestable by a computer is also provided

In situations where vision of the manipulstor work area by the operator is difficult or impossible, these methods of touch sensor picture construction could be a good aid or replacement for the usual television camera

## 9 2 Recomendations

A touch sensor would have to be developed that could sense points very rapidly for touch generated pictures to be of practical use That way a picture could be essentially "painted" with the sensor Also the surface construction program would have to be made to go faster to be able to use it in real-time This should not be impossible as the number of required calculations to attach each point to the polyhedron can be held to a maximum value

There are many aspects of surface construction from points that could use further study

1 ) A method is needed to decide if there should be more than one polyhedron or surface in front of the manipulator This in turn leads to the problem of attaching or detaching different polyhedra from each other

2 ) An interesting problem would be to find a method to construct polyhedra with holes passing through them A polyhedron with a hole does not follow Euler's Formula

3 ) No allowance was made in this study for a moving object If the motion were known then there ought to be a way to compensate for this in the construction on the screen

## REFERENCES

1 ) C M Winey, "Computer Simulated Visual and Tactile Feedback as an Aid to Manipulator and Vehicle Control", Masters Thesis, MIT, 1981

2 ) K Tani, "Supervisory Control of Remote Manipulation with Compensation for Moving Target", Report for Man-Machine Systems Laboratory, MIT, 1980

3 ) T L Brooks, "Superman A System for Supervisory Manipulation and the Study of Human/Computer Interactions", Masters Thesis, MIT, 1979

4 ) T B Sheridan, W L Verplank, "Human and Computer Control of Undersea Teleoperators", Man-Machine Systems Laboratory Report, MIT, 1979

5 ) L D Harmon, "The Sense of Touch Begins to Gather Momentum", Sensor Review, April 1981, 81-89

6 ) D D Grossman, R H Taylor, "Interactive Generation of Object Models with a Manipulator", IEEE Transactions on Systems, Man, and Cybernetics, Vol SMC-8, Sept 1978

7 ) W M Newman, R F Sproull, "Principles of Interactive Computer Graphics", McGraw-Hill, 2nd Ed, 1979

8 ) C T Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters", IEEE Trans Comput C-20, 1971, 68-86

9 ) G Gini, M Gini, "Object Description with a Manipulator", The Industrial Robot, March 1978

APPENDIX I

## COMPUTER PROGRAM DESCRIPTION

Interdependent tasks such as manipulator simulation, vector graphic display, and polyhedron construction where all combined in one Fortran 4-Plus program because it was most practical that they all work at the same time Raster graphic display was done on a seperate program as it did not have to run in real-time

### MAIN PROGRAM

The main program, TOUCH, handled manipulator simulation, touch sensing, and program initialization TOUCH was basicly a stripped down version of C Winey's ARM program [ 1 ] Only those parts that were required for manipulator simulation were saved because cycle time was critical The touch sensing capability was added and took care of locating points and touch vectors any time a touch sensor switch was found to be tripped Also some algorithms were added to improve absolute point coordinate calculation as described in Chapter 5

When running, the processor would simply loop through TOUCH continually refreshing the manipulator display and waiting for an outside command Control would be transfered to subroutine DISP in the event of a keyboard input or to subroutine CON if a touch sensor tripped when a polyhedron was being constructed Cycle time through TOUCH was about 20 milliseconds which was fast enough to simulate a smooth moving manipulator and give it a reasonably good reaction time to respond to touch inputs

Subroutine DISP responded to any keyboard inputs and took care of display managment It controlled view angle, set program parameters, and organized information output It was responsible for creating, deleting, and starting construction of polyhedra DISP was called every cycle of TOUCH when it was required that the display be dynamicly rotated or moved This increased cycle time to 26 milliseconds

Subroutine CON took care of adding new points to an existing polyhedron If no polyhedron existed, CON would do the process of initialization described in Chapter 6 CON decided which was the best facet to attach to and made sure that it did not violate any rules for a closed polyhedra After the point was attached CON did the job of deciding which facet pairs to check for smoothing

Subroutine FACE compared facet pairs and decided when they should be switched with compliment facets It determined the angles between neighboring facets and checked that new facets did not violate any rules for closed polyhedra FACE was be called by CON when checking facet pairs and could also be called by DISP when the operator wanted to check or change facets from the keyboard

Subroutine CTOUR drew evenly spaced contour lines across the existing polyhedron These contour lines were always drawn on the object coordinate z plane

Subroutine JROL performed rotations in screen coordinates for control by the joystick or trackball

The following list of subroutines took care of individual tasks that were often required by main subroutines

Subroutine PIERC compared relationships between a line and a triangle It determined if the line pierced through the triangle, if the triangle faced away from the base of the line, and if the line pointed away from the triangle It could also determine the distance along the direction of the line from the base of the line to the plane described by the triangle PIERC was used to determine if two facets were concave or convex, if a touch vector was at an angle greater than 90 degrees to a facet, or if a line segment stuck through a facet

CROSS - determined the normal vector of a plane described by three points in 3-space

ANGL - determined the angle difference between two vectors in 3-space

SEARCH - found the third point of a facet on an existing polyhedron if given the two other points in sequence for that facet

VECT - drew a line on the screen between two specified touch points

The following are the set of library subroutines that were used to control the vector graphics display processor

MGINIT -initialize the Megatek

MGSEND -send data in display buffer to Megatek

SETINT -set the light intensity for all lines drawn after it

DRWI3 -draw a 3-D line

MOVI3 -move to a new 3-D location without drawing line

PNTI3 -draw a point in 3-D

NPOINT -find last line number being used by Megatek

MODIFY -modify next command in Megatek with next call

LDPTRO -reset beginning of Megatek display and erase everything after it

LDTRN3 -send transformation coefficients for rotation, translation, and zoom for all lines drawn after it

## RASTER DISPLAY

The program DRW read 3-dimensional points and polyhedron data from a data file from DISP The points were preformatted on the vector graphic display DRW drew all the facets facing toward the screen on the Lexidata Furthermost facets were drawn first so that they would be erased if a closer facet was in front of them Shading was accomplished by relating a facet to the angle between a facet normal and a vector simulating a light source direction The light source direction could be changed with a trackball very quickly by changing the shading lookup table

The following are a list of subroutines used to read the trackball and control the Lexidata

TBALL - read trackball x and y velocities and the combined value of three switches

DSVEC - drew a line between two points and selected a shading lookup number

DSLLU - changed the shade of one lookup number

DSLWT - changed the shades of many lookup number according to an array

## APPENDIX B1 COMPUTER PROGRAM FOR DISPLAY OF MANIPULATOR AND SURFACE POINTS

```
        PROGRAM TOUCH
C INITIALIZE PROGRAM
        DIMENSION IPT(4),XX(4),XY(4),XZ(4),XT(4),YX(4),YY(4)
        DIMENSION YZ(4),ZX(4),ZY(4),ZZ(4),YT(4),ZT(4)
        DIMENSION SCL(7,2),IA(16),A(7)
        DIMENSION IOSB(2),IBUF(12),IPOT(10),MS(10),IPARAM(6)
        COMMON /DMABUF/ IDUM(2298),ADAT(51,3),BRP(36,3),
     1  ICON(90,2),IBRC(50,2),IFC(200,3),M(100,3)
        COMMON /FACT/IFMAX,NX(30),NA,IPS,NCON,NPOL,ICCN,
     1  IVECT,ISUP,IRX
        COMMON /IPTPS/ IANG(100,2),ICHECK,VEX
        COMMON /DISPL/ICM,XXD,XYD,XZD,XTD,YXD,YYD,YZD,
     1  YTD,ZXD,ZYD,ZZD,ISHAD,IARM,IWALL,IROLL,JSTICK,IDOTR
C INITIALIZE THE MEGATEK AND A/D
        CALL ANINIT
        CALL MGINIT
        CALL SETINT(13)
        CALL NPOINT(IREP)
C INITIALIZE THE KEYBOARD MONITER ROUTINE
        CALL GETADR(IPARAM(1),ICMD)
        IPARAM(2)=1
        IEXC="033
        LLL="114
        IAAA="101
C INITIALIZE VIEW AND MENU
100     IRX=-5
        CALL DISP
        IRX=0
        ICM="114
        CALL DISP
C SET LINK LENGTHS AND ORIGIN
        AY=55 625
        AZ=1600
        SZ=720
        ZOG=480
        YOG=960
        VEX=2
C READ SCALING FACTORS FOR A/D OUTPUT OF ANGLES
        OPEN(UNIT=4,NAME='SCALE DAT',TYPE='OLD')
        READ(4,*)((SCL(I,J),J=1,2),I=1,7)
        CLOSE(UNIT=4,DISPOSE='SAVE')
C READ POINT DATA FOR MANIPULATOR
        OPEN(UNIT=4,NAME='ARMSDT DAT',TYPE='OLD')
        DO 101 I=1,60
        READ(4,*,END=102)ADAT(I,1),ADAT(I,2),ADAT(I,3)
101     CONTINUE
102     CLOSE(UNIT=4,DISPOSE='SAVE')
C READ CONNECTIVITY DATA FOR MANIPULATOR
        OPEN(UNIT=4,NAME='ARMSCN DAT',TYPE='OLD')
        DO 103 I=1,100
        READ(4,*,END=104)ICON(I,1),ICON(I,2)
103     CONTINUE
104     CLOSE(UNIT=4,DISPOSE='SAVE')
C READ POINT DATA FOR TOUCH SENSOR
        OPEN(UNIT=4,NAME='BRSHDT DAT',TYPE ='OLD')
        DO 105 I=1,50
        READ(4,*,END=106)BRP(I,1),BRP(I,2),BRP(I,3)
105     CONTINUE
106     CLOSE(UNIT=4,DISPOSE='SAVE')
C READ CONNECTIVITY DATA FOR TOUCH SENSOR
        OPEN(UNIT=4,NAME='BRSHCN DAT',TYPE='OLD')
        DO 107 I=1,50
        READ(4,*,END=108)IBRC(I,1),IBRC(I,2)
        NBCON=I
107     CONTINUE
108     CLOSE(UNIT=4,DISPOSE='SAVE')
C INPUT ARM AND WALL LINES INTO MEGATEK
        IXXX=83+NBCON
        DO 128 I=1,IXXX
        INK=I-NBCON
        IF(I EQ 1)GOTO 341
        IF(INK EQ 36)GOTO 343
        IF(INK EQ 56)GOTO 342
        IF(INK EQ 72)GOTO 344
        GOTO 346
C INPUT TOUCH SENSOR
341     CALL SETINT(13)
        CALL NPOINT(IPT(2))
        IARM=IPT(2)-1
        GOTO 345
C INPUT SHOULDER
342     CALL NPOINT(IPT(3))
        GOTO 345
C INPUT FOREARM
343     CALL NPOINT(IPT(4))
        GOTO 345
C INPUT WALLS
344     CALL SETINT(13)
        CALL NPOINT(IPT(1))
        IWALL=IPT(1)-1
345     CALL LDTRN3(1 ,0 ,0 ,3000 ,0 ,1 ,0 ,0 )
346     IF(INK LE 0)GOTO 250
        MR=ICON(INK,1)
        MM=ICON(INK,2)
        IX1=40 *ADAT(MR,1)
        IX2=40 *ADAT(MM,1)
        IY1=40 *ADAT(MR,2)
        IY2=40 *ADAT(MM,2)
        IZ1=40 *ADAT(MR,3)
        IZ2=40 *ADAT(MM,3)
        GOTO 249
250     MR=IBRC(I,1)
```

```
        MM=IBRC(I,2)
        IX1=40 *BRP(MR,1)
        IX2=40 *BRP(MM,1)
        IY1=40 *BRP(MR,2)
        IY2=40 *BRP(MM,2)
        IZ1=40 *BRP(MR,3)
        IZ2=40 *BRP(MM,3)
        IF(IX1 EQ 880 OR IX2 EQ 880)GOTO 128
249     CALL MOVI3(IX1,IY1,IZ1)
        CALL DRWI3(IX2,IY2,IZ2)
C SEND TO DISPLAY
        CALL MGSEND
128     CONTINUE
129     CONTINUE
C SET DISPLAY AFTER MANIPULATOR
        CALL SETINT(13)
        CALL NPOINT(NCON)
        CALL MGSEND
C READ ARM POSITION FROM A/D CONVERTER AND CONVERT TO
VOLTAGE
C READ TOUCH SENSOR SWITCHES
112     CALL AINSQ(16,22,IA)
        CALL DIN(20,ISP)
135     DO 113 I=1,7
        A(I)=FLOAT(IA(I))/3276 2
113     CONTINUE
C SCALE A/D OUTPUT, FILTER, AND CALCULATE SINES _COSINES
914     THZ=SCL(1,1)*A(5)+SCL(1,2)
        THX=SCL(2,1)*A(7)+SCL(2,2)
        THYZ=SCL(3,1)*A(6)+SCL(3,2)
        THY=THYZ-THZ
        THA=SCL(4,1)*A(2)+SCL(4,2)
        THR=SCL(5,1)*A(3)+SCL(5,2)
        THL=SCL(6,1)*A(4)+SCL(6,2)
        S1=SIN(THZ)
        S2=SIN(THX)
        S4=SIN(THA)
        C1=COS(THZ)
        C2=COS(THX)
        C4=COS(THA)
C PREFORM PUSHROD CALCULATION
        ZP1=-2 25*S2-4 5*C2*CP3
        XGXP=3 1-2 25*C2+4 5*S2*CP3
        ZGZP=-4 5*COS(THY)+2 25*S2+4 5*C2*CP3
        YP=4 5*SIN(THY)+SQRT(324 72-XGXP*XGXP-ZGZP*ZGZP)
        SP3=(YP-18 )/4 5
        THY1=ASIN(SP3)
        CP3=COS(THY1)
C ROTATE JOINT 90 DEGREES
        S3=-CP3
        C3=SP3
C PREFORM DIFFERENTIAL CALCULATION
```

```
        S5=SIN((THR+THL)/2)
        S6=SIN((THL-THR)/1 65)
        C5=COS((THR+THL)/2)
        C6=COS((THL-THR)/1 65)
C DO LINK TRANSFORMATIONS
C SHOULDER TRANSFORMS
        XX(3)=C2
        XY(3)=-S2
        YX(3)=C1*S2
        YY(3)=C1*C2
        YZ(3)=-S1
        YT(3)=YOG
        ZX(3)=S2*S1
        ZY(3)=S1*C2
        ZZ(3)=C1
        ZT(3)=ZOG
C FOREARM TRANSFORMATIONS
        XX(4)=C2*C4-S2*C3*S4
        XY(4)=-C2*S4-S2*C3*C4
        XZ(4)=S2*S3
        YX(4)=C1*S2*C4+C1*C2*C3*S4-S1*S3*S4
        YY(4)=-C1*S2*S4+C1*C2*C3*C4-S1*S3*C4
        YZ(4)=-C1*C2*S3-S1*C3
        YT(4)=S1*SZ+YOG
        ZX(4)=S1*S2*C4+S1*C2*C3*S4+C1*S3*S4
        ZY(4)=-S1*S2*S4+S1*C2*C3*C4+C1*S3*C4
        ZZ(4)=-S1*C2*S3+C1*C3
        ZT(4)=-C1*SZ+ZOG
C HAND TRANSFORMATIONS
        XX(2)=XX(4)*C6+XY(4)*C5*S6+XZ(4)*S5*S6
        XY(2)=-XX(4)*S6+XY(4)*C5*C6+XZ(4)*S5*C6
        XZ(2)=-XY(4)*S5+XZ(4)*C5
        XT(2)=-XZ(4)*AZ+XY(4)*AY
        YX(2)=YX(4)*C6+YY(4)*C5*S6+YZ(4)*S5*S6
        YY(2)=-YX(4)*S6+YY(4)*C5*C6+YZ(4)*S5*C6
        YZ(2)=-YY(4)*S5+YZ(4)*C5
        YT(2)=-YZ(4)*AZ+YT(4)+YY(4)*AY
        ZX(2)=ZX(4)*C6+ZY(4)*C5*S6+ZZ(4)*S5*S6
        ZY(2)=-ZX(4)*S6+ZY(4)*C5*C6+ZZ(4)*S5*C6
        ZZ(2)=-ZY(4)*S5+ZZ(4)*C5
        ZT(2)=-ZZ(4)*AZ+ZT(4)+ZY(4)*AY
C
C DO DISPLAY TRANSFORM AND SEND TO DISPLAY
        DO 371 I=2,4
60      XX1=XX(I)*XXD+YX(I)*XYD+ZX(I)*XZD
        XY1=XY(I)*XXD+YY(I)*XYD+ZY(I)*XZD
        XZ1=XZ(I)*XXD+YZ(I)*XYD+ZZ(I)*XZD
        XT1=XT(I)*XXD+YT(I)*XYD+ZT(I)*XZD+XTD
        YX1=XX(I)*YXD+YX(I)*YYD+ZX(I)*YZD
        YY1=XY(I)*YXD+YY(I)*YYD+ZY(I)*YZD
        YZ1=XZ(I)*YXD+YZ(I)*YYD+ZZ(I)*YZD
        YT1=XT(I)*YXD+YT(I)*YYD+ZT(I)*YZD+YTD
        CALL MODIFY(IPT(I))
        CALL LDTRN3(XX1,XY1,XZ1,XT1,YX1,YY1,YZ1,YT1)
371     CONTINUE
C DO WALL TRANSFORMATION
        CALL MODIFY(IPT(1))
        CALL LDTRN3(XXD,XYD,XZD,XTD,YXD,YYD,YZD,YTD)
C
C EXAMINE TOUCH SENSOR SWITCHES
        IF(ISP EQ "177777)GOTO 372
62      DO 372 I=1,10
        J=2**(I-1)
C PUT WAIT LIMIT ON SWITCHES OF 4 CYCLES
        IF((ISP AND J) EQ 0)IPOT(I)=1
        IF(IPOT(I) EQ 0 AND MS(I) GT 0)MS(I)=MS(I)-1
        IF(IPOT(I) EQ 0 OR MS(I) NE 0)GOTO 373
        MS(I)=4
C RING TERMINAL BELL IF QIO IS OFF
        NBEEP="007
        IF(IRX EQ -4)WRITE(5,765)NBEEP
765     FORMAT(' ',1A1)
C INCREMENT POINT COUNTER
        IPS=IPS+1
C FIND SENSOR CENTERLINE
        XP=BRP(I,1)*40
        YP=BRP(I,2)*40
        ZP=BRP(I,3)*40
        XB=BRP(I+10,1)*40
        YB=BRP(I+10,2)*40
        ZB=BRP(I+10,3)*40
C DO TRANSFORM TO GET TOUCH POINT AND VECTOR IN
C MANIPULATOR COORDINATES
        MMMX=XP*XX(2)+YP*XY(2)+ZP*XZ(2)+XT(2)
        MMMY=XP*YX(2)+YP*YY(2)+ZP*YZ(2)+YT(2)
        MMMZ=XP*ZX(2)+YP*ZY(2)+ZP*ZZ(2)+ZT(2)
        XBT=XB*XX(2)+YB*XY(2)+ZB*XZ(2)+XT(2)
        YBT=XB*YX(2)+YB*YY(2)+ZB*YZ(2)+YT(2)
        ZBT=XB*ZX(2)+YB*ZY(2)+ZB*ZZ(2)+ZT(2)
        IF(IDOTR EQ 1)WRITE(2,*)MMMX,MMMY,MMMZ
        IF(IPS GT 100)GOTO 374
C POINT COORDINATES
        M(IPS,1)=MMMX
        M(IPS,2)=MMMY
        M(IPS,3)=MMMZ
C TOUCH VECTOR ANGLES *10000 (TO STORE AS INTEGERS)
        VL=SQRT((M(IPS,1)-XBT)**2+(M(IPS,2)-YBT)**2)
        IANG(IPS,1)=ATAN2(YBT-M(IPS,2),XBT-M(IPS,1))*10000
        IANG(IPS,2)=ATAN2(ZBT-M(IPS,3),VL)*10000
C DO CONNECTION IF ENABLED
        IF(ICCN NE 1)GOTO 374
        TIME= SECNDS(0 0)
        CALL CON
        TIME=SECNDS(TIME)
        IF(IRX EQ -4)TYPE *,IPS,TIME
C RING BELL TO INDICATE COMPLETION
        IF(IRX EQ -4)WRITE(5,765)NBEEP
        GOTO 373
C DRAW POINT ON SCREEN
374     CALL PNTI3(MMMX,MMMY,MMMZ)
        CALL MGSEND
373     IPOT(I)=0
372     CONTINUE
C ENABLE QIO IF ARM IS TWISTED
        IF(IRX EQ -4 AND THA GT 3 0)IRX=0
        IF(IRX EQ -4)GOTO 112
C CHANGE DISPLAY TRANSFORMATIONS IF VIEW IS CHANGING
        IF(IROLL EQ 1)CALL DISP
        IF(IVECT EQ 2)GOTO 112
C READ KEYBOARD
        IF(IFF NE 1)CALL QIO("10400,5,3,,IOSB,IPARAM,IDS)
        IFF=1
        CALL READEF(3,IUU)
        IF(IUU NE 2)GOTO 112
        ICM=ICMD
        WRITE(5,999)IEXC,LLL,IEXC,IAAA
999     FORMAT('+',4A)
        IFF=0
        CALL DISP
        ICM=0
C LOOK AT DISPLAY FLAG
382     IF(IRX EQ -1)CALL LDPTRO(NCON)
        IF(IRX EQ -1)CALL MGSEND
        IF(IRX EQ -2)GOTO 100
        GOTO 112
        END
```

## APPENDIX C SUBROUTINES TO CONSTRUCT A POLYHEDRON FROM SURFACE POINT DATA

```
        SUBROUTINE CON
        COMMON /DMABUF/ IDUM(3060),NF(20),ID(20),
     1  IFC(200,3),M(100,3)
        COMMON /IPTPS/IANG(100,2)
        COMMON /FACT/IFMAX,NX(30),NA,IPS,NCON,NPOL,ICCN,
     1  IVECT,ISUP,IRX
C INITIALIZE VARIABLES
        NDIST=10  ' MAXIMUM DISTANCE TO FACET CENTROID
        NMAX=5    ' NUMBER OF FACETS FOR COMPLETE CHECKS
357     IPRC1=0
        PD1=0
        DL1=0
        IB=0
        ITRY=0
        NFMX=NMAX
        PI=3 1415927
        IF (IPS GT 3)GOTO 3  'IPS=CURRENT NUMBER OF POINTS
        IF (IPS GT 1)GOTO 1
C DRAW DOT FOR FIRST POINT
        CALL PNTI3(M(1,1),M(1,2),M(1,3))
        GOTO 5
1       IF (IPS GT 2)GOTO 2
C DRAW LINE BETWEEN FIRST 2 POINTS
        CALL VECT(1,2)
        GOTO 5
C CONSTRUCT INITIALIZING FACETS ON FIRST 3 POINTS
2       CALL VECT(2,3)
        CALL VECT(3,1)
        IFC(1,1)=3   ' LOAD FIRST 2 FACETS FIRST 3 POINTS
        IFC(1,2)=2
        IFC(1,3)=1
        IFC(2,1)=1
        IFC(2,2)=2
        IFC(2,3)=3
        IFMAX=2  ' NUMBER OF FACETS ON EXISTING POLYHEDRON
5       CALL MGSEND
        RETURN
3       CONTINUE
        DO 320 I=1,20
        ID(I)=40 *NDIST
320     NF(I)=0
C FIND DISTANCE FROM POINT TO CENTROID OF ALL FACETS
        DO 321 I=1,IFMAX
        IFC1=IFC(I,1)
        IFC2=IFC(I,2)
        IFC3=IFC(I,3)
        FXA=(M(IFC1,1)+M(IFC2,1)+M(IFC3,1))/3 -M(IPS,1)
        FYA=(M(IFC1,2)+M(IFC2,2)+M(IFC3,2))/3 -M(IPS,2)
        FZA=(M(IFC1,3)-M(IFC2,3)+M(IFC3,3))/3 -M(IPS,3)
        LD=SQRT(FXA*FXA+FYA*FYA+FZA*FZA)
```

```
C ADD FACET TO LIST OF NEAREST FACETS IF CLOSE ENOUGH
        IF(LD GE ID(NMAX))GOTO 321
        ID(NMAX)=LD               ' DISTANCE TO NEAR FACET
        NF(NMAX)=I                ' NUMBER OF NEAR FACET
        DO 322 J=1,NMAX-1
        J1=NMAX-J
        IF(LD GE ID(J1))GOTO 321
        ID(J1+1)=ID(J1)
        NF(J1+1)=NF(J1)
        ID(J1)=LD
        NF(J1)=I
322     CONTINUE
321     CONTINUE
C BEGIN SEARCH FOR BEST FACET
323     DO 100 IC=1,NFMX
        I=IC
        IF(ITRY EQ 1)GOTO 324
        IF(NF(IC) LT 1 OR NF(IC) GT IFMAX)GOTO 100
        I=NF(IC)
C
324     IFC1=IFC(I,1)
        IFC2=IFC(I,2)
        IFC3=IFC(I,3)
        ICW=0    'SET FLAG TO SIGNIFY ORDINARY FACET CHECK
C
C FIND IF TOUCH VECTOR PIERCES FACET (IPEIRC)
C FIND WHICH WAY FACET IS FACING POINT (LOUTF)
C FIND DISTANCE BETWEEN POINT AND FACET PLANE ALONG
C TOUCH VECTOR (PDIST)
        CALL PIERC(IPS,IFC1,IFC2,IFC3,IPIERC,LOUTF,
     1  PDIST,ICW)
C
C DECIDE IF FACET IS BEST SO FAR
C
C CHECK IF DIRECTION VECTOR POINTS FOWARD THRU FACET
        IF (IPIERC LE 0)GOTO 40
C
C REJECT POINT IF FACET FACES TOWARD TOUCH POINT
        IF (LOUTF GE 0)GOTO 38
        IB=0
        TYPE *,'NEGATIVE PIERCING FACET'
        GOTO 52
C
C COMPARE TO BEST FACET
38      IF (IPRC1 LE 0 OR PDIST LT PD1)GOTO 60
        GOTO 50
C
C REJECT ALL OTHER FACETS IF BEST IS PIERCED POSITVE
40      IF (INTPNT EQ 1)GOTO 50
C
C CHECK CASE WHERE DIRECTION VECTOR POINTS AWAY THRU FACET
        IF (IPIERC EQ 0)GOTO 45
        IF (PDIST GT 0) GOTO 50
        IF (LOUTF EQ 1)GOTO 50
        IF (IPRC1 EQ 0)GOTO 60
        IF (PDIST LT PD1)GOTO 50
        GOTO 60   ' GO FOR FURTHER TESTS
C CHECK CASE WHERE DIRECTION VECTOR DOESNT PIERCE FACET
45      IF (LOUTF EQ 1)GOTO 50
        IF (IPRC1 NE 0)GOTO 50
C
C FIND DISTANCE TO CENTROID OF FACET
        FXA=(M(IFC1,1)+M(IFC2,1)+M(IFC3,1))/3 -M(IPS,1)
        FYA=(M(IFC1,2)+M(IFC2,2)+M(IFC3,2))/3 -M(IPS,2)
        FZA=(M(IFC1,3)+M(IFC2,3)+M(IFC3,3))/3 -M(IPS,3)
        DL=SQRT(FXA*FXA+FYA*FYA+FZA*FZA)
        IF (DL1 EQ 0)GOTO 60
        IF (DL1 LE DL)GOTO 50 'REJECT IF NOT NEAREST SO FAR
C
C CHECK PIERCING OF OLD FACETS BY NEW LINES
60      DO 310 J=1,IFMAX
C SET FLAG TO CHECK PIERCING OF LINE SEGMENT THROUGH FACET
        ICWF=4
        CALL PIERC(IPS,IFC1,IFC(J,1),IFC(J,2),
     1  IFC(J,3),LOT,PDD,ICWF)
        IF(ICWF EQ 6)GOTO 50
        CALL PIERC(IPS,IFC2,IFC(J,1),IFC(J,2),
     1  IFC(J,3),LOT,PDD,ICWF)
        IF(ICWF EQ 6)GOTO 50
        CALL PIERC(IPS,IFC3,IFC(J,1),IFC(J,2),
     1  IFC(J,3),LOT,PDD,ICWF)
310     IF(ICWF EQ 6)GOTO 50
C
C CHECK PIERCING OF NEW FACETS BY ALL OTHER TOUCH VECTORS
        DO 51 J=1,IPS
        ICWF=0
        IPP=0
        CALL PIERC(J,IPS,IFC2,IFC3,IPP,LOT,PDD,ICWF)
        IF(IPP GT 0)GOTO 50
        CALL PIERC(J,IPS,IFC3,IFC1,IPP,LOT,PDD,ICWF)
        IF(IPP GT 0)GOTO 50
        CALL PIERC(J,IPS,IFC1,IFC2,IPP,LOT,PDD,ICWF)
        IF(IPP GT 0)GOTO 50
51      CONTINUE
C SAVE POINT AS BEST SO FAR AND SAVE ALL ITS ATTRIBUTES
        IF(IPIERC EQ 0)DL1=DL
        PD1=PDIST
        IPRC1=IPIERC
        IB=I                 ' NUMBER OF BEST FACET
50      CONTINUE
100     CONTINUE
C RETURN IF NO GOOD FACET IS FOUND
        IF(IB EQ 0 AND ITRY EQ 0)GOTO 326
52      IF(IB NE   GOTO 10
      IPS=IPS-1
      IF(IRX EQ -4)WRITE(5,234)
234   FORMAT('    *****REJECT POINT*****')
      RETURN
326   ITRY=1  ' MAKE SECOND TRY BY CHECKING ALL FACETS
      NFMX=IFMAX
      GOTO 323
C
C DRAW LINES FROM NEW POINT TO CHOSEN FACET
10    DO 55 I=1,3
55    CALL VECT(IPS,IFC(IB,I))
      CALL MGSEND
C
C GET RID OF OLD FACET AND ADD 3 NEW ONES
      IFC(IFMAX+1,1)=IPS
      IFC(IFMAX+2,1)=IPS
      IFC(IFMAX+1,2)=IFC(IB,2)
      IFC(IFMAX+2,2)=IFC(IB,3)
      IFC(IFMAX+1,3)=IFC(IB,3)
      IFC(IFMAX+2,3)=IFC(IB,1)
      IFC(IB,3)=IPS
      IFMAX=IFMAX+2
C
C  SELECT FACET PAIRS FOR CHECKING SMOOTHNESS
C
      DO 181 I=1,30
181   NX(I)=0              'LIST OF POINTS TO CHECK AROUND
      NX(1)=IPS
      NX(2)=IFC(IB,2)
      NX(3)=IFC(IB,3)
      NX(4)=IFC(IFMAX,2)
      NA=4
      NEND=0
140   IF(NA GT 30)NA=30
      NX1=NX(NA)
      NA=NA-1
      K1=0
      DO 182 I=1,30
182   IF(NX1 EQ NX(I))K1=K1+1
      IF(K1 GT 3)GOTO 143
      DO 141 I=1,IFMAX
      DO 142 J=1,3
      IF(NX1 NE IFC(I,J))GOTO 142
      K1=NX1
      K2=IFC(I,1+MOD(J,3))
      CALL FACE(K1,K2)
      NEND=NEND+1
      IF(NEND GT 50)GOTO 144
142   CONTINUE
141   CONTINUE
143   IF(NA GT 0)GOTO 140
144   RETURN
```

```
        END
        SUBROUTINE FACE(K1,K2)
        DIMENSION IA(4)
        COMMON /DMABUF/ IDUM(3100),IFC(200,3),M(100,3)
        COMMON /FACT/ IFMAX,NX(30),NA,IPS,NCON,NPOL
        COMMON /IPTPS/ IANG(100,2),ICHECK,VEX,IFCC
C FIND ALTERNATE SET OF POINTS M1,M2
C       ISTICK=0                        '    INITIAL FACETS
        IFCC=0                          '
        IF(K1 EQ K2)RETURN              '           M1
        IF(K1 EQ 0 OR K2 EQ 0)RETURN    '            *
556     IF(K1 LT K2)GOTO 5              '           / \
        IFACE1=K1                       '          /   \
        K1=K2                           '         /     \
        K2=IFACE1                       '    K1  *-------* K2
5       IFACE1=0                        '         \     /
        CALL SEARCH(K1,K2,M1,IFACE1)    '          \   /
        CALL SEARCH(K2,K1,M2,IFACE2)    '           \ /
        IF(M1 EQ 0 OR M2 EQ 0)RETURN    '            *
70      IF(ICHECK EQ 1)WRITE(5,*)K1,K2,M1,M2'       M2
        IF(M1 EQ M2)RETURN              '
        CALL SEARCH(M1,M2,J,I)          '    OUT OF THE PAGE
        IF(J NE 0)RETURN                '    IS OUTSIDE THE
C IF ICHECK=2 FORCE FACET CHANGE        '    POLYHEDRON
        IF(ICHECK EQ 2)GOTO 160
C
C FIND ATRIBUTES OF COMPLIMENT FACETS
90      CALL PIERC(K1,M1,M2,K2,IPRC3,LOUTF2,PDIST,ICW3)
        CALL PIERC(K2,M2,M1,K1,IPRC4,LOUTF3,PDIST,ICW4)
C IF ICHECK=1 THIS SUBROUTINE ONLY PRINTS THE FACET DATA
        IF(ICHECK NE 1)GOTO 689
        TYPE *,'POINT  LOUTF  ICW  IPIERC'
        WRITE(5,*)K1,LOUTF2,ICW3,IPRC3
        WRITE(5,*)K2,LOUTF3,ICW4,IPRC4
689     IF(IPRC3 LE 0 AND IPRC4 LE 0)GOTO 155
        ISTICK=1
        IF(LOUTF2 GT 0)RETURN
C
C
C CHECK IF TOUCH VECTOR AND FACET NORMAL ARE OVER
C 90 DEGREES APART
155     IF(ICW3 NE 1 AND ICW4 NE 1)GOTO 159
        IF(LOUTF2 EQ -1)RETURN
        GOTO 157
159     A1=0
        A2=0
C FIND THE 4 PERIPHERY POINTS
        CALL SEARCH(K1,M1,M11,I)
        CALL SEARCH(M1,K2,M12,I)
        CALL SEARCH(K2,M2,M22,I)
        CALL SEARCH(M2,K1,M21,I)
C FIND ALL THE SURFACE NORMALS
```

```
      CALL CROSS(K1,K2,M1,1)
      CALL CROSS(K2,K1,M2,2)
      CALL CROSS(M1,M2,K2,3)
      CALL CROSS(M2,M1,K1,4)
      CALL CROSS(K1,M1,M11,5)
      CALL CROSS(M1,K2,M12,6)
      CALL CROSS(K2,M2,M22,7)
      CALL CROSS(M2,K1,M21,8)
C FIND ANGLE BETWEEN ADJACENT FACETS AND ADD TOGETHER
      CALL ANGL(6,1,A1)
      CALL ANGL(7,2,A1)
      CALL ANGL(8,2,A1)
      CALL ANGL(5,1,A1)
      CALL ANGL(5,4,A2)
      CALL ANGL(6,3,A2)
      CALL ANGL(7,3,A2)
      CALL ANGL(8,4,A2)
      CALL ANGL(1,2,A1)
      CALL ANGL(3,4,A2)
420   IF(ICHECK NE 1)GOTO 421
      TYPE *,'ORIGINAL ANGLE TOTAL=',A1
      TYPE *,'COMPLIMENT ANGLE TOTAL=',A2
421   IF(A1 LE A2)RETURN
C
C CHECK IF NEW LINE PIERCES ANY FACETS
157   CALL PIERC(M1,K2,K1,M2,IPRC3,LOUTF1,PDIST,ICW3)
      CALL PIERC(M2,K1,K2,M1,IPRC4,LOUTF1,PDIST,ICW4)
      IF(ICHECK NE 1)GOTO 156
      TYPE *,'POINT  LOUTF  ICW  IPIERC'
      WRITE(5,*)M1,LOUTF1,ICW3,IPRC3
      WRITE(5,*)M2,LOUTF1,ICW4,IPRC4
      RETURN
156   IF(IPRC3 LE 0 AND IPRC4 LE 0)GOTO 158
      IF(LOUTF2 EQ 1)RETURN
158   IF((ICW3 EQ 1 OR ICW4 EQ 1) AND LOUTF2 EQ -1)RETURN
      DO 160 I=1,IFMAX
      ICWF=4
      CALL PIERC(M1,M2,IFC(I,1),IFC(I,2),IFC(I,3),
     1 LOUTF1,PDIST,ICWF)
      IF(ICWF EQ 6)RETURN
160   CONTINUE
C
C CHECK IF ANY LINES PIERCE NEW FACETS
      DO 568 I=1,IFMAX
      DO 569 J=1,3
      KK1=IFC(I,J)
      KK2=IFC(I,1+MOD(J,3))
      IF(KK1 GE KK2)GOTO 569
      ICWF=4
      CALL PIERC(KK1,KK2,K1,M2,M1,LOUTF1,PDIST,ICWF)
      IF(ICWF EQ 6)RETURN
      CALL PIERC(KK1,KK2,K2,M1,M2,LOUTF1,PDIST,ICWF)
```

```
        IF(ICWF EQ 6)RETURN
569     CONTINUE
568     CONTINUE
C
C CHECK IF ANY TOUCH VECTORS PIERCE NEW FACETS
        IF(ISTICK EQ 1)GOTO 570
        ICWF=0
        DO 570 I=1,IPS
        IF(I EQ K1 OR I EQ K2 OR I EQ M1 OR I EQ M2)GOTO 570
        CALL PIERC(I,K1,M2,M1,IPRC1,LOUTF1,PDIST,ICWF)
        IF(IPRC1 EQ 0)GOTO 571
        IF(PDIST GT 0)RETURN
571     CALL PIERC(I,K2,M1,M2,IPRC1,LOUTF1,PDIST,ICWF)
        IF(IPRC1 EQ 0)GOTO 570
        IF(PDIST GT 0)RETURN
570     CONTINUE
        IFCC=1
C
C RECORD NEW FACETS
        IFC(IFACE1,1)=K1             !      CONVERTED FACETS
        IFC(IFACE1,2)=M2             !
        IFC(IFACE1,3)=M1             !             M1
        IFC(IFACE2,1)=K2             !              *
        IFC(IFACE2,2)=M1             !             /I\
        IFC(IFACE2,3)=M2             !            / I \
C                                    !           /  I  \
C  RECORD NEW LINES TO BE CHECKED    !        K1 *   I   * K2
        NA=NA+4                      !           \  I  /
        IF(NA GT 30)GOTO 300         !            \ I /
        NX(30)=M1                    !             \I/
        NX(29)=M2                    !              *
        NX(28)=K1                    !             M2
        NX(27)=K2
        DO 161 J=1,4
        KK1=NX(30)
        DO 161 I=1,29
        KK2=NX(I)
        NX(I)=KK1
        KK1=NX(I+1)
        NX(I+1)=KK2
161     CONTINUE
C
C DRAW NEW POLYHEDROM
300     CALL LDPTRO(NCON)
        DO 310 I=1,IFMAX
        DO 309 J=1,3
        ILM1=IFC(I,J)
        ILM2=IFC(I,1+MOD(J,3))
        IF(ILM1 GT ILM2)GOTO 308
        CALL VECT(ILM1,ILM2)
308     CONTINUE
309     CONTINUE
```

```
310     CONTINUE
        CALL MGSEND
        RETURN
        END
C              SUBROUTINE  PIERC
C THIS SUBROUTINE CALCULATES ATTRIBUTES BETWEEN A VECTOR AND
C A TRIANGLE   IN TOUCH VECTOR MODE KP IS THE TOUCH
C POINT NUMBER AND K1-K2-K3 IS INPUT CORNER POINTS OF THE
C TRIANGLE  IN THIS MODE ICW WILL BE OUTPUT AS 1 IF THE
C TRIANGLE KP-K1-K2 HAS A SURFACE NORMAL MORE THAN 90
C DEGREES FROM THE TOUCH VECTOR
C IN LINE SEGMENT MODE, (ICW=4), KP IS INPUT AS THE PRIMARY
C ENDPOINT OF THE LINE SEGMENT AND K1 IS THE SECONDARY
C ENDPOINT AND K2-K3-K4 IS THE TRIANGLE
C
C AS OUTPUT, K4=1 IF THE VECTOR PIERCES THE FACET ON THE
C POSITIVE SIDE OF THE FACET AND K4=-1 IF IT PIERCES FROM
C THE NEGATIVE SIDE  K4=0 IF NEITHER IS TRUE  LOUTF=1 IF THE
C FACET FACES AWAY FROM THE PRIMARY POINT OR TOUCH POINT AND
C LOUTF=-1 IF IT FACES TOWARD THE POINT  PDIST IS THE
C DISTANCE FROM THE TOUCH POINT OR PRIMARY POINT TO THE
C FACET SURFACE ALONG THE VECTOR
C
        SUBROUTINE PIERC(KP,K1,K2,K3,K4,LOUTF,PDIST,ICW)
        DIMENSION RX(3),RY(3),RZ(3),PX(3),PY(3),PZ(3),KA(3)
        COMMON /DMABUF/IDUM(3700),M(100,3)
        COMMON /FACT/ IFMAX,NX(30),NA,IPS,NCON,NPOL
        COMMON /IPTPS/ IANG(100,2)
        DATA PI/3 1415927
        IF(ICW NE 4)GOTO 40
C LINE SEGMENT MODE
        KA(1)=K2
        KA(2)=K3
        KA(3)=K4
C IGNORE COMPARISIONS IF ANY POINTS ARE THE SAME
        DO 41 IIS=1,3
        IF(KA(IIS) EQ K1)RETURN
41      IF(KA(IIS) EQ KP)RETURN
C FIND ANGLES SIMILAR TO TOUCH VECTORS
        RDX=M(K1,1)-M(KP,1)
        RDY=M(K1,2)-M(KP,2)
        RDZ=M(K1,3)-M(KP,3)
        PGAP=SQRT(RDX*RDX+RDY*RDY+RDZ*RDZ)
        PG=SQRT(RDX*RDX+RDY*RDY)
        IF(PG NE 0)GOTO 42
        CTHET=1
        STHET=0
        GOTO 43
42      CTHET=RDX/PG
        STHET=RDY/PG
43      IF(PGAP NE 0)GOTO 44
        CPHI=1
        SPHI=0
        GOTO 50
44      SPHI=RDZ/PGAP
        CPHI=PG/PGAP
        GOTO 50
C ORDINARY MODE
C KP IS POINT WITH TOUCH VECTOR
C K1,K2,K3 DESCRIBE THE FACET
40      IF(KP EQ K1 OR KP EQ K2 OR KP EQ K3)RETURN
        CTHET=COS(FLOAT(IANG(KP,1))* 0001)
        STHET=SIN(FLOAT(IANG(KP,1))* 0001)
        CPHI=COS(FLOAT(IANG(KP,2))* 0001)
        SPHI=SIN(FLOAT(IANG(KP,2))* 0001)
        KA(1)=K1
        KA(2)=K2
        KA(3)=K3
50      DO 30 J=1,3
        PX(J)=M(KA(J),1)
        PY(J)=M(KA(J),2)
        PZ(J)=M(KA(J),3)
C GET POINTS IN COORDINATES OF KP POINT AND
C TRANSFORM COORDINATES SUCH THAT THE VECTOR LAYS ON THE
C X AXIS
        RDX=PX(J)-M(KP,1)
        RDY=PY(J)-M(KP,2)
        RDZ=PZ(J)-M(KP,3)
        RX(J)=(CTHET*RDX+STHET*RDY)*CPHI+SPHI*RDZ
        RY(J)=-STHET*RDX+CTHET*RDY
        RZ(J)=-(CTHET*RDX+STHET*RDY)*SPHI+CPHI*RDZ
30      CONTINUE
C
C       CHECK PIERCING
C IF THE VECTOR PIERCES THE TRIANGLE, THE CORNER POINTS
C WILL SURROUND THE X AXIS
        IPIERC=0
        T1=ATAN2(RY(1),RZ(1))
        T2=ATAN2(RY(2),RZ(2))-T1
        T3=ATAN2(RY(3),RZ(3))-T1
        IF(T2 LT 0)T2=T2+2*PI
        IF(T3 LT 0)T3=T3+2*PI
C
C CHECK IF TOUCH VECTOR IS GREATER THAN 90 DEG FROM NORMAL
        IF(ICW EQ 4)GOTO 55
C IF THE TWO OTHER POINTS GO SEQUENTIALLY CLOCKWISE WHEN
C LOOKING DOWN THE TOUCH VECTOR, THEN THE FACET IS MORE
C THAN 90 DEGREES AWAY
        ICW=0
        IF(T2 LT PI)ICW=1     ' GREATER THAN 90 DEG
C
55      IF(T2 GT PI)GOTO 32
        IF(T3 GT PI AND T3 LT T2+PI)IPIERC=1
        GOTO 36
```

```
32      IF(T3 LT PI AND T3 GT T2-PI)IPIERC=1
36      CONTINUE
C
C CHECK IF FACET POINTS OUT OR IN
C    FIND CROSS PRODUCT OF FACET
        DO 408 I=2,3
        RX(I)=RX(I)-RX(1)
        RY(I)=RY(I)-RY(1)
408     RZ(I)=RZ(I)-RZ(1)
        QX=RY(2)*RZ(3)-RZ(2)*RY(3)
        QY=RZ(2)*RX(3)-RX(2)*RZ(3)
        QZ=RX(2)*RY(3)-RY(2)*RX(3)
C   D IS POSITIVE IF FACET POINTS AWAY
        D=(RX(1)*QX+RY(1)*QY+RZ(1)*QZ)
        PDIST=100000
        IF(QX NE 0)PDIST=D/QX
C LOUTF=1 IF FACET FACES AWAY FROM POINT AND -1 OTHERWISE
        LOUTF=-1
        IF(D LT 0)LOUTF=1
C PDIST IS DISTANCE ALONG TOUCH VECTOR TO FACET SURFACE
        IF(PDIST LT 0)IPIERC=-IPIERC
        IF(ICW NE 4)K4=IPIERC
        IF(ICW NE 4 OR IPIERC EQ 0)RETURN
C INDICATE THAT SEGMENT PIERCES FACET
        IF(PDIST GT 0 AND PDIST LT PGAP)ICW=6
        RETURN
        END
        SUBROUTINE CROSS(M1,M2,M3,I)
C THIS SUBROUTINE FINDS THE SURFACE NORMAL OF A FACET
        COMMON /DMABUF/IDUM(3700),M(100,3)
        COMMON /TVEC/ TX(8),TY(8),TZ(8)
        A1=M(M2,1)-M(M1,1)
        A2=M(M2,2)-M(M1,2)
        A3=M(M2,3)-M(M1,3)
        B1=M(M3,1)-M(M1,1)
        B2=M(M3,2)-M(M1,2)
        B3=M(M3,3)-M(M1,3)
        TX(I)=A2*B3-A3*B2
        TY(I)=A3*B1-A1*B3
        TZ(I)=A1*B2-A2*B1
        RETURN
        END
C
        SUBROUTINE ANGL(I,J,A)
C THIS SUBROUTINE FINDS THE ANGLE BETWEEN TWO VECTORS
        COMMON /TVEC/TX(8),TY(8),TZ(8)
        R=TX(I)*TX(J)+TY(I)*TY(J)+TZ(I)*TZ(J)
        S1=SQRT((TX(I)*TX(I)+TY(I)*TY(I)+TZ(I)*TZ(I)))
        S2=SQRT((TX(J)*TX(J)+TY(J)*TY(J)+TZ(J)*TZ(J)))
        B=R/(S1*S2)
        IF(ABS(B) GT 1)TYPE *,'ERROR IN ANGL'
        B=ABS(ACOS(B))
```

```
      A=A+B
      RETURN
      END
C
      SUBROUTINE SEARCH(N1,N2,N3,N4)
C THIS SUBROUTINE FINDS THE 3RD POINT OF A FACET GIVEN THE
C OTHER TWO IN SEQUENCE  IT ALSO RETURNS THE FACET NUMBER
      COMMON /DMABUF/ IDUM(3100),IFC(200,3)
      COMMON /FACT/ IFMAX,NX(30),NA,IPS
      N3=0
      DO 10 I=1,IFMAX
      J=0
      IF(N1 EQ IFC(I,1))J=1
      IF(N1 EQ IFC(I,2))J=2
      IF(N1 EQ IFC(I,3))J=3
      IF(J EQ 0)GOTO 10
      IF(N2 NE IFC(I,1+MOD(J,3)))GOTO 10
      N3=IFC(I,1+MOD(J+1,3))
      N4=I
      RETURN
10    CONTINUE
      RETURN
      END
C
      SUBROUTINE VECT(I1,I2)
C THIS SUBROUTINE DRAWS A LINE BETWEEN TWO TOUCH POINTS
      COMMON /DMABUF/IDUM(3700),M(100,3)
10    CALL MOVI3(M(I1,1),M(I1,2),M(I1,3))
      CALL DRWI3(M(I2,1),M(I2,2),M(I2,3))
      RETURN
      END
```

APPENDIX D        SUBROUTINES FOR DISPLAY MANAGEMENT

```
        SUBROUTINE DISP
        DIMENSION T(7),DT(7)
        BYTE IBUF(3)
        COMMON /DMABUF/ IDUM(3100),IFC(200,3),M(100,3)
        COMMON /IPTPS/ IANG(100,2),ICHECK,VEX,IFCC
        COMMON /FACT/IFMAX,NX(30),NA,IPS,NCON,NPOL,ICCN,
     1  IVECT,ISUP,IRX
        COMMON /DISPL/ICM,XXD,XYD,XZD,XTD,YXD,YYD,YZD,YTD,
     1  ZXD,ZYD,ZZD,ISHAD,IARM,IWALL,IROLL,JSTICK,IDOTR
        DATA INTPNT,INWALL,INARM /13,13,13/
C DRAW INSTRUCTIONS
        IF(ICM EQ "113 OR IRX EQ -5)GOTO 208
        IRX=0
        IQS=0
        IF(ICM NE "117)GOTO 733
C INITIATE JOYSTICK ROTATIONS
        IQS=1
        IF(JSTICK EQ 0)JSTICK=-1        ' JOYSTICK FLAG
        JSTICK=-JSTICK
        IF(JSTICK NE 1)IROLL=0
733     IF(JSTICK NE 1)GOTO 300
        IROLL=1
        CALL JROL(TX,TY,TZ,IQS)
        IF(IQS EQ -1)JSTICK=-JSTICK
        IF(JSTICK NE 1)IROLL=0
300     IF(ICM NE "125)GOTO 301
C INITIATE SCREEN OSCILLATIONS
        TYPE *,'INPUT CYCLES/SEC AND MAX ANGLE'
        ACCEPT *,PER,OSC
        OSC=OSC/360
        PER=PER*6 283
        TOS=SECNDS(0 0)
        IROLL=1
C CHECK TWO KEY COMMAND CONDITION FLAG
301     IF(ICM NE 0 AND IPCON EQ 1)GOTO 700
        IF(ICM NE 0 AND IPCON EQ 2)GOTO 800
        IF(ICM EQ "120)IPCON=2
        IF(ICM EQ "131)IPCON=1
        IF(JSTICK EQ 1)RETURN
        IF(ICM EQ "131 OR ICM EQ "120)RETURN
        IF(ICM EQ "040)GOTO 400      ' STOP ROTATIONS
        IF(IROLL EQ 1)GOTO 200       ' SKIP FOR ROTATIONS
        IF(ICM EQ "132)STOP
        IF(ICM EQ "123)IRX=-1        ' REDRAW DISPLAY
        IF(ICM NE "115)GOTO 458      ' SET INTENSITY OF MANIP
        INARM=MOD(1+INARM,16)
        CALL MODIFY(IARM)
        CALL SETINT(INARM)
458     IF(ICM NE "127)GOTO 459      ' SET INTENSITY OF WALLS
        INWALL=MOD(1+INWALL,16)
        CALL MODIFY(IWALL)
        CALL SETINT(INWALL)
459     IF(ICM NE "111)GOTO 461      ' SET INTENSITY OF POINTS
        INTPNT=MOD(1+INTPNT,16)
        CALL MODIFY(NCON-1)
        CALL SETINT(INTPNT)
461     IF(ICM EQ "122)IRX=-2        ' INITIALIZE PROGRAM
        IF(ICM EQ "110)IRX=-4        ' STOP QIO
        IF(IRX LT 0)RETURN
602     IF(ICM NE "067)GOTO 603      ' SET TOP VIEW
        VP1=0
        VP2=- 75
        VP3=0
603     IF(ICM NE "061)GOTO 604      ' SET FRONT VIEW
        VP1=- 75
        VP2=0
        VP3=0
604     IF(ICM NE "060)GOTO 605      ' CENTER PICTURE
        TX=0
        TY=0
        TZ=0
605     IF(ICM NE "065)GOTO 200      ' SET SIDE VIEW
        VP1= 5
        VP2=0
        VP3=0
200     IF(ICM NE "114)GOTO 618      ' SET NORMAL VIEW
        VP1=- 55
        VP2= 04
        VP3=0
        S=1
618     IF(ICM EQ "070)DT(1)= 0008      ' SET ROTATION SPEEDS
        IF(ICM EQ "062)DT(1)=- 0008
        IF(ICM EQ "064)DT(2)= 0008
        IF(ICM EQ "066)DT(2)=- 0008
        IF(ICM EQ "105)DT(6)= 0008
        IF(ICM EQ "124)DT(6)=- 0008
        IF(ICM EQ "075)DT(3)=10
        IF(ICM EQ "047)DT(3)=-10
        IF(ICM EQ "133)DT(4)=10
        IF(ICM EQ "134)DT(4)=-10
        IF(ICM EQ "073)DT(7)=10
        IF(ICM EQ "173)DT(7)=-10
        IF(ICM EQ "071)DT(5)= 004
        IF(ICM EQ "063)DT(5)=- 004
        DO 450 I=1,7
450     IF(ABS(DT(I)) GT  00001)IROLL=1
        J=0
        DO 451 I=1,7
        T(I)=T(I)+DT(I)
451     IF(T(I) NE 0)J=J+1
        IF(J EQ 0 AND PER EQ 0 )IROLL=0
460     VP3=VP3+T(6)
        VP2=VP2+T(1)
```

```
       VP1=VP1+T(2)
       TOS=SECNDS(TOS1)
       TOS1=TOS1+TOS
       TOS=MOD(TOS1*PER,6 283)
       VPT=VP1+OSC*SIN(TOS)
       TY=TY+T(3)/S
       TX=TX+T(4)/S
       TZ=TZ+T(7)/S
       S=S*(1 +T(5))
       IF(S GT 15 9)S=15 9
       DO 452 I=1,7
452    DT(I)=0
       ICM=0
201    SP1=SIN(VPT*6 28)
       SP2=SIN(VP2*6 28)
       CP1=COS(VPT*6 28)
       CP2=COS(VP2*6 28)
       CP3=COS(VP3*6 28)
       SP3=SIN(VP3*6 28)
C CALCULATE SCREEN TRANSFORMATIONS
       XXD=CP1*CP3*S
       XYD=CP1*SP3*S
       XZD=-SP1*S
       XTD=(TX*XXD+TY*XYD+TZ*XZD)
       YXD=(-CP2*SP3+SP2*SP1*CP3)*S
       YYD=(CP2*CP3+SP2*SP1*SP3)*S
       YZD=SP2*CP1*S
       YTD=(TX*YXD+TY*YYD+TZ*YZD)
       ZXD=(SP2*SP3+CP2*SP1*CP3)*S
       ZYD=(-SP2*CP3+CP2*SP1*SP3)*S
       ZZD=CP2*CP1*S
       RETURN
400    DO 401 I=1,7
401    T(I)=0
       RETURN
C  INITIATE POLYHEDRON CONSTRUCTION
700    IF(ICM NE "107)GOTO 701
       CALL LDPTRO(NCON)
       IFMAX=0
       ICCN=1
       IPS=0
       DO 121 J=1,3
       DO 121 I=1,100
       M(I,J)=0
       IFC(I,J)=0
       IFC(I*2,J)=0
       IF(J EQ 3)GOTO 121
       IANG(I,J)=0
121    CONTINUE
701    IF(ICM NE "122)GOTO 703   ' RESTART POLYHEDRON
       IPS=IPST
       ICCN=1
```

```
703     IF(ICM NE "127)GOTO 704    ' REDRAW WALLS
        DO 705 I=1,IFMAX
779     DO 705 J=1,3
        IF(IFC(I,J) GT IFC(I,1+MOD(J,3)))GOTO 705
        CALL VECT(IFC(I,J),IFC(I,1+MOD(J,3)))
705     CALL MGSEND
704     IF(ICM NE "101)GOTO 70' CONNECT POINTS ALREADY READ
        CALL LDPTRO(NCON)
        ICCN=1
        DO 777 I=1,IFMAX
        DO 777 J=1,3
777     IFC(I,J)=0
        IFMAX=0
        IPST=IPS
        DO 707 I=1,IPST
        IPS=I
707     CALL CON
706     IF(ICM NE "124)GOTO 708  ' DRAW TOUCH VECTORS
        DO 709 I=1,IPS
        CALL MOVI3(M(I,1),M(I,2),M(I,3))
        CTHET=COS(FLOAT(IANG(I,1))* 0001)
        STHET=SIN(FLOAT(IANG(I,1))* 0001)
        CPHI=COS(FLOAT(IANG(I,2))* 0001)
        SPHI=SIN(FLOAT(IANG(I,2))* 0001)
        MXX=100 *CTHET*CPHI+M(I,1)
        MYY=100 *STHET*CPHI+M(I,2)
        MZZ=100 *SPHI+M(I,3)
709     CALL DRWI3(MXX,MYY,MZZ)
        CALL MGSEND
C DRAW CONTOURS
708     IF(ICM NE "103 AND ICM NE "110)GOTO 781
        MYY=50
        IF(ICM EQ "103)GOTO 710
        WRITE(5,783)
783     FORMAT(' INPUT NUMBER OF SECTIONS')
        READ(5,*)MYY
782     CALL CTOUR(MYY)
710     IF(ICM NE "120)GOTO 711  ' CHECK IF FACETS FOLD
        ICW5=0
        DO 713 I=1,IFMAX
        DO 713 J=1,IFMAX
        DO 713 K=1,3
        MYY=IFC(J,K)
        MZZ=IFC(J,1+MOD(K,3))
        IF(MYY GE MZZ)GOTO 713
        MXX=4
        CALL PIERC(MYY,MZZ,I,I,CP1,CP2,SP1,MXX)
        IF(MXX NE 6)GOTO 713
        ICW5=ICW5+1
        WRITE(5,714)MYY,MZZ,IFC(I,1),IFC(I,2),IFC(I,3)
714     FORMAT(' LINE',I4,' TO',I4,' PIERCES FACET',3I4)
713     CONTINUE
```

```
711      IF(ICM NE "106)GOTO 715  ' RECHECK FACETS
         MZZ=0                    ' FOR SMOOTHNESS
         DO 716 I=1,IFMAX
         DO 433 J=1,3
         MXX=IFC(I,J)
         MYY=IFC(I,1+MOD(J,3))
         IF(MXX GE MYY)GOTO 433
         IF(IFCC EQ 1)MZZ=MZZ+1
         CALL FACE(MXX,MYY)
433      CONTINUE
716      CONTINUE
         WRITE(5,12)MZZ
12       FORMAT('   OF FACET CHANGES=',I6)
715      IF(ICM NE "116)GOTO 717     ' DRAW POINT NUMBERS
         DO 719 I=1,IPS
         CALL MOVI3(M(I,1),M(I,2),M(I,3))
         ENCODE(3,718,IBUF)I
         CALL CHAR(IBUF,3,1,0)
718      FORMAT(I3)
         CALL MGSEND
719      CONTINUE
717      IF(ICM NE "114)GOTO 725  ' LOOK AT FACET PAIR DATA
         WRITE(5,729)
         READ(5,*)K1,K2
         ICHECK=1
         CALL FACE(K1,K2)
         ICHECK=0
725      IF(ICM NE "111)GOTO 726
         ICHECK=2
         WRITE(5,729)
729      FORMAT(' INPUT K1 AND K2')
         READ(5,*)K1,K2
         CALL FACE(K1,K2)
         ICHECK=0
726      IF(ICM NE "115)GOTO 727  ' GET COORDS OF A POINT
         WRITE(5,728)             ' AND ITS TOUCH VECTOR
728      FORMAT(' INPUT POINT  ')
         READ(5,*)I
         WRITE(5,*)M(I,1),M(I,2),M(I,3)
         CTHET=COS(FLOAT(IANG(I,1))* 0001)
         STHET=SIN(FLOAT(IANG(I,1))* 0001)
         CPHI=COS(FLOAT(IANG(I,2))* 0001)
         SPHI=SIN(FLOAT(IANG(I,2))* 0001)
         MXX=100 *CTHET*CPHI+M(I,1)
         MYY=100 *STHET*CPHI+M(I,2)
         MZZ=100 *SPHI+M(I,3)
         WRITE(5,*)MXX,MYY,MZZ
727      IF(ICM NE "112)GOTO 735  ' WRITE ALL FACET DATA
         DO 731 I=1,IFMAX
         WRITE(5,*)IFC(I,1),IFC(I,2),IFC(I,3)
731      CONTINUE
735      IF(ICM NE "130)GOTO 736 ' CHECK IF ANY TOUCH VECTORS
```

```
      DO 737 I=1,IFMAX          ' PIERCE THE POLYHEDRON
      DO 738 J=1,IPS
      MXX=0
      IF(J EQ IFC(I,1) OR J EQ IFC(I,2) OR J EQ IFC(I,
     1 3))GOTO 738
      CALL PIERC(J,IFC(I,1),IFC(I,2),IFC(I,3),CP1,CP2,
     1 SP1,MXX)
      IF(CP1 EQ 0 OR SP1 LE 0)GOTO 738
      CP2=SP1/40
      WRITE(5,739)J,IFC(I,1),IFC(I,2),IFC(I,3),CP2
739   FORMAT(' VECTOR',I4,' PIERCES FACET',3I4,'  AT',
     1 F9 4,' INCHES')
738   CONTINUE
737   CONTINUE
736   IPCON=0
      RETURN
800   IF(ICM NE "104)GOTO 802      ' DRAW POINTS
      DO 803 I=1,IPS
803   CALL PNTI3(M(I,1),M(I,2),M(I,3))
      CALL MGSEND
802   IF(ICM NE "112)GOTO 804      ' DELETE POINT
      TYPE *,'INPUT NUMBER OF POINT TO DELETE,0=END'
      ACCEPT *,MEND
      IF(MEND EQ 0 OR MEND GT IPS)RETURN
      DO 805 I=MEND,IPS
      M(I,1)=M(I+1,1)
      M(I,2)=M(I+1,2)
      M(I,3)=M(I+1,3)
      IANG(I,1)=IANG(I+1,1)
805   IANG(I,2)=IANG(I+1,2)
      IFMAX=0
      IPS=IPS-1
      DO 806 I=1,200
      IFC(I,1)=0
      IFC(I,2)=0
806   IFC(I,3)=0
804   IF(ICM NE "103)GOTO 807    ' CLEAR ALL POINTS
      DO 808 I=1,100
      DO 808 J=1,3
      M(I,J)=0
      IFC(I,J)=0
      IFC(I+100,J)=0
      IF(J EQ 3)GOTO 808
      IANG(I,J)=0
808   CONTINUE
      ICCN=0
      IPS=0
      IFMAX=0
807   IF(ICM EQ "113)ICCN=0 'STOP POINT CONNECTIONS
      IF(ICM NE "114)GOTO 809 'WRITE DATA TO FILE
      TYPE *,'FOR LEVIDATA "1" FOR MEGATEK "2"'
      ACCEPT *,IWRT
```

```
        IF(IWRT EQ 1)OPEN(UNIT=1,NAME='LEX DAT',TYPE='NEW')
        IF(IWRT EQ 2)OPEN(UNIT=1,NAME='MEG DAT',TYPE='NEW')
        WRITE(1,*)IPS,IFMAX
        DO 810 I=1,IPS
        MXX=M(I,1)*XXD+M(I,2)*XYD+M(I,3)*XZD+XTD
        MYY=M(I,1)*YXD+M(I,2)*YYD+M(I,3)*YZD+YTD
        MZZ=M(I,1)*ZXD+M(I,2)*ZYD+M(I,3)*ZZD
        IF(IWRT EQ 2)WRITE(1,*)M(I,1),M(I,2),M(I,3)
810     IF(IWRT EQ 1)WRITE(1,*)MXX,MYY,MZZ
        DO 811 I=1,IFMAX
811     WRITE(1,*)IFC(I,1),IFC(I,2),IFC(I,3),
     1  IANG(I,1),IANG(I,2)
        CLOSE(UNIT=1,DISPOSE='SAVE')
809     IF(ICM NE "122)GOTO 815 ' READ DATA FROM FILE
        OPEN(UNIT=1,NAME='MEG DAT',TYPE='OLD')
        READ(1,*)IPS,IFMAX
        DO 814 I=1,IPS
814     READ(1,*)M(I,1),M(I,2),M(I,3),IANG(I,1),IANG(I,2)
        DO 813 I=1,IFMAX
813     READ(1,*)IFC(I,1),IFC(I,2),IFC(I,3)
        CLOSE(UNIT=1,DISPOSE='SAVE')
815     IF(ICM NE "116)GOTO 818 '  ARRANGE TO WRITE LARGE
        IF(IDOTR EQ 1)GOTO 819  ' LARGE NUMBER OF POINTS
        OPEN(UNIT=2,NAME='DOT DAT',TYPE='NEW')
        IDOTR=1
        GOTO 818
819     CLOSE(UNIT=2,DISPOSE='SAVE')
        IDOTR=0
818     IF(ICM NE "115)GOTO 820' READ MORE THAN 100 POINTS
        OPEN(UNIT=2,NAME='DOT DAT',TYPE='OLD')
        DO 822 I=1,8000
        READ(2,*,END=821)MMX,MMY,MMZ
        CALL PNTI3(MMX,MMY,MMZ)
822     CALL MGSEND
821     CLOSE(UNIT=2,DISPOSE='SAVE')
        TYPE *,I
820     IPCON=0
        RETURN
C PRINT DISPLAY CONTROL INSTRUCTIONS
208     TYPE *,' TO ROTATE TYPE '
        TYPE *,' 8=UP  2=DOWN  6=RIGHT  4=LEFT  5=FRONT ',
     1  '1=SIDE  7=TOP'
        TYPE *,'  L=ORIENTED VIEW   E=CW  T=CCW'
        TYPE *,' TO TRANSLATE TYPE '
        TYPE *,' "="=UP,"=DOWN,[=RIGHT,=LEFT,O=CENTER,, ',
     1  '=FOWARD,{=BACK'
        TYPE *,' 9=ZOOM UP  3=ZOOM DOWN, TYPE "M"',
     1  'FOR MANIPULATOR'
        TYPE *,' TYPE "S" TO ERASE, "W" FOR WALLS, I FOR ',
     1  'POINT INT'
        TYPE *,' TYPE "Z" TO EXIT, "R" TO REPEAT, "K" FOR ',
     1  'INSTRUCTIONS'
```

```
      TYPE *,' TYPE  "H" TO HALT KEYBOARD ROTATIONS'
      TYPE *,' C'
      TYPE *,' "P"   FOR POINT MANIPULATION -- THEN TYPE'
      TYPE *,'  N= READ POINTS TO DOT   R=RECALL POINT ',
     1  'AND LINE DATA'
      TYPE *,' TYPE D=DRAW POINTS J=DELETE SPECIFIC POINT'
      TYPE *,' C=CLEAR EVERYTHING, K=DOT ENABLE, L=SAVE ',
     1  'FOR LEX OR MEG'
      TYPE *,' C'
      TYPE *,' FOR POINT CONNECTIONS -- FIRST TYPE ',
     1  '"Y" THEN- '
      TYPE *,' G=START, S=START WITH SURFACE, B=SUPPRESS',
     1  ' BASE POINTS'
      TYPE *,' E=END, R=RESUME, W=REDRAW, T=TOUCH ',
     1  'VECTORS, C=50 CONTOUR'
      TYPE *,' SECTIONS, H=CONTOUR SECTIONS, A=CONNECT ',
     1  'POINTS ALREADY READ'
      TYPE *,' V=CHANGE CONCAVITY FACTOR  X=CHECK TOUCH ',
     1  'VECTOR PIERCING'
      TYPE *,' P=CHECK LINE PIERCING OF FACETS  F=CHECK',
     1  ' ALL FACETS'
      TYPE *,' I=CHANGE FACETS,L=LOOK AT FACETS,N=',
     1  'NUMBER FACETS'
      TYPE *,' M=COORDS OF POINT  J=FACET NUMBERS'
      TYPE *,' O=JOYSTICK OR TRACKBALL'
      IRX=0
      RETURN
      END
      SUBROUTINE CTOUR(IS)
C THIS SUBROUTINE DRAWS CONTOURS AROUND THE POLYHEDROM
C IN THE X-Y PLANE
C (IS) IS THE NUMBER OF CONTOURS
      DIMENSION NF(4)
      COMMON /DMABUF/ IDUM(3100),IFC(200,3),M(100,3)
      COMMON /IPTPS/ IANG(100,2)
      COMMON /FACT/ IFMAX,NX(30),NA,IPS,NCON,NPOL,ICCN,
     1  IVECT,ISUP,IRX
      COMMON /DISPL/ ICM,XXD,XYD,XZD,XTD,YXD,YYD,YZD,
     1  YTD,ZXD,ZYD,ZZD,ISHAD,IARM,IWALL,IROLL
      MAXZ=-2000
      MINZ=2000
C FIND THE MAX AND MIN Z VALUES OF THE POLYHEDRON
      DO 10 I=1,IPS
      IF(M(I,3) GT MAXZ)MAXZ=M(I,3)
10    IF(M(I,3) LT MINZ)MINZ=M(I,3)
      S=FLOAT(MAXZ-MINZ)/FLOAT(IS+1)
        DO 20 I=1,IS
        IZ=MAXZ-I*S   ' IZ IS THE GAP BETWEEN CONTOURS
          DO 30 J=1,IFMAX
C SEE IF FACET LAYS ACROSS THE CONTOUR IN QUESTION
432         DO 40 K=1,3
      IF(M(IFC(J,K),3) GE IZ AND M(IFC(J,1+MOD(K,3)),
     1  3) LE IZ)GOTO 50
40          CONTINUE
          GOTO 30
50        NF(1)=IFC(J,1)
          NF(2)=IFC(J,2)
          NF(3)=IFC(J,3)
          NF(4)=IFC(J,1
          D=FLOAT((IZ-M(NF(K),3)))/FLOAT((M(NF(K),
     1    3)-M(NF(K+1),3)))
            IY1=D*(M(NF(K),2)-M(NF(K+1),2))+M(NF(K),2)
            IX1=D*(M(NF(K),1)-M(NF(K+1),1))+M(NF(K),1)
          CALL MOVI3(IX1,IY1,IZ)
45          DO 70 K=1
        IF(M(NF(K),3) LE IZ AND M(NF(K+1),3) GE IZ)GOTO 80
            GOTO 70
80          D=FLOAT((IZ-M(NF(K),3)))/FLOAT((M(NF(K),
      1     3)-M(NF(K+1),3)))
90          IY2=D*(M(NF(K),2)-M(NF(K+1),2))+M(NF(K),2)
            IX2=D*(M(NF(K),1)-M(NF(K+1),1))+M(NF(K),1)
            IF(IY1 EQ -1900 AND IY2 EQ -1900)GOTO 70
          CALL DRWI3(IX2,IY2,IZ)
70          CONTINUE
30        CONTINUE
          CALL MGSEND
20      CONTINUE
        RETURN
        END
```

## APPENDIX E     PROGRAM FOR RASTER DISPLAY OF POLYHEDRON

```
        PROGRAM DRW
C THIS PROGRAM DRAWS A SHADED PICTURE OF A POLYHEDRON
C GIVEN THE 3-D COODINATES OF ALL ITS POINTS AND ITS
C CONNECTIVITY DATA
        DIMENSION NF(4),M(100,3),IFC(200,3),NN(200,2)
        DIMENSION R1(3),R2(3),IY(3)
        INTEGER BUFF1(200),BUFF2(200),BUFF3(200)
        DATA MM1,MM2,MM3/255,255,255/
C INITIALIZE DISPLAY
        CALL DSOPN(2,IE)
        CALL DSCSL(2,0,0)
        CALL DSCER
        CALL DSCLR(4095)
        I5=1
        PI=3 1415
C READ DATA FILE
        OPEN(UNIT=4,NAME='DL1 [200,214]O DAT',TYPE ='OLD')
        READ(4,*)IPS,IFMAX
        DO 400 I=1,IPS
400     READ(4,*)M(I,1),M(I,2),M(I,3)
        DO 401 I=1,IFMAX
401     READ(4,*)IFC(I,1),IFC(I,3),IFC(I,2)
        CLOSE(UNIT=4,DISPOSE='SAVE')
        IFM=IFMAX
C REJECT ALL FACETS THAT FACE AWAY
        DO 402 M4=1,IFM
        I=IFM-M4+1
        IF(IFC(I,1) EQ 0)GOTO 402
        DO 403 J=1,3
        R1(J)=M(IFC(I,2),J)-M(IFC(I,1),J)
403     R2(J)=M(IFC(I,3),J)-M(IFC(I,1),J)
        QZ=R1(1)*R2(2)-R1(2)*R2(1)
        IF(QZ LT 0)GOTO 402
        IFMAX=IFMAX-1
        DO 435 J=I,IFM
        DO 435 K=1,3
        IFC(J,K)=IFC(J+1,K)
435     CONTINUE
402     CONTINUE
C ORDER FACETS SO NEAREST ARE DRAWN LAST
        DO 405 J=1,IFMAX
        DO 406 I=1,IFMAX-1
        IF(IFC(I,1) EQ 0)GOTO 406
        M1=MAX0(M(IFC(I,1),3),M(IFC(I,2),3),M(IFC(I,3),3))
        M2=MAX0(M(IFC(I+1,1),3),M(IFC(I+1,2),3),
     1  M(IFC(I+1,3),3))
        IF(M1 GT M2)GOTO 406
407     DO 408 K=1,3
        N=IFC(I,K)
        IFC(I,K)=IFC(I+1,K)
408     IFC(I+1,K)=N
```

```
406     CONTINUE
405     CONTINUE
C DRAW FACETS AS 2 DIMENSIONAL TRIANGLES ON SCREEN
        DO 409 I=1,IPS
        NN(I,1)=M(I,1)*4/25+320
409     NN(I,2)=-M(I,2)*4/25+256
        DO 410 I=1,IFMAX
        DO 413 J=1,3
413     IY(J)=J
        DO 411 J=1,3
        DO 412 K=1,2
        IF(NN(IFC(I,IY(K)),2) LE NN(IFC(I,IY(K+1)),
     1  2))GOTO 412
        N=IY(K)
        IY(K)=IY(K+1)
        IY(K+1)=N
412     CONTINUE
411     CONTINUE
        IYT=NN(IFC(I,IY(1)),2)
        IYM=NN(IFC(I,IY(2)),2)
        IYB=NN(IFC(I,IY(3)),2)
        IXT=NN(IFC(I,IY(1)),1)
        IXM=NN(IFC(I,IY(2)),1)
        IXB=NN(IFC(I,IY(3)),1)
        IF(IYB EQ IYT)GOTO 410
        IF(IYM EQ IYT)GOTO 441
        F=FLOAT(IYM-IYT)/FLOAT(IYB-IYT)
440     IX2M=IXT+FLOAT(IXB-IXT)*F
        IF(IYB LT 1)GOTO 410
        IF(IYT GT 512)GOTO 410
        DO 414 J=IYT,IYM
        F=FLOAT(J-IYT)/FLOAT(IYM-IYT)
        IX1=IXT+FLOAT(IXM-IXT)*F
        IX2=IXT+FLOAT(IX2M-IXT)*F
        CALL TRI(IX1,IX2,J,I)
414     CONTINUE
441     IF(IYB EQ IYM)GOTO 410
        DO 415 J=IYM,IYB
        F=FLOAT(J-IYM)/FLOAT(IYB-IYM)
        IX1=IXM+FLOAT(IXB-IXM)*F
        IX2=IX2M+FLOAT(IXB-IX2M)*F
        CALL TRI(IX1,IX2,J,I)
415     CONTINUE
410     CONTINUE
C READ DATA FROM TRACKBALL
1       CALL TBALL(IXX,IYY,IZ)
        IF(IXX EQ 0 AND IYY EQ 0 AND IZ EQ 0)GOTO 1
        IAX=IAX+IXX*TS
        IAY=IAY+IYY*TS
        IF(IZ EQ 1)GOTO 1
        IF(IZ EQ 0)GOTO 436
        IF(IZ EQ IZ2)GOTO 1
        IF(IZ NE 3)GOTO 437
        TYPE *,'INPUT BACKGROUND BLUE-GREEN-RED SHADES,
     1  0-255'
        ACCEPT *,M1,M2,M3
        CALL DSLLU(1024,M1,1024,M1)
        CALL DSLLU(2048,M2,2048,M2)
        CALL DSLLU(3072,M3,3072,M3)
437     IF(IZ NE 6)GOTO 438
        TYPE *,'INPUT OBJECT BLUE-GREEN-RED SHADES,0-255'
        ACCEPT *,MM1,MM2,MM3
438     IF(IZ EQ 7)CALL EXIT
C CALCULATE SHADES FOR ALL TRIANGLES
436     AX=FLOAT(IAX)*PI/180
        AY=FLOAT(IAY)*PI/180
        IZ2=IZ
        SX=SIN(AX)
        SY=COS(AX)*SIN(AY)
        SZ=-COS(AX)*COS(AY)
        DO 422 I=1,IFMAX
        IF(IFC(I,1) EQ 0)GOTO 422
        DO 423 J=1,3
        R1(J)=M(IFC(I,2),J)-M(IFC(I,1),J)
423     R2(J)=M(IFC(I,3),J)-M(IFC(I,1),J)
        QX=R1(2)*R2(3)-R1(3)*R2(2)
        QY=R1(3)*R2(1)-R1(1)*R2(3)
        QZ=R1(1)*R2(2)-R1(2)*R2(1)
        DLEN=SQRT(QX*QX+QY*QY+QZ*QZ)
        DENS=(QX*SX+QY*SY+QZ*SZ)/DLEN
        IF(DENS LT 0)DENS=0
        BUFF1(I)=DENS*FLOAT(MM1)
        BUFF2(I)=DENS*FLOAT(MM2)
        BUFF3(I)=DENS*FLOAT(MM3)
422     CONTINUE
C SEND SHADES TO DISPLAY
        CALL DSLWT(1025,IFMAX,BUFF1)
        CALL DSLWT(2049,IFMAX,BUFF2)
        CALL DSLWT(3073,IFMAX,BUFF3)
        GOTO 1
        END
        SUBROUTINE TRI(IX1,IX2,J,I)
C THIS SUBROUTINE DRAWS THE ACTUAL LINES ON THE SCREEN
        IF(J GT 512)RETURN
        IF(J LT 1)RETURN
        IF(IX1 GT 640)IX1=640
        IF(IX2 GT 640)IX2=640
        IF(IX1 LT 1)IX1=1
        IF(IX2 LT 1)IX2=1
        IF(ABS(IX1-IX2) LT 1)RETURN
        CALL DSVEC(IX1,J,IX2,J,I)
        RETURN
        END
```