THREE-DIMENSIONAL VARIATIONAL GEOMETRY

IN

COMPUTER-AIDED DESIGN

by

VINCENT C. LIN

B.S., June 1979, University of California, Berkeley

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

(MAY 1981)

Signature of Author _____
Department of Mechanical Engineering
May, 1981

Certified by _____
Thesis Supervisor

Accepted by _____
Chairman, Department Commitee

THREE-DIMENSIONAL VARIATIONAL GEOMETRY

IN

COMPUTER-AIDED DESIGN

by

VINCENT C. LIN

Submitted tc the Department of Mechanical Engineering on May, 1981 in partial fulfillment of the requirements for the degree Master of Science.

ABSTRACT

The focus of this investigation is a method for modification of geometry in computer-aided design systems. Called variational geometry, it uses a single representation to describe the complete family of geometries which share a given topology. A three-dimensional shape model is defined with respect to a set of characteristic points. The positions of these characteristic points are fixed by a set of nonlinear algebraic equations which describe constraints imposed by engineering dimensions. Explicit constraints fix distances, angles, radii, and diameters. Implicit constraints fix parallel and perpendicular planes, and prevent rigid body translation and rotation. Area and volumetric properties may also be used as constraints.

Modification of three-dimensional geometry is accomplished with minimal user input. The dimension to be changed is selected and its new value entered. A modified Newton-Raphson method is used to solve the set of constraint equations for the new geometry with no additional user input.

A procedure for minimizing computational requirements is presented. The procedure uses known relationships between dimensional constraints and their associated characteristic points to identify the minimunm set of constraint equations and characteristic points which must be solved to effect a given dimensional change. For n characteristic points, solution time is shown to be of order O(n).

Thesis Supervisor: David C. Gossard
Title: Associate Professor

ACKNOWLEDGEMENTS

Many thanks to Professor David Gossard for his guidance and advice.

To Dr. Phillip Meyfarth, Mr. Robert Congdon, and Mr. Kunwoo Lee of the MIT CAD LAB for their help and support; to Robert Light of Deere and Company and Mark Stiles of MIT's Joint Computer Facility for their contributions; to Patty, Ray, Jo, and my parents for providing moral support; and to A.C. for inspiration.

This project was sponsored by ComputerVision.

TABLE OF CONTENTS

LIST OF FIGURES

TABLE OF SYMBOLS

SET NOTATION

$\{\ \}$: elements of a set

$\in$: is an element of

$\notin$: is not an element of

$\emptyset$: empty set

$\cup$: union of two sets

$\cap$: intersection of two sets

$\wedge$: and

$\vee$: or

$|$: such that

$\subseteq$: subset of

$\subset$: proper subset of

ANALYTICAL GEOMETRY

(x,y,z) : coordinates of a point

$P=(x,y,z)$: Point P with coordinates (x,y,z)

$P1P2$: line segment from $P1=(x1,y1,z1)$ to $P2=(x2,y2,z2)$

$|P1P2|$: length of segment P1P2

$\overline{P1P2}$: vector extending from $P1=(x1,y1,z1)$ to $P2=(x2,y2,z2)$

$|\overline{P1P2}|$: magnitude of vector $\overline{P1P2}$

$\underline{u}$: vector u

$[u1,u2,u3]$: scalar components of a vector $\underline{u}$
$[u1,u2,u3] = u1\hat{i} + u2\hat{j} + u3\hat{k}$

$|\underline{u}|$: magnitude of vector $\underline{u}$
$|\underline{u}| = \sqrt{u1^2 + u2^2 + u3^2}$

$\hat{w}$: unit vector w

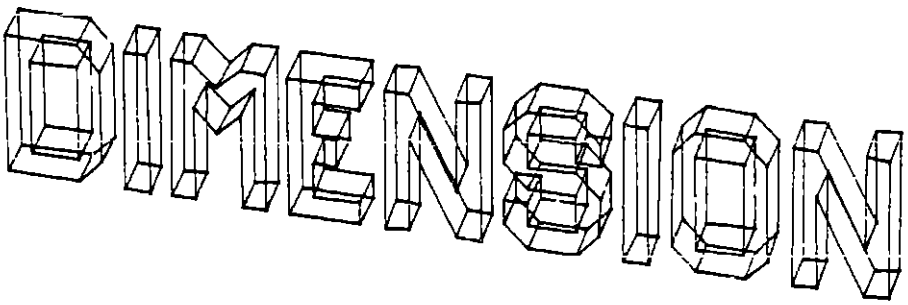
$\underline{u}.\underline{v}$: scalar(dot) product of vectors $\underline{u}$ and $\underline{v}$
$\underline{u}.\underline{v} = u1v1 + u2v2 + u3v3$

$\underline{u}X\underline{v}$: vector(cross) product of vectors $\underline{u}$ and $\underline{v}$
$\underline{u}X\underline{v}$ = [u2v3-u3v2,u3v1-u1v3,u1v2-u2v1]

$\underline{u}.(\underline{v}X\underline{w})$: scalar triple product

$$\underline{u}.(\underline{v}X\underline{w}) = \begin{vmatrix} u1 & u2 & u3 \\ v1 & v2 & v3 \\ w1 & w2 & w3 \end{vmatrix}$$

$$= u1 \begin{vmatrix} v2 & v3 \\ w2 & w3 \end{vmatrix} + u2 \begin{vmatrix} v3 & v1 \\ w3 & w1 \end{vmatrix} + u3 \begin{vmatrix} v1 & v2 \\ w1 & w2 \end{vmatrix}$$


DIMENSION

1.0 INTRODUCTION

1.1 OBJECTIVES

One of the most important features of computer-aided design systems is the ability to represent geometry by a shape model. Because of the iterative nature of design, geometry is modified often. A friendly user interface to allow simple modification and manipulation of geometry is essential to the effectiveness of CAD systems and is the central focus of this investigation.

Current geometric modelling systems use a rigid geometry definition structure in which creation of geometric entities (e.g. points, lines, surfaces) require exact specification of their positions in three-dimensional space. This is accomplished either by specifying the placements of the entities through input of x, y, and z coordinates or through a set of translations and rotations of surfaces or three-dimensional primitive solids.

The objective of this thesis investigation is to study methods by which three-dimensional geometries can be manipulated and modified with minimal user interaction. Of

particular interest is the method called variational geometry, in which a single representation is used to describe the complete family of geometries which share a given topology. A discussion of variational geometry applied to two-dimensional objects is given by Light [20]. The current investigation focuses on three-dimensional geometries and methods for increasing computational efficiency of the method.

The implementation of variational geometry is based upon a topologically and geometrically complete data structure. Faces, edges and vertices are defined with respect to a set of characteristic (defining) points. Three-dimensional constraints are then applied to these characteristic points. Geometry modification is reduced to two steps; selecting the dimension(s) to be changed and entering the new dimensional value(s).

For any dimension change, a matrix method is used to reconcile the set of constraint equations and coordinate points. To increase the speed and efficiency of this method, the relationships between the constraining equations and coordinate points are used to segment the total number of equations and coordinate points into a smaller and more managable subset.

These concepts have been implemented in a prototype system, the DIMENSION system. Currently, the class of shapes which can be modelled include those with planar, cylindrical, spherical, and conical surfaces. The system currently interfaces with four data files containing topology, geometry, constraints, and dimensions. Different parts can be modelled by the creation of new data files.

Input methods which define geometry with low accuracy can be used since the use of variational geometry allows for easy modification of geometry. Once the set of data files has been created, modification of the dimension and related data files is accomplished through a simple interactive procedure.

1.2 RELATED WORK

The concept of constraining the coordinate points of a part in computer-aided design can be seen in the early work of Ivan Sutherland [29]. Sutherland used various relationships between the coordinates of a part to constrain its geometry. For example, to make a line between two points vertical, a constraint satisfaction routine attempted to reduce the difference in the x coordinates of the two points to zero by manipulating the constrained coordinates. Sutherland defined a number of constraints for the two-dimensional case. Sutherland did not use geometric constraints for definition of the complete geometry nor was it used for the purpose of modification of geometry after the part had been created.

Robin Hillyard and Ian Braid [16],[17],[18] developed a more general approach which used constraint equations to relate positions of vertex points of a geometry to the dimensions shown or implied in an engineering drawing.

When the variations in the dimensions are small, the set of equations can be linearized. For example, the equation describing the linear dimension between two points,

Pi and Pj (Figure 1), is given by

$$u = \frac{(\underline{di} - \underline{dj}) \cdot \underline{l}}{|\underline{l}|} \qquad (1.1)$$

where u is the variation in the dimension l, $\underline{di}$ and $\underline{dj}$ are the variations in position of vertices Pi and Pj, $\underline{l}$ is the separation vector between vertices Pi and Pj, and $|\underline{l}|$ is the dimensional value.

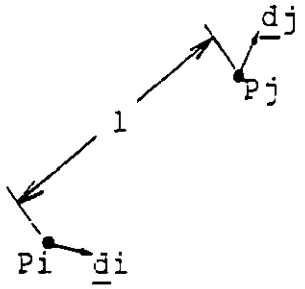


Figure 1, Linear dimension between two points.

The complete set of constraint equations may be expressed in matrix form, $R\underline{d} = \underline{u}$ where R is the rigidity matrix, $\underline{d}$ the vertex displacement vector, and $\underline{u}$ the vector of variations in dimensions (bounded by the tolerances). For a given set of dimensional variations, the vertex displacements can be determined by a relaxation technique.

The use of constraint equations to define geometry was furthur developed by Robert Light [2Ø]. Light developed a prototype two-dimensional system to modify the geometry of two-dimensional parts. Constraints were defined between points for two-dimensional shape models. As an example, the linear distance between two points in two-dimensions, P1=(x1,y1) and P1=(x2,y2) is given by

$$(x2 - x1)^2 - (y2 - y1)^2 - D^2 = Ø. \qquad (1.2)$$

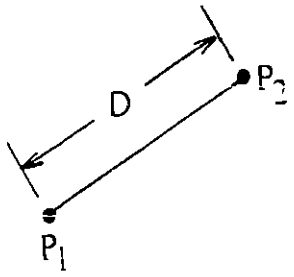


Figure 2, Two-dimensional constraint of the linear distance between two points.

Given the dimensional values, the positions of the points were determined by the Newton-Raphson method to solve the set of simultaneous non-linear constraint equations. The form of the set of equations is J $\underline{dx}$ = $\underline{r}$, where J is the Jacobian matrix, $\underline{dx}$ the displacement vector, and $\underline{r}$ the residual vector.

Overdimensioning, underdimensioning, and redundant dimensioning were detected using a modified Doolittle's method. Methods for increasing computational efficiency for the set of two-dimensional constraints were also developed.

The objectives of this investigation are to identify and derive constraint equations for three-dimensional geometries and to develop methods to increase the efficiency of the numerical computations.

2.0 DATA REPRESENTATION

2.1 GEOMETRY AND TOPOLOGY

In three dimensions, a body is bounded by a set of faces. Faces may be planar or curved. Faces intersect in edges, which may be straight or curved. Edges intersect at vertices.

For the shape of a part to be completely defined, its geometry as well as its topology must be completely specified. Geometry refers to the physical locations of the faces, edges and vertices in three-dimensional space. Topology refers to the way the faces, edges, and vertices of a body are connected. Each face is bounded by a ring of edges and vertices. Each edge is bounded by two vertices and two faces.

From a data base of complete geometrical and topological information, different views can be generated for drawing purposes, volumetric properties can be computed, and part consistency and well-formedness can be automatically checked.

Current geometric modelling system data bases fall into two categories. In the first, complete geometrical information is stored. Topological information is not completely specified. Completeness and part consistency can only be checked visually. This first class of geometric modellers has been called the "wire-frame modellers"

The second category of geometric modellers store complete topological and geometrical information. This can be done in different ways. The method of storage depends on the type of application of the system, the algorithms used in the system, and the tradeoff between storage requirements and time required for searches and calculations.

In the second category of geometric modellers, enough information is stored in the data base so that part consistency can be checked. Conditions of closure, surface orientation, and non-self intersection must be met in order for a shape to be consistent and well-formed. The data base of this type of geometric modelling system has given rise to the term "solid modellers".

In the DIMENSION system, both complete geometrical information as well as complete topological information are stored. This provides the capability to check the well-formedness of a part.

A complete description of the creation of the geometrical and topological data base for the DIMENSION system is presented by Congdon [3]. Only the general structure of how the data is organized and what is stored will be presented here. Currently, only planar, cylindrical, spherical, and conical surfaces can be represented.

The general structure of the geometrical and topological data base for a body is shown in Figure 3, where the rectangles denote data files. In the body data file are stored the number of faces, edges, and vertices which make up the body, as well as pointers to the faces, edges and vertices.

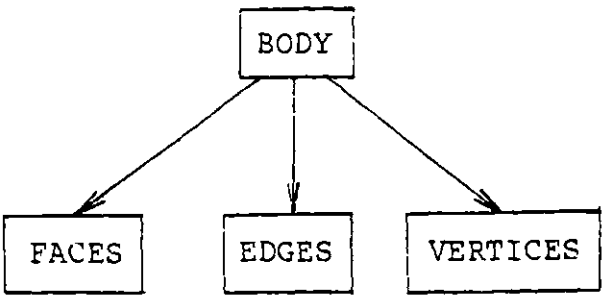


Figure 3, General data base structure.

Geometry of a component is described as a set of face, edge, and vertex files. The structure of the face, edge and vertex files are shown in Figure 4.

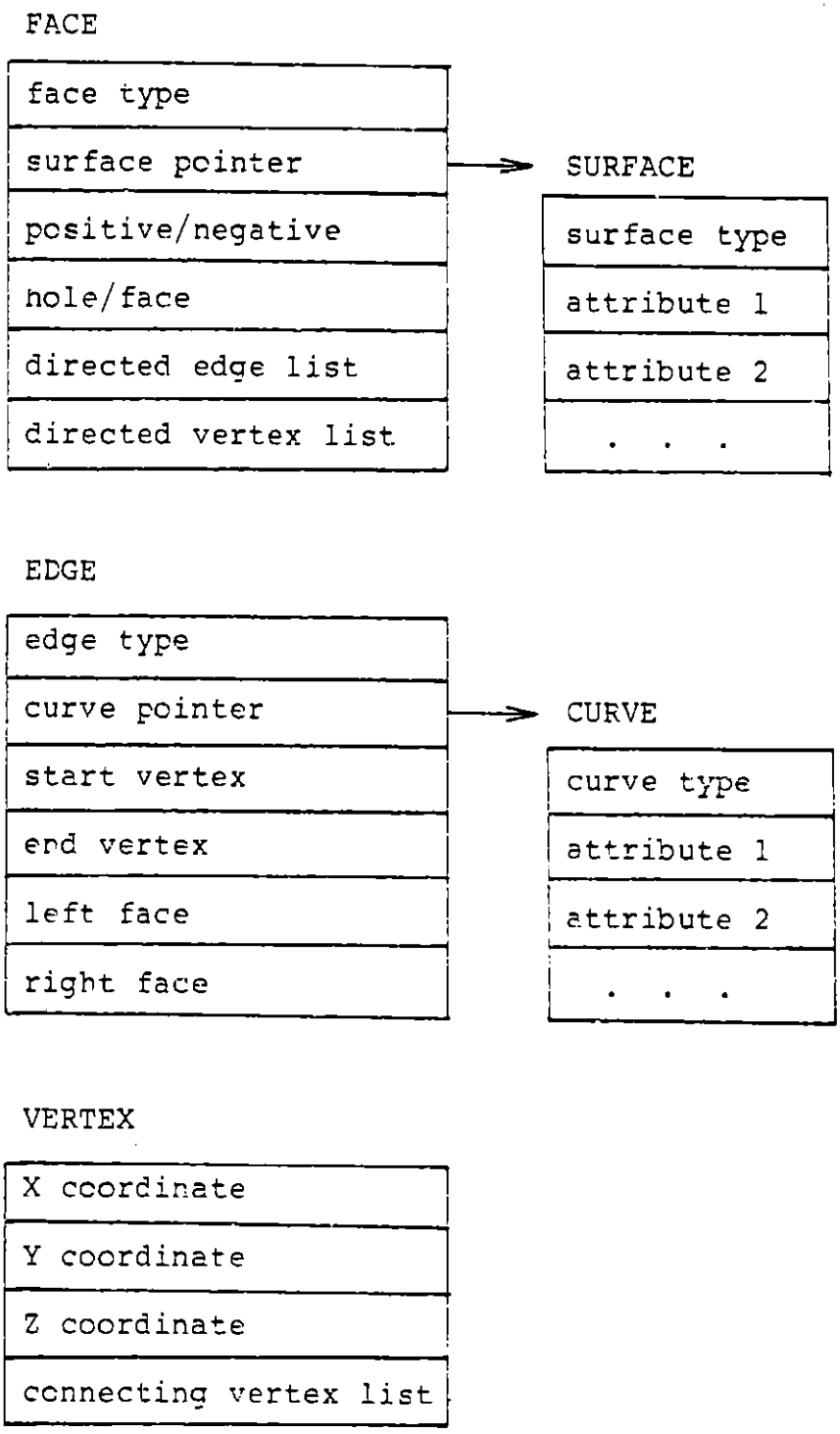


Figure 4, Structure of face, edge, and vertex data files.

The vertex file contains the x, y, and z coordinates of each vertex. Space is allocated for storage of the list of adjacent vertices of each vertex.

The edge file contains the edge type, a pointer to the curve file if the edge is not a straight line, pointers to the edge's start and end vertices and its adjacent left and right faces.

The curve file contains the curve type and its attribute list. Currently, only circular arcs are treated. For a circular arc, its center, radius and normal vector are stored.

The face file contains the face type, a pointer to the surface file, whether it is a positive or a negative face, whether it is a hole or a face (the face number of the face the hole belongs to if hole), the list of directed edges surrounding the face and the list of directed vertices of the face.

The surface file contains the surface type and its attribute list. Currently, only planes, cylindrical, conical, and spherical surfaces are treated. For each surface, space is allocated for storage of coefficients of its implicit equation.

The topological relationships between the faces, edges and vertices are illustrated in Figure 5. Faces point to edges and vertices. Edges point to faces and vertices, and vertices point to vertices.

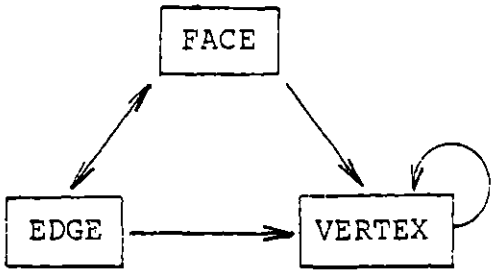


Figure 5, Topological relationships between faces, edges, and vertices.

2.2 CONSTRAINTS

The DIMENSION system provides for constraints between the set of characteristic or defining points and parameters of the surfaces. This information is maintained

in two data files. The structure of the constraint and dimension data files are shown in Figure 6.

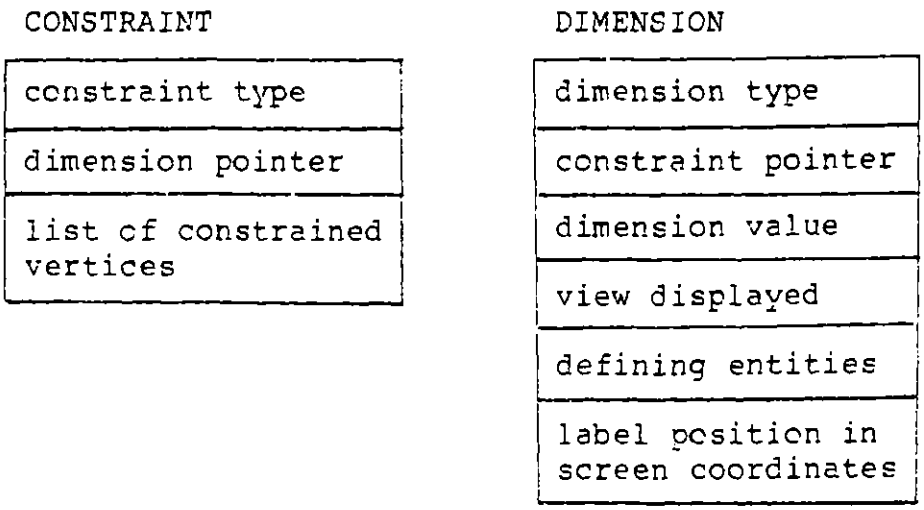


Figure 6, Structure of constraint and dimension data files.

The constraint file contains, for each constraint, the type of constraint and the defining points of the surfaces which are constrained. If the constraint is a dimensional constraint, it contains a pointer to the dimension file. The dimension file is used for display of dimensions on drawings as well as to allow the user to interactively modify dimensional values. The dimension file contains the type of dimension, a pointer to the constraint file, the value of the dimension, the view in which the dimension is to be displayed, the defining points of that dimension in the view which it is to be displayed for purposes of

displaying the dimension lines, and the screen coordinates of the dimension label position.

3.Ø THREE-DIMENSIONAL CONSTRAINTS

3.1 GENERAL

An object represented by N points has n=3N degrees of freedom in three-dimensional space. To fix the position of all the points and, therefore, the geometry of an object, 3N independent items of information, derived from the constraints are needed. A constraint can provide one or more items of information. Each of the items of information can be written as a nonlinear equation in terms of the point coordinates.

A full set of independent constraints will give rise to 3N non-linear equations. This set of equations must be solved simultaneously to determine the coordinate points of the object.

Constraints can be divided into two groups, implicit and explicit. Implicit constraints are usually not represented in engineering drawings but are understood or implied by the draftsman and the reader of the drawing. For example, in the object in Figure 7, parallel planes and perpendicular planes are usually not explicitly labeled.

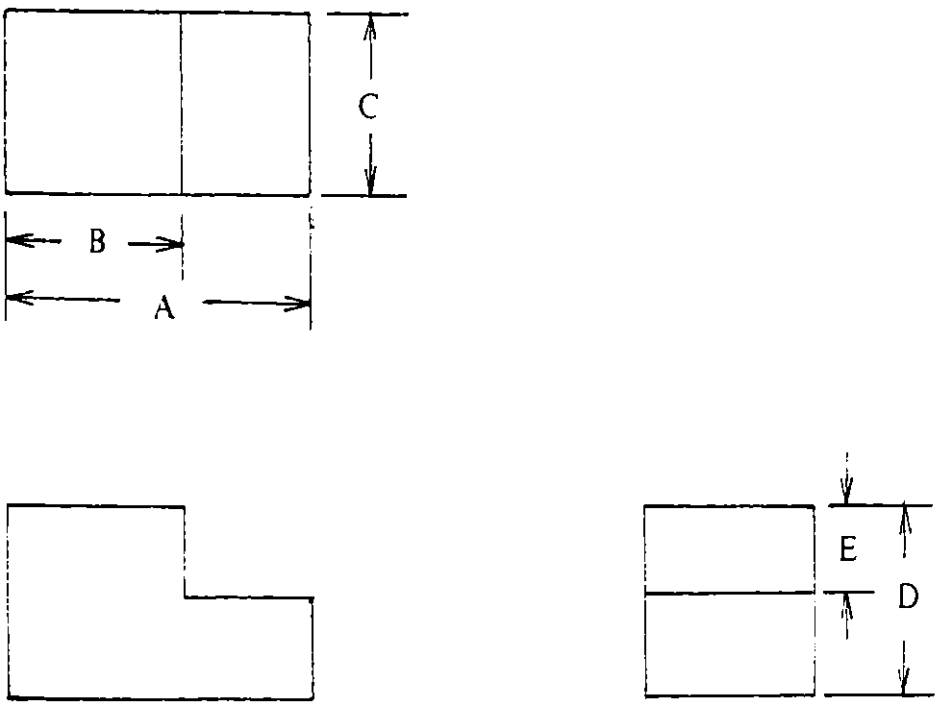


Figure 7, Drawing of a 3-D part.

It is common drafting practice to draw the planes parallel or perpendicular and not give dimensions to specify parallelism and perpendicularity. Exceptions occur when draft angles, parallelism tolerances, and perpendicularity tolerances are explicitly specified.

Explicit constraints specify a dimensional value. Explicit constraints result from distance, angular, radial or diametrical dimensions. In the example shown in Figure 7, dimensions A,B,C,D, and E specify distances between parallel planes. These dimensions result in both explicit and implicit constraints.

The constraint equations for a set of common constraints was derived. The list of three dimensional constraints is shown in Table 1.

LIST OF THREE DIMENSIONAL CONSTRAINTS

1. Rigid body translation.
2. Rigid body rotation.
3. Four points coplanar.
4. M points coplanar
5. Vertical plane.
6. Horizontal plane.
7. Perpendicular planes.
8. Parallel planes.
9. Line parallel to plane.
10. Parallel lines.
11. Equal x distance between two points.
12. Equal y distance between two points.
13. Equal z distance between two points.

14. Equal linear distance between two points.

15. X distance between two points.

16. Y distance between two points.

17. Z distance between two points.

18. Linear distance between two points.

19. Distance from a point to a plane.

20. Distance from a point to a line.

21. Distance between parallel lines.

22. Distance from a line to a plane.

23. Distance between two planes.

24. Angle between two lines.

25. Angle between a line and a plane.

26. Angle between two planes.

Table 1, List of three-dimensional constraints.

In a future CAD system incorporating this approach, the designer could interactively apply the constraints to the topological representation of a three-dimensional object. Alternately, a semi-automatic constraining scheme could be used in which the implicit constraints are automatically generated by the computer and explicit constraints manually assigned by the designer.

Although twenty-six constraints between entities in three-dimensions have been derived, the set of constraints which are used for most common engineering parts is somewhat smaller. Many of the mathematical relationships representing the constraints can be derived from analytical geometry.

3.2 IMPLICIT CONSTRAINTS

1) RIGID BODY TRANSLATION: A body is constrained in rigid body translation if one vertex on the body, Pr=(xr,yr,zr), is fixed. Equation (3.1) fixes x translation. Equation (3.2) fixes y translation. Equation (3.3) fixes z translation.

$$xr = \emptyset \qquad (3.1)$$

$$yr = \emptyset \qquad (3.2)$$

$$zr = \emptyset \qquad (3.3)$$

2) RIGID BODY ROTATION: A body is constrained in rotation by fixing lines on the body from rotation about the three axes. If three points on a body, P1=(x1,y1,z1), P2=(x2,y2,z2), and P3=(x3,y3,z3) are defined as shown in Figure 8 and P1 is fixed in rigid body translation, equation (3.4) fixes a line from rotating about the z axis. Equation (3.5) fixes that same line from rotating about the y axis. Equation (3.6) fixes a line which has no component in the x direction from rotating about the x axis.

$$y2 - y1 = 0 \qquad (3.4)$$

$$z2 - z1 = 0 \qquad (3.5)$$

$$z3 - z1 = 0 \qquad (3.6)$$

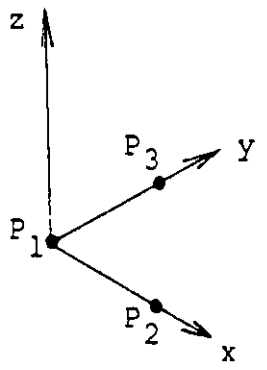


Figure 8, Three points on a body.

3) FOUR POINTS COPLANAR: Three non-colinear points P1=(x1,y1,z1), P2=(x2,y2,z2), and P3=(x3,y3,z3), define a plane. Let P=(x,y,z) be an additional point in that plane. The vectors $\overline{P1P}$, $\overline{P1P2}$, $\overline{P1P3}$ are coplanar so $\overline{P1P}$ is perpendicular to the vector product of $\overline{P1P2}$ and $\overline{P1P3}$.

$$\overline{P1P} \cdot (\overline{P1P2} \times \overline{P2P3}) = \emptyset \qquad (3.7)$$

equation (3.7) may be expanded,

$$\begin{vmatrix} x-x1 & y-y1 & z-z1 \\ x2-x1 & y2-y1 & z2-z1 \\ x3-x1 & y3-y1 & z3-z1 \end{vmatrix} = \emptyset$$

thus, the four points are coplanar when,

$$Ax + By + Cz + D = \emptyset \qquad (3.8)$$

where

$$A = y2z3 - y2z1 - y1z3 - y3z2 + y3z1 + y1z2 \qquad (3.9)$$

$$B = -x2z3 + x2z1 + x1z3 + x3z2 - x3z1 - x1z2 \qquad (3.10)$$

$$C = x2y3 - x2y1 - x1y3 - x3y2 + x3y1 + x1y2 \qquad (3.11)$$

$$D = -x1A - y1B - z1C \qquad (3.12)$$

4) M POINTS COPLANAR: To constrain a M-pointed polygon to be coplanar, M-3 constraints of four points coplanar may be used. The polygon can be subdivided into M-3 quadrilaterals and each constrained to be coplanar.

5) VERTICAL PLANE: A plane, Ax + By + Cz + D = Ø, is constrained to be vertical if the z component of its normal vector, [A,B,C], is zero. If P1=(x1,y1,z1), P2=(x2,y2,z2), and P3=(x3,y3,z3) are three points on the plane, then from Equation (3.11),

$$C = x2y3 - x2y1 - x1y3 - x3y2 + x3y1 + x1y2 = Ø \qquad (3.13)$$

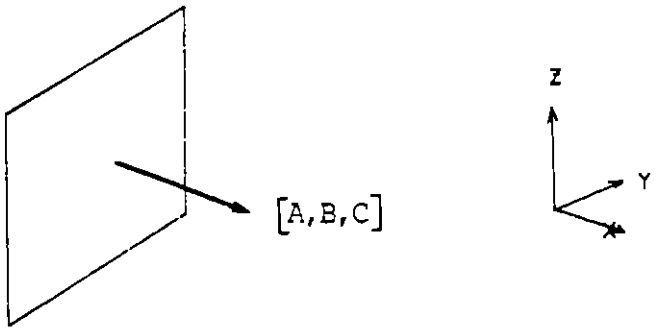


Figure 9, Vertical plane.

6) HORIZONTAL PLANE: The coordinate axis is defined in such a way that the x-y plane defines the horizontal direction. A plane, Ax + By + Cz + D = Ø, is constrained to be horizontal if the x and y components of its normal vector, [A ,B ,C], are zero. If P1=(x1,y1,z1), P2=(x2,y2,z2), and P3=(x3,y3,z3) are three points on the plane, then from Equations (3.9) and (3.1Ø),

$$A = y2z3 - y2z1 - y1z3 - y3z2 + y3z1 + y1z2 = \emptyset \quad (3.14)$$

$$B = -x2z3 + x2z1 + x1z3 + x3z2 - x3z1 - x1z2 = \emptyset \quad (3.15)$$

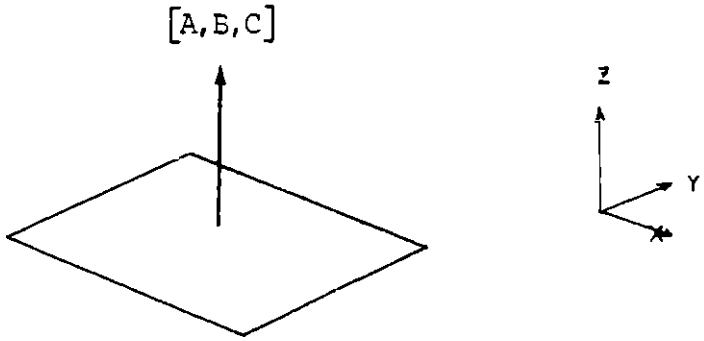


Figure 10, Horizontal plane.

7) PERPENDICULAR PLANES: P1=(x1,y1,z1), P2=(x2,y2,z2), and P3=(x3,y3,z3) are three points on one plane. P4=(x4,y4,z4), P5=(x5,y5,z5), and P6=(x6,y6,z6) are three points on another plane. In order for the two planes, defined by A1x + B1y + C1z + D1 = 0 and A2x + B2y + C2z + D2 = 0, to be perpendicular, their normal vectors, [A1,B1,C1] and [A2,B2,C2], must be perpendicular and their scalar product zero.

$$A1A2 + B1B2 + C1C2 = \emptyset \quad (3.16)$$

where

$$A1 = y2z3 - y2z1 - y1z3 - y3z2 + y3z1 + y1z2$$

```
B1 = -x2z3 + x2z1 + x1z3 + x3z2 - x3z1 - x1z2
C1 =  x2y3 - x2y1 - x1y3 - x3y2 + x3y1 + x1y2
A2 =  y5z6 - y5z4 - y4z6 - y6z5 + y6z4 + y4z5
B2 = -x5z6 + x5z4 + x4z6 + x6z5 - x6z4 - x4z5
C2 =  x5y6 - x5y4 - x4y6 - x6y5 + x6y4 + x4y5
```

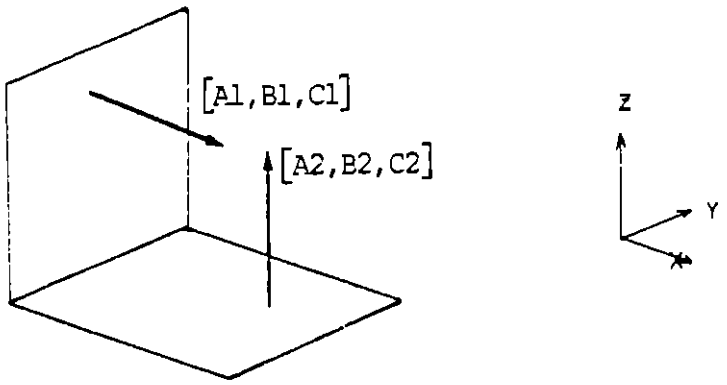


Figure 11, Perpendicular planes.

8) LINE PARALLEL TO A PLANE: P3=(x3,y3,z3), P4=(x4,y4,z4), and P5=(x5,y5,z5) are three points on a plane. A line is constrained to be parallel to the plane, Ax + By + Cz + D = Ø, if the vector on the line, [x2-x1,y2-y1,z2-z1] is perpendicular to the plane normal vector [A,B,C]. Two vectors are perpendicular if their scalar product is zero.

$$A(x2-x1) + B(y2-y1) + C(z2-z1) = \emptyset \qquad (3.17)$$

where

$$A = y4z5 - y4z3 - y3z5 - y5z4 + y5z3 + y3z4$$

$$B = -x4z5 + x4z3 + x3z5 + x5z4 - x5z3 - x3z4$$

$$C = x4y5 - x4y3 - x3y5 - x5y4 + x5y3 + x3y4$$

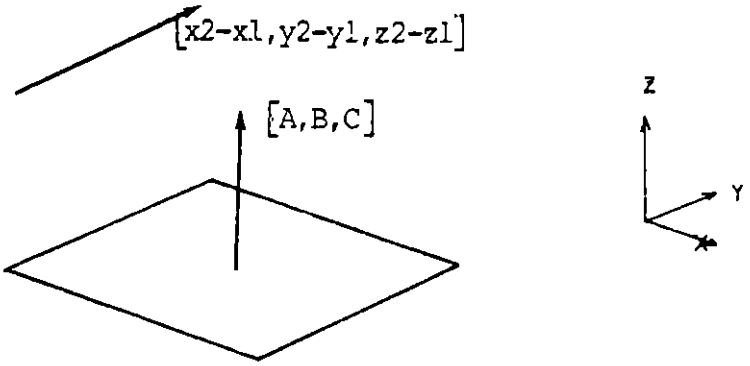


Figure 12, Line parallel to a plane.

9) PARALLEL PLANES: Since this constraint carries two items of information, two equations of a line parallel to a plane are required. The two lines must be intersecting lines of one of the planes. Another interpretation could be to use one constraint of the angle between the two normal vectors of the planes to be zero and one constraint of the normal vectors being coplanar.

10) PARALLEL LINES: This constraint carries two items of information. First, the four points defining the two lines must be coplanar. Second, the angle between the two lines is constrained to be zero.

11) EQUAL X DISTANCE BETWEEN POINT PAIRS: The x distance between P1=(x1,y1,z1) and P2=(x2,y2,z2) and the x distance between P3=(x3,y3,z3) and P4=(x4,y4,z4) are constrained to be equal.

$$\sqrt{(x4-x3)^2 - (x2-x1)^2} = \emptyset \qquad (3.18)$$

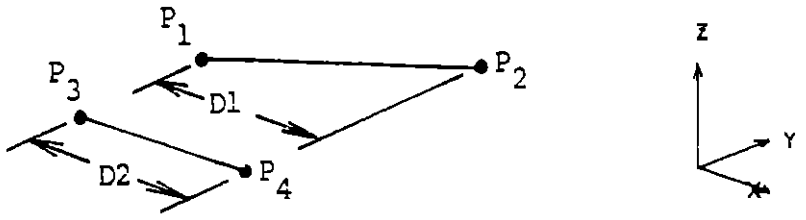


Figure 13, Equal x distance between point pairs.

This form of the equation was used instead of $(x4-x3)^2-(x2-x1)^2 = \emptyset$ to keep the values of the resulting partial derivatives in the Jacobian matrix approximately equal in

magnitude with the values of the partials from other constraints to minimize roundoff errors in the numerical methods.

12) EQUAL Y DISTANCE BETWEEN POINT PAIRS: The y distance between P1=(x1,y1,z1) and P2=(x2,y2,z2) and the y distance between P3=(x3,y3,z3) and P4=(x4,y4,z4) are constrained to be equal.

$$\sqrt{(y4-y3)^2} - \sqrt{(y2-y1)^2} = 0 \qquad (3.19)$$

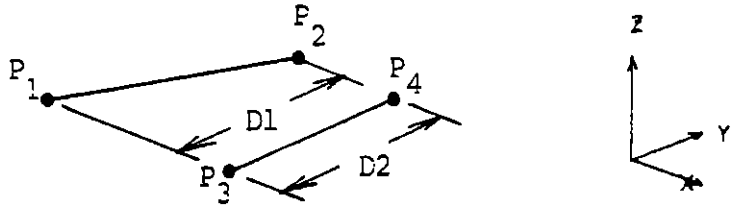


Figure 14, Equal y distance between point pairs.

13) EQUAL Z DISTANCE BETWEEN POINT PAIRS: The z distance between P1=(x1,y1,z1) and P2=(x2,y2,z2) and the z distance between P3=(x3,y3,z3) and P4=(x4,y4,z4) are constrained to be equal.

$$\sqrt{(z4-z3)^2} - \sqrt{(z2-z1)^2} = 0 \qquad (3.20)$$

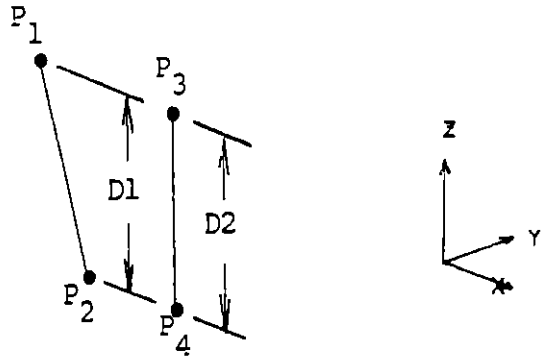


Figure 15, Equal z distance between point pairs.

14) EQUAL LINEAR DISTANCE BETWEEN POINT PAIRS: The linear distance between P1=(x1,y1,z1) and P2=(x2,y2,z2) and the linear distance between P3=(x3,y3,z3) and P4=(x4,y4,z4) are constrained to be equal.

$$\sqrt{(x4-x3)^2+(y4-y3)^2+(z4-z3)^2}-\sqrt{(x2-x1)^2+(y2-y1)^2+(z2-z1)^2}= 0 \quad (3.21)$$

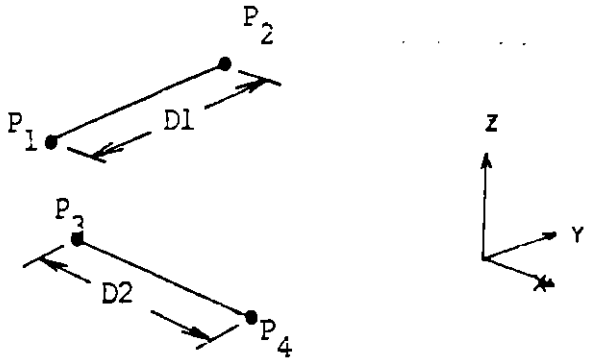


Figure 16, Equal linear distance between point pairs.

3.3 EXPLICIT CONSTRAINTS

The following three constraints are useful for dimensioning a part in the three orthographic views.

15) X DISTANCE BETWEEN TWO POINTS: This constraint fixes one point, P1=(x1,y1,z1) to be a certain distance, Dx, away from a second point, P2=(x2,y2,z2) in the x direction.

$$\sqrt{(x2 - x1)^2} - Dx = \emptyset \qquad (3.22)$$

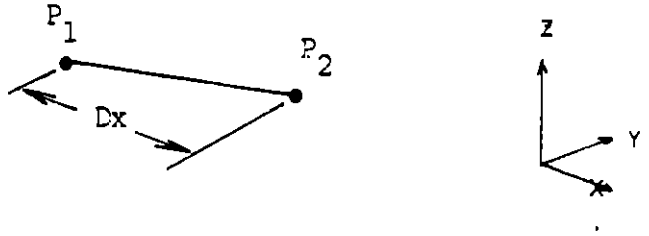


Figure 17, X distance between two points.

16) Y DISTANCE BETWEEN TWO POINTS: This constraint fixes one point, P1=(x1,y1,z1) to be a certain distance, Dy, away from a second point, P2=(x2,y2,z2) in the y direction.

$$\sqrt{(y2 - y1)^2} - Dy = \emptyset \qquad (3.23)$$

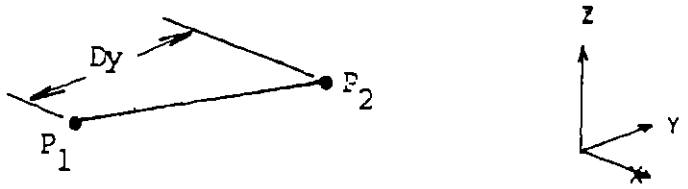


Figure 18, Y distance between two points.

17) Z DISTANCE BETWEEN TWO POINTS: This constraint fixes one point, P1=(x1,y1,z1) to be a certain distance, Dz, away from a second point, P2=(x2,y2,z2) in the z direction.

$$\sqrt{(z2 - z1)^2} - Dz = \emptyset \qquad (3.24)$$

Figure 19, Z distance between two points.

18) LINEAR DISTANCE BETWEEN TWO POINTS: This constraint fixes one point, P1=(x1,y1,z1) to be a certain linear distance, D, away from a second point, P2=(x2,y2,z2).

$$\sqrt{(x2-x1)^2 + (y2-y1)^2 + (x2-x1)^2} - D = \emptyset \quad (3.25)$$

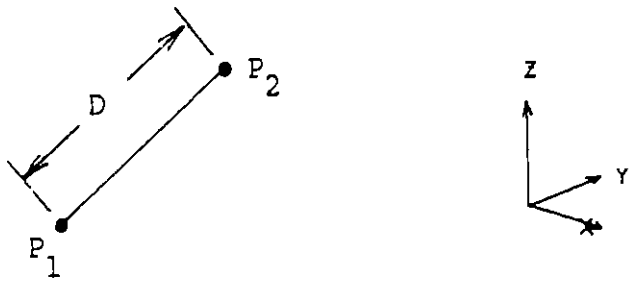


Figure 20, Linear distance between two points.

19) DISTANCE FROM A POINT TO A PLANE: This constraint fixes the distance from the plane Ax + By + Cz + D = Ø to the point, P1=(x1,y1,z1) which is not on the plane. Since P1 is not on the plane, Ax1 + By1 + Cz1 + D ≠ Ø. Let P2=(x2,y2,z2) be the projection of P1 on the given plane. Then Ax2 + By2 + Cz2 + D = Ø. Let $\underline{u}$ be the plane normal vector.

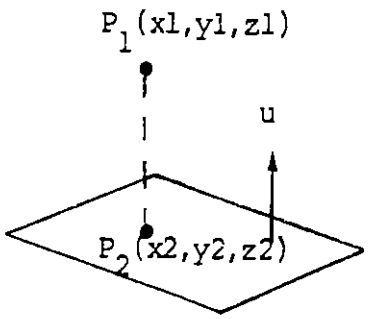


Figure 21, Distance from a point to a plane.

CASE 1: If P1 and $\underline{u}$ lie on the same side of the plane, the angle between $\underline{u}$ and $\overline{P2P1}$ is $\emptyset^{\circ}$ and

$$\frac{\underline{u} \cdot \overline{P2P1}}{|\underline{u}| \; |\overline{P2P1}|} = \cos \emptyset^{\circ}$$

$$\frac{A(x1-x2) + B(y1-y2) + C(z1-z2)}{\sqrt{A^2 + B^2 + C^2} \; |\overline{P2P1}|} = 1$$

since $D = - Ax2 - By2 - Cz2$,

$$\frac{Ax1 + By1 + Cz1 + D}{\sqrt{A^2 + B^2 + C^2} \; |\overline{P2P1}|} = 1$$

$$|\overline{P2P1}| = \frac{Ax1 + By1 + Cz1 + D}{\sqrt{A^2 + B^2 + C^2}} \qquad (3.26)$$

CASE 2: If P1 and $\underline{u}$ lie on opposite sides of the plane, the angle between $\underline{u}$ and $\overline{P2P1}$ is 180° and

$$\frac{\underline{u} \cdot \overline{P2P1}}{|\underline{u}| \; |\overline{P2P1}|} = \cos 180^{\circ}$$

$$\frac{A(x1-x2) + B(y1-y2) + C(z1-z2)}{\sqrt{A^2 + B^2 + C^2}\ \ |\overline{P2P1}|} = -1$$

replacing D by - Ax2 - By2 - Cz2,

$$\frac{Ax1 + By1 + Cz1 + D}{\sqrt{A^2 + B^2 + C^2}\ |\overline{P2P1}|} = -1$$

$$- |\overline{P2P1}| = \frac{Ax1 + By1 + Cz1 + D}{\sqrt{A^2 + B^2 + C^2}} \qquad (3.27)$$

So to constrain the distance, R, from the point P1 to the plane Ax + By + Cz + D = Ø,

$$\frac{Ax1 + By1 + Cz1 + D}{\pm\sqrt{A^2 + B^2 + C^2}} - R = Ø \qquad (3.28)$$

where the sign before the radical is opposite to that of D.

2Ø) DISTANCE FROM A POINT TO A LINE: This constraint fixes the perpendicular distancce from a point, P1=(x1,y1,z1), to a line, P2P3.

Let $\underline{v}$ be the vector from P2 to P1

$$\underline{v} = (x1-x2)\hat{i} + (y1-y2)\hat{j} + (z1-z2)\hat{k}$$

Let $\hat{u}$ be the unit vector from P2 to P3

$$\hat{u} = \frac{(x3-x2)}{|P2P3|}\hat{i} + \frac{(y3-y2)}{|P2P3|}\hat{j} + \frac{(z3-z2)}{|P2P3|}\hat{k}$$

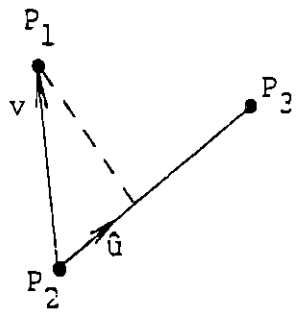


Figure 22, Distance from a point to a line.

The magnitude of the vector product of $\hat{u}$ and $\underline{v}$ is equal to the magnitude of $\hat{u}$ times the magnitude of $\underline{v}$ times the sine of the angle between them.

$$|\hat{u} \times \underline{v}| = |\hat{u}||\underline{v}| \sin\theta$$

Since $\hat{u}$ is a unit vector, $|\hat{u}| = 1$

$$|\hat{u} \times \underline{v}| = |\underline{v}| \sin\theta$$

Since $|\underline{v}| \sin\theta$ is the required perpendicular distance from point P1 to the line, we have,

$$|\hat{u} \times \underline{v}| = DIST$$

$$|\hat{u} \times \underline{v}| = \begin{vmatrix} u2 & u3 \\ v2 & v3 \end{vmatrix}\hat{i} + \begin{vmatrix} u3 & u1 \\ v3 & v1 \end{vmatrix}\hat{j} + \begin{vmatrix} u1 & u2 \\ v1 & v2 \end{vmatrix}\hat{k}$$

$$= (u2v3-u3v2)\hat{i} + (u3v1-u1v3)\hat{j} + (u1v2-u2v1)\hat{k}$$

$$DIST=|\hat{u} \times \underline{v}|=\sqrt{(u2v3-u3v2)^2+(u2v1-u1v2)^2+(u1v2-u2v1)^2} \quad (3.29)$$

where

$$v1 = x1 - x2, \quad v2 = y1 - y2, \quad v3 = z1 - z2$$

$$u1 = \frac{x3 - x2}{|P2P3|}, \quad u2 = \frac{y3 - y2}{|P2P3|}, \quad u3 = \frac{z3 - z2}{|P2P3|}$$

$$|P2P3| = \sqrt{(x3-x2)^2 + (y3-y2)^2 + (z3-z2)^2}$$

21) DISTANCE BETWEEN PARALLEL LINES: This constraint constrains two lines to be parallel and the distance from a point to a line.

22) DISTANCE FROM A LINE TO A PLANE: This constraint constrains the line to be parallel to the plane and the distance from one point of the line to the plane.

23) DISTANCE BETWEEN TWO PLANES: This constraint constrains the two planes to be parallel and the distance from a point to a plane contributing a total of three items of information.

24) ANGLE BETWEEN TWO LINES: The cosine of the angle between two lines can be found by summing the products of the respective direction cosines of the two vectors on the lines, [x1-x2,y1-y2,z1-z2] and [x3-x2,y3-y2,z3-z2].

$$\frac{(x1-x2)(x3-x2) + (y1-y2)(y3-y2) + (z1-z2)(z3-z2)}{\sqrt{(x1-x2)^2+(y1-y2)^2+(z1-z2)^2}\sqrt{(x3-x2)^2+(y3-y2)^2+(z3-z2)^2}} - \cos\theta = 0 \qquad (3.30)$$

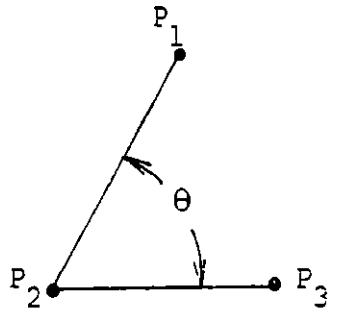


Figure 23, Angle between two lines.

25) ANGLE BETWEEN A LINE AND A PLANE: The angle between a line and the plane Ax + By + Cz + D = Ø is the complement of the angle between the vector on the line, [x2-x1,y2-y1,z2-z1], and the normal vector of the plane, [A,B,C].

$$\frac{A(x2-x1) + B(y2-y1) + C(z2-z1)}{\sqrt{(x2-x1)^2+(y2-y1)^2+(z2-z1)^2}\sqrt{A^2+B^2+C^2}} - \cos(90-\theta) = 0 \quad (3.31)$$

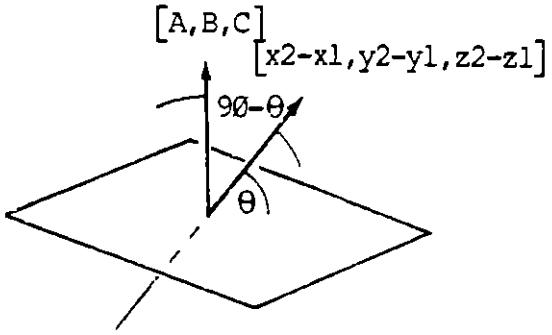


Figure 24, Angle between a line and a plane.

26) ANGLE BETWEEN PLANES: This constraint constrains two planes, A1x + B1y + C1z + D1 = Ø and A2x + B2y + C2z + D2 = Ø, to have a certain angular separation, θ . The angle between two planes can be found by taking the angle between the normal vectors of the two planes, [A1,B1,C1] and [A2,B2,C2]. Since the cosine of the angle,

θ , between two vectors is equal to the sum of the products of the corresponding direction cosines of the two vectors,

$$\frac{A1A2 + B1B2 + C1C2}{\sqrt{A1^2+B1^2+C1^2}\sqrt{A2^2+B2^2+C2^2}} - \cos(180 - \theta) = 0 \qquad (3.22)$$

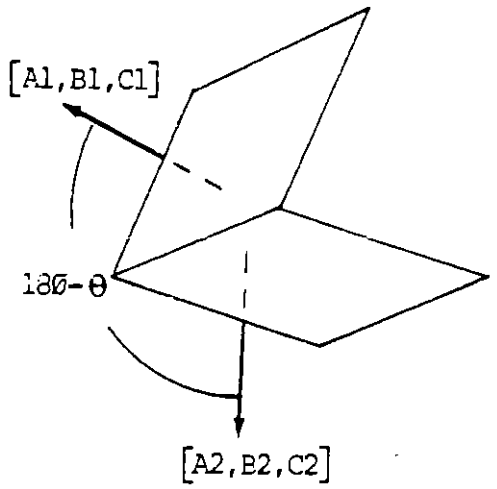


Figure 25, Angle between planes.

Note that the normal vectors are defined to point outward from the planes of the object.

This constraint should not be confused with the concept of the angular position of one plane with respect to another. This concept is useful for the process of creating a dimensioning scheme. Two items of information must be given to fix the angular position of one plane with respect to another. This concept can be best illustrated by an example. Figure 26 shows two intersecting planes. Plane 2 can be thought of as a base plane. Figure 27 is a view

which gives the edge view of plane 1. Clearly, the angle, θ, must be fixed.

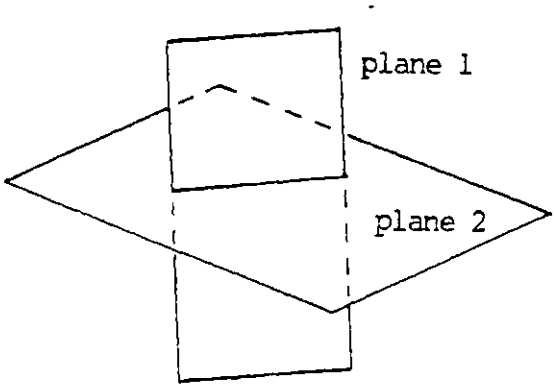


Figure 26, Two intersecting planes.

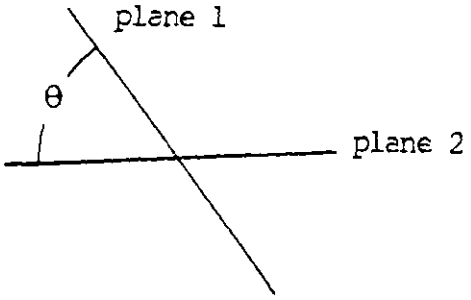


Figure 27, Edge view of two intersecting planes.

The fixing of this angle corresponds to the constraint of the angle between two planes presented previously. Figure 28 shows a top view of the two intersecting planes. It can be seen that plane 1 can rotate with respect to plane 2 tracing out the angle, α, and still keep the angle, θ,

fixed. Therefore, to fix the angular position of one plane with respect to another, both α and θ must be fixed.

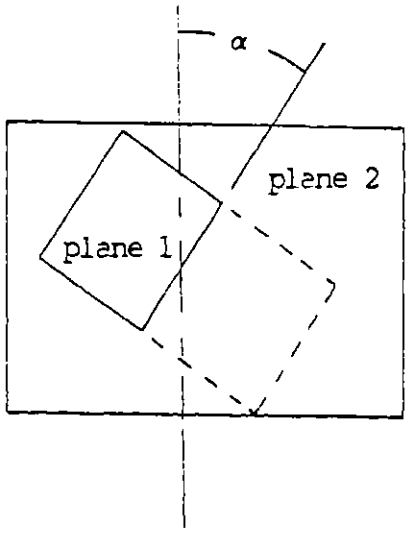


Figure 28, Top view of two intersecting planes.

3.4 CONSTRAINING CURVED SURFACES

Presently, the types of curved surfaces represented are limited to cylindrical, spherical, and conical surfaces. The technique used to constrain these surfaces is to constrain the defining points and parameters of each surface. From these parameters and defining points, the implicit or parametric equations of each surface can be easily derived for mass property and surface intersection calculations. The defining points and parameters for some

typical surfaces are shown in Figures 29 - 35.

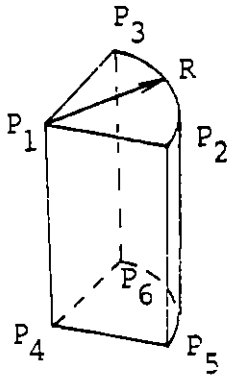


Figure 29, Cylindrical round: defined by 6 points.

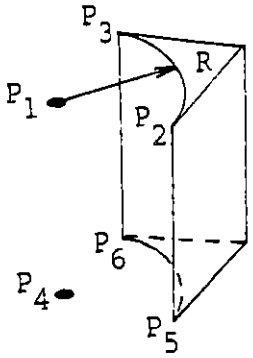


Figure 30, Cylindrical fillet: defined by 6 points.

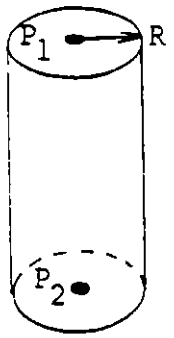


Figure 31, Cylindrical solid: defined by 2 points and 1 radius.

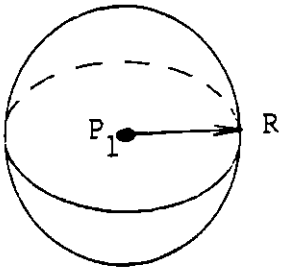


Figure 32, Spherical solid: defined by 1 point and 1 radius.

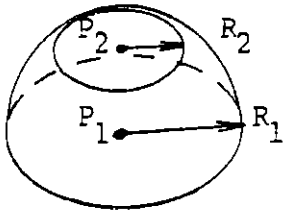


Figure 33, Partial sphere: defined by 2 points and 2 radii.

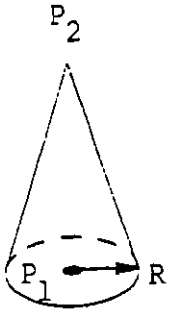


Figure 34, Right conical solid: defined by 2 points and 1 radius.

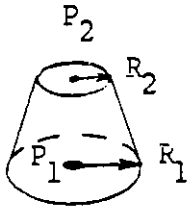


Figure 35, Right conical section: defined by 2 points and two radii.

The list of constraints for curved surfaces are given in Table 2. In deriving the mathematical relationships for the curved surface constraints, no new types of relationships need to be added to the existing set of three-dimensional constraints. A radius constraint can be specified in two ways. The first is a constraint of the linear distance between two points (Figure 36), the center and a second point on the defining surface. The second method of specifying a radius constraint is to make the radius a variable and set its value equal to some constant.

A diameter constraint is just a special form of the radius constraint and can be handled similarly.

LIST OF CONSTRAINTS FOR CURVED SURFACES

1. Radius.
2. Diameter.
3. Tangency.

Table 2, Constraints for curved surfaces.

A tangency constraint between a cylindrical round and a plane can be described by a constraint of perpendicular planes (Figure 37).

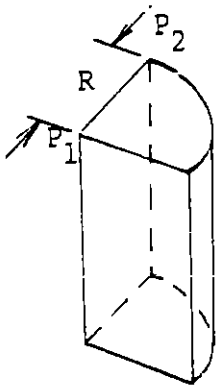


Figure 36, Radius constraint.

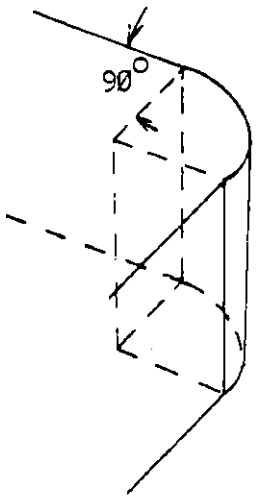


Figure 37, Tangency constraint.

4.Ø ASSIGNMENT OF CONSTRAINTS

Currently, the constraints are assigned by reading from the constraint data file. The constraint data file contains n constraint equations each with an identifying code and the entities constrained. This constraint file is to be created by the designer of the part.

In order for the DIMENSION system to become useful as a design tool, more efficient methods must to be developed for the assignment of constraints. Three different approaches are possible.

The first is an interactive approach where the designer of the part manually assigns constraints to the part. To assign a constraint, the designer would simply select a constraint from a menu and picks the points to be constrained with a data tablet and digitizing pen. The part should be capable of being rotated about the three axes and hidden lines removed. Different views could be generated to assist in constraint assignment.

There are a number of problems accompanying this approach to assign constraints. The process would be tedius and time consuming for parts which require a large number of constraints. The process is also error prone. The user could easily choose a redundant dimensioning scheme.

An alternate approach is an automatic constraining scheme. A set of rules and heuristics for constraint assignment could be established. Given the topology of a part, the computer could then automatically generate all the constraints.

A face finding algorithm can be used to constrain points to be coplanar. One such algorithm is given by Congdon [3]. Examples of other possible rules are: constrain the first point entered in rigid body translation, constrain all lines and planes that are within a specified tolerance of being parallel to the x-y plane to be horizontal, constrain two planes which are within a certain tolerance of 90 degrees to be perpendicular.

One technique for automatically generating the explicit constraints of a polyhedra is to group the faces of the part into sets of parallel planes. Within each parallel plane

set, a datum plane is chosen. All other planes in a set can be constrained by linear distance constraints to be a certain distance from the specified datum plane. Sets of parallel planes can be constrained to have a specific angular separation.

The problem with a fully automatic dimensioning scheme is that the dimensioning scheme should reflect the function of a part. For any given part, one dimensioning scheme may be chosen for one application and another for a different application. A fully automatic dimensioning scheme cannot distinguish between the different possibilities.

Some combination of the two approaches should be taken. Those constraints which will tend to be invariant for different dimensioning schemes of a part should be automatically generated by the computer. Those constraints which reflect the function of a part should be assigned by the designer.

Examples of constraints which tend to be invariant with different constraining schemes are: four points coplanar, rigid body translation, rigid body rotation, parallel planes, perpendicular planes, diameter and radius. Examples

of constraints which reflect the function of a part are: distance between planes, points and lines.

5.Ø NUMERICAL METHODS

The defining points of a three-dimensional object give rise to 3N constraint equations which must be solved simultaneously for the 3N variables which are the x,y, and z coordinates of the points. The method used in the DIMENSION system is the Newton-Raphson method (see Appendix C). For each step of the Newton-Raphson method, a linear system of equations of the form $A\underline{x} = \underline{b}$ must be solved. The matrix, A, contains the partial derivatives of the constraint equations with respect to each coordinate (also called the Jacobian). The vector, $\underline{x}$, contains the displacements or changes in coordinates. The vector $\underline{b}$, contains the values (also called the residuals) of the constraint equations.

In using the Newton-Raphson method, problems of roundoff and ill-conditioning must be controlled. Ill-conditioning is a measure of numeric instability in the presence of roundoff. This effect is purely numerical since in principle, a solution to $A\ \underline{x} = \underline{b}$ can always be found by inverting A. If the determinant of A is near zero, then the solution will be highly sensitive to the numerical noise generated by roundoff.

The inverse matrix is never explicitly computed. One method by which a linear system of equations may be solved is a variant of Gaussian elimination called Doolittle's method with partial pivoting (see Appendix D). The use of partial pivoting tends to control the error due to roundoff. In Doolittle's method, there exists a possibility of one of the diagonal elements having a zero value causing a zero divide. If the last diagonal element of the matrix, A, vanishes, the determinant of the matrix is zero. This means that the matrix is singular and can neither be inverted nor can the linear equations be solved uniquely. If an earlier diagonal element vanishes, it does not necessarily mean that A is singular. It means only that a leading submatrix of A has a zero determinant. Also, dividing by a small pivot element can give rise to large errors in the computed results. Therefore, the largest element in a column is chosen as the pivot element. The corresponding row pointers are interchanged.

Other methods of minimizing roundoff in the numerical methods are to accumulate the vector inner products in double precision in Doolittle's method and by proper scaling of the Jacobian.

The constraint equations are written in such a way that unnecessary powers of terms are eliminated. The partials of the constraint equations, then, should not produce elements which are an order of magnitude larger or smaller than other elements in the Jacobian. For example, the constraint equation for the distance between two points can be written in two ways:

$$(x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2 - D^2 = \emptyset \qquad (5.1)$$

$$\sqrt{(x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2} - D = \emptyset \qquad (5.2)$$

The second representation is chosen to be consistent with the representation of other constraint equations.

6.Ø COMPUTATIONAL EFFICIENCY

6.1 GENERAL

In three dimensions, the number, type, and complexity of the constraint equations increase. The matrices also become significantly larger. Three methods were used to increase the computational efficiency of the numerical methods. The first uses the modified Newton-Raphson method instead of the standard Newton-Raphson method. The second uses sparse matrix methods to solve the set of linear

equations in each iteration of the modified Newton-Raphson method instead of Doolittle's method. The third involves segmentation of the Jacobian matrix. Without segmentation of the Jacobian, n equations in n coordinates must be solved given any dimension change. It is the purpose of the segmentation method to reduce the number of equations and coordinates to a smaller subset.

6.2 MODIFIED NEWTON-RAPHSON METHOD

The modified Newton-Raphson method eliminates the need to invert the Jacobian at each iteration. This benefit carries with it the cost of slower convergence than the Newton-Raphson method.

For the Newton-Raphson method:

$$\underline{X}n+1 = \underline{X}n - J^{-1}(\underline{X}n)\ f(\underline{X}n) \qquad (6.1)$$

For the modified Newton-Raphson method:

$$\underline{X}n+1 = \underline{X}n - J^{-1}(\underline{X}o)\ f(\underline{X}n) \qquad (6.2)$$

where J, the Jacobian matrix and its inverse, J^{-1} is computed once at $\underline{X}o$.

The geometric interpretation of the modified Newton-Raphson method in two dimensions is that instead of computing the tangent to the curve at each iteration, the same slope derived in the first tangent is used. Figure 38 illustrates this.

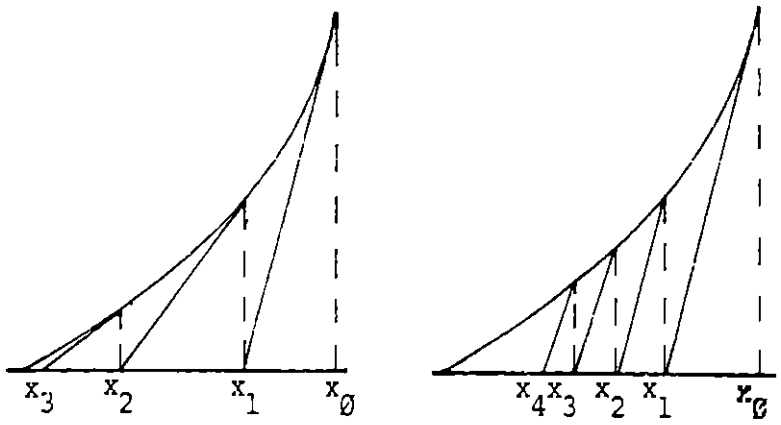


Figure 38, Geometric interpretation of the Newton-Raphson method and the modified Newton-Raphson method.

Two factors determine the computational time which would be saved by using the modified Newton-Raphson method instead of the Newton-Raphson method: the time saved in computing each iteration, and the number of iterations necessary for convergence. The first factor was evaluated by comparing the time saved in computing each iteration after the first since the first iteration is the same in both methods.

Figure 39 shows a plot of the time required to compute each iteration after the first for different numbers of equations. Figure 40 shows a plot of the time required for a set of 58 equations in 58 unknowns as a function of the number of iterations performed. From Figure 39, it can be seen that as the number of equations increases, modified Newton-Raphson becomes more efficient. From Figure 40, it can be seen that as the number of iterations increase, modified Newton-Raphson also becomes more efficient.

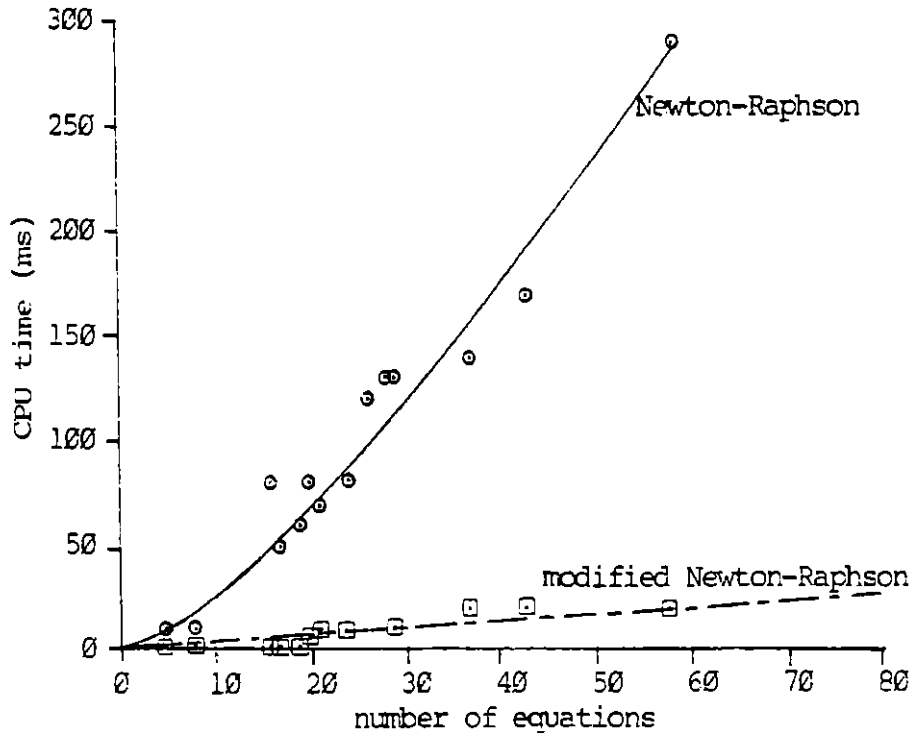


Figure 39, Plot of the CPU time required to compute each iteration after the first vs the number of equations.

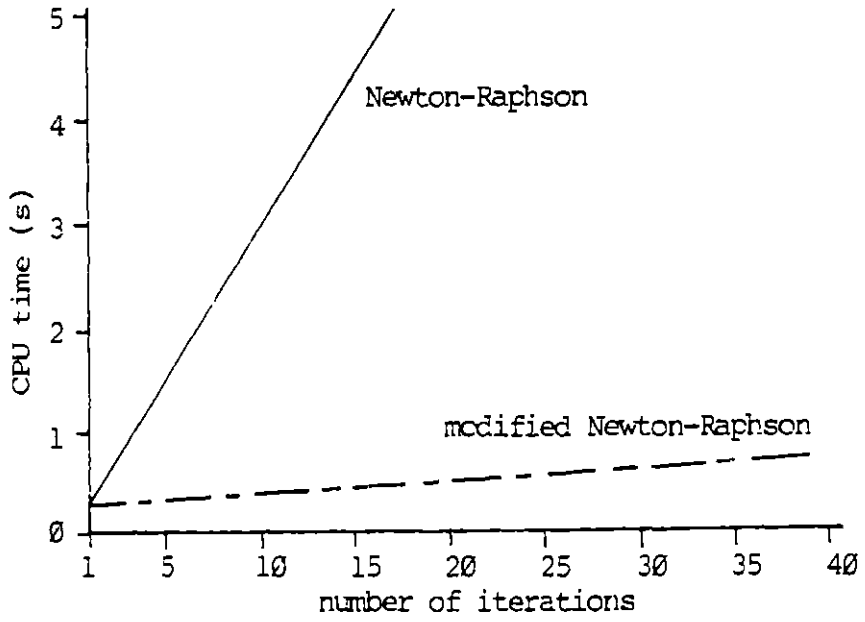


Figure 40, Plot of CPU time for a set of 58 equations in 58 unknowns vs the number of iterations performed (extrapolated from a single iteration).

It is more difficult to directly compare the number of iterations necessary for convergence for the two methods. The factors which affect the rate of convergence are the starting value, the particular set of equations used, and roundoff errors. On the different sets of equations tested, the worse case encountered was that twice the number of iterations were required for convergence using the modified Newton-Raphson method. By looking at Figures 39 and 40,

even if the worse case conditions were encountered, using the modified Newton-Raphson method would still be more efficient than the standard Newton-Raphson method.

A quantitative estimate of the savings in (single user VAX 11/78Ø) CPU time can be obtained from Figures 39 and 4Ø. Assuming that the slopes of the plotted curves are linear, Equation (6.3) gives the CPU time in milliseconds as a function of the number of equations and the number of iterations performed by the Newton-Raphson method. Equation (6.4) gives the CPU time in milliseconds as a function of the number of equations and the number of iterations performed by the modified Newton-Raphson method. For the worse case examined, Imnr was no more than 2Inr. The convergence criterion in both cases was that the displacements of all the coordinate points of the part be sufficiently small, i.e. $|dxi| < \delta \quad i = 1,2,\ldots,n, \quad \delta = .\text{Ø}\text{Ø}5$.

$$tnr = 4.98\ E\ Inr \qquad (6.3)$$

where

tnr = CPU time in ms using the Newton-Raphson method

E = number of equations

Inr = number of iterations required using the Newton-Raphson method

$$tmnr = 4.98\ E + [.34\ E(Imnr - 1)] \qquad (6.4)$$

where

tmnr = CPU time in ms using the modified Newton-Raphson method

E = number of equations

Imnr = number of iterations required using the modified Newton-Raphson method

Table 3 shows the percentage savings in CPU time in using the modified Newton-Raphson method instead of the Newton-Raphson method for two sample dimension alterations.

E	Inr	Imnr	tnr(ms)	tmnr(ms)	% savings
28	2	4	557.8	168.Ø	69.9
43	4	5	1Ø7Ø.7	272.6	74.5

Table 3, Percent savings in CPU time in using the modified Newton-Raphson method for two sample dimension alterations.

6.3 SPARSE ELIMINATION

In the systems of constraint equations used in variational geometry the Jacobian matrix often has a low percentage of non-zero elements. The sparseness of a matrix can be described by a number of factors, the most common being the sparsity factor, Sk, where Sk = number of non-zeros / n^2 . For small Sk, it is advantageous to use sparse matrix methods to solve the system of linear equations, $A\underline{x} = \underline{b}$ (see Appendix E).

The main idea underlying sparse matrix methods is to permute the original matrix into lower block triangular form by a set of row and column permutations:

$$P'AQ' = \begin{bmatrix} A11 & & & & \\ A21 & A22 & & \emptyset & \\ A31 & A32 & A33 & & \\ . & . & . & Aii & \\ Ak1 & & . & . & Akk \end{bmatrix} \qquad (6.5)$$

where P' and Q' are identity matrices upon which suitable row and column interchanges have been performed.

We can then perform Gaussian elimination on the subsystems, Aii, i=1,2,...,k where $\emptyset \leq k \leq n$:

$$PiAiiQi = LiUi \quad i=1,2,...,k \qquad (6.6)$$

where Li is unit lower triangular and Ui is upper triangular. By combining with permutations P' and Q' we have:

$$PAQ = \begin{bmatrix} L1U1 & & & & \\ A21 & L2U2 & & \emptyset & \\ A31 & A32 & L3U3 & & \\ . & . & . & LiUi & \\ Ak1 & Ak2 & . & . & LkUk \end{bmatrix} \qquad (6.7)$$

The growth of non-zeros are restricted to the diagonal blocks. The maximum order of the system to be considered at any one time is reduced to the order of the largest submatrix, Aii. By partitioning $\underline{x}$ and $\underline{b}$ in a manner similar to A, the overall system can be solved by block forward substitution using both the diagonal and subdiagonal blocks:

$$A11x1 = b1$$

$$A22x2 = b2 - A21x1 \qquad (6.8)$$

$$Aiixi = bi - \sum_{q=1}^{i-1} Aiqxq \quad i=1,2,\ldots,k$$

In the current implementation of the DIMENSION system, HARWELL's subroutine package is used [6]. The time required to solve the linear system, $A\underline{x} = \underline{b}$ using Doolittle's method is of $O(n^3)$. The time required to solve the same system using the block triangularization method described above is of $O(m^2/n)$, where m is the number of non-zeros in the matrix A. In the system of equations used in variational geometry, $m = O(n)$. Therefore, total time required for the sparse matrix methods is of $O(n)$. Figure 41 compares the time required to solve the set of equations $A\underline{x} = \underline{b}$ with Doolittle's method versus block triangularization and decomposition in the sparse matrix routines.

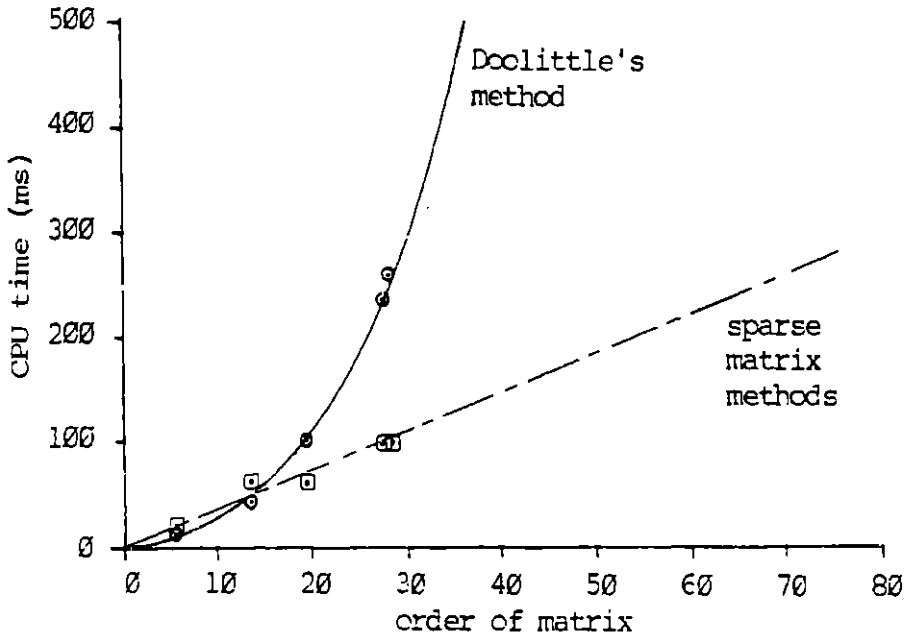


Figure 41, Comparison of Doolittle's method and block triangularization/decomposition

The storage requirement for the sparse elimination method is of O(n) as compared to the $O(n^2)$ storage requirement of the standard Doolittle's method.

Instead of storing the full n x n Jacobian matrix, only non-zero elements are stored. In the current imlementation, three linear arrays replace the n x n array required to store the Jacobian. A real array stores non-zero values of the Jacobian. Two integer*2 arrays store the row and column indices of the corresponding non-zero elements.

Alternate storage methods could furthur reduce the storage requirements. One such method stores the non-zero elements by row(column) order form. In this scheme, a real array stores the values of the non-zeros. An integer*2 array stores the starting address of the first non-zero in each row(column). An integer*2 array stores the column(row) indices of the non-zero elements. The current storage method is chosen for ease of interface to the HARWELL sparse subroutine package.

The upper bound on the number of non-zeros in the Jacobian matrix is 18n. This is because each constraint equation currently constrains a maximum of six points for a total of 18 coordinates. Although 18n is the maximum bound on m, m is rarely, if ever, this large. A more reasonable estimate of m is $m \approx 5n$. Figure 42 compares conventional n^2 storage requirement with the current sparse storage requirement for $m = 18n$ and for $m = 5n$.

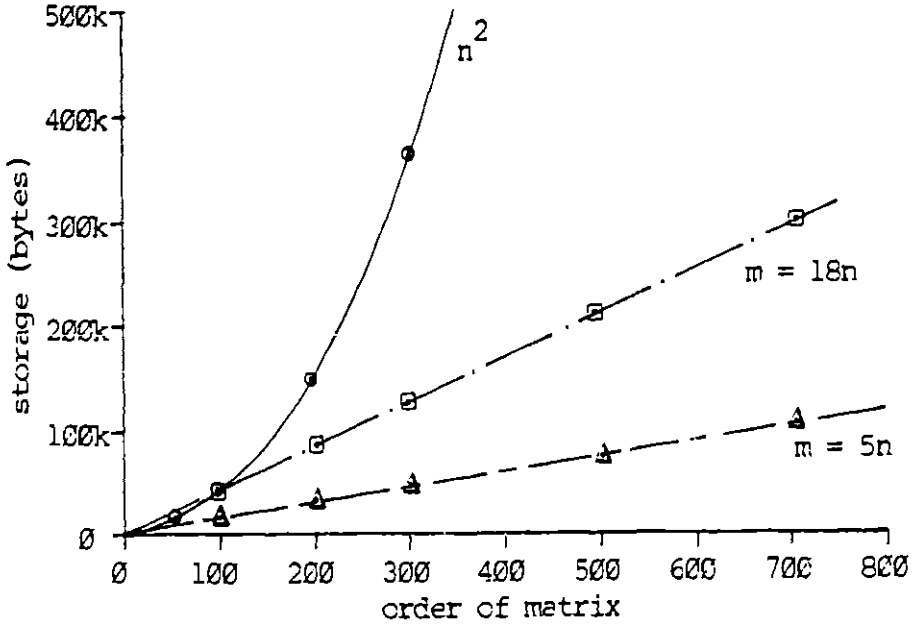


Figure 42, Plot of storage requirements vs the order of the Jacobian matrix for n^2 storage and sparse storage.

6.4 SEGMENTATION

The size of the Jacobian matrix used in the solution of the numerical methods increases as more points are used to define an object. In three dimensions, the matrix contains $9N^2$ elements for N defining points of an object. A closer inspection of the constraint equations and coordinate points show that for a given dimension change, usually not all the

coordinates will change and not all the constraint equations will be affected. Given a dimension change, there exists a subset of the total coordinate points which will change and a subset of the total constraint equations which are required to solve for the changed coordinate points. Almost always, these subsets are less than the total set. The magnitude of this subset depends directly on the constraining scheme chosen.

In order to make the DIMENSION system more time efficient, a method of segmenting the total set of constraint equations and coordinate points was developed. The method is called the propagation method.

6.4.1 JACOBIAN MATRIX

The Jacobian matrix, A, is a matrix of partial derivatives of functions fi, i=1,2,...,n, with respect to variables xj, j=1,2,...,n where fi are the constraint equations and xi are the coordinates of the defining points of an object. An element of the Jacobian, aij, represents the partial derivative of constraint equation i with respect to coordinate j evaluated at the current coordinate points.

Geometrically, a constraint equation describes a surface in n-dimensional space, and the partial derivative of a constraint with respect to a coordinate can be considered the slope of the tangent hyperplane to the constraint surface in the direction of that coordinate.

There are two situations in which the Jacobian may have a zero element, aij = Ø. First, coordinate j does not appear in constraint equation i. Second, coordinate j does appear in equation i, but a small change in coordinate j has no effect on equation i.

The interpretation for aij ≠ Ø is that a change in coordinate j will affect constraint equation i. Geometrically, this can be regarded as a non-zero slope of the tangent hyperplane to the constraint, fi, in the direction of coordinate xj.

The zero-non zero structure of the Jacobian matrix was utilized in the process of segmentation. The Jacobian was converted into a binary matrix, that is, a matrix with only zeros and ones. Associated with A = [aij] is B = [bij] where bij = Ø if aij = Ø and bij = 1 if aij≠ Ø.

6.4.2 GRAPH THEORETIC MODEL

To solve the problem of segmentation, a graph theoretic model was used. Before proceeding with a discussion of the solution to the segmentation problem, some definitions will be given.

A graph, G(V,E) is a structure which consists of a set of vertices V = {v1,v2,...,vn} and a set of edges E = {e1,e2,...,en}. Note that the terms vertex and edge as used in this discussion are not to be confused with the terms vertex and edge used previously to define the topology of a part.

Each edge connects a pair of vertices u and v. Edge e is then said to be incident on vertices u and v. A graph is directed if the vertex pairs (u,v) associated with each edge, e, is an ordered pair. Edge e is then said to be directed from vertex u to vertex v. A graph is undirected if the end vertices of all edges are unordered and the edges have no direction. Edges are said to be parallel if they have the same pair of end vertices. An edge is a self-loop if it begins and ends on the same vertex. A simple graph is one in which there are no parallel edges or self-loops.

On paper, vertices are normally represented as dots or circles and edges as lines. Directed edges are represented as arrowed lines. In a computer, a graph, G(V,E) with n vertices can be represented by an n x n matrix A = [aij], where aij = 1 if there exists an edge connecting vertex i to vertex j in G. Otherwise aij = Ø. This matrix is called the adjacency matrix. The adjacency matrix requires $O(n^2)$ storage. Often times, the adjacency matrix is sparse. In these cases, a sparse storage method can be used.

A graph, G(V,E) is bipartite if the set of vertices, V can be partitioned into two sets, R and S such that each edge of G joins a vertex in R with a vertex in S, where $V = R \cup S$, $R \cap S = \emptyset$. A matching, M of G =(R,S,E) occurs if every vertex is incident to at most one edge of M. Each edge, $\{ri,sj\} \in E$, $ri \in R$, $sj \in S$, has one end vertex in R and one in S. A maximum matching of a bipartite graph is a maximum number of edges, no two of which meet at a common vertex. A matching, M is called complete if |M| = |R|, where |M| represents the members of set M and |R| represents the members of set R.

Let B be a binary matrix of order n with entries bij. Let E be the set of edges. Let R be the set of constraint equations. Let S be the set of coordinates. Associated with matrix, B = [bij] is a bipartite graph, G, with 2n vertices, {ri} $\in$ R and {sj} $\in$ S, i,j = 1,2,...,n, in which ri and sj are joined by edge {ri,sj} $\in$ E if and only if bij = 1 (see Figure 43).

$$A = \begin{bmatrix} 0 & 10 & 9 \\ 2 & 4 & 0 \\ 1 & 0 & 7 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

G =

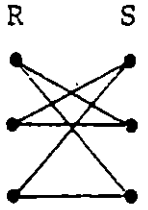


Figure 43, Bipartite graph model of a matrix.

6.4.3 PROPAGATION METHOD

The purpose and goal of the propagation method is to find a subset of the total number of variables and a subset of the total number of constraint equations which need to be solved for given any dimension change. The propagation method does this by exploiting the relationships between the constraint equations and the coordinates of the points of the part. This process can be divided into two parts. The

first part involves finding a matching between each constraint equation and a coordinate such that each equation has a matched coordinate and each coordinate appears as a matched coordinate for one equation only. The second part is a chain type propagation method whereby if an equation is changed, its corresponding matched coordinate will change. This change in a coordinate will affect other equations which in turn will change other coordinates. The process propagates in this way until the subset of equations and coordinates which need to be solved is found.

In order for a constraining scheme to be permissible, i.e. no redundant equations, each variable must be constrained by at least one equation. Likewise, each constraint equation must constrain at least one variable. Therefore, in order for a permissible constraining scheme to exist, there must be at least one matching set in which each equation is matched with a unique variable. This is a necessary and sufficient condition for a valid constraining scheme. Given this matching set, if any constraint equation (dimension) is altered, we know that its corresponding matched variable will change.

If we model the set of constraint equations and the set of coordinates as the two sets of vertices in a bipartite graph, the first step of the propagation method, then, is a process by which a complete matching in a bipartite graph is found.

Although given different names and used in different fields, the concept of finding a complete matching in a bipartite graph has been well documented in literature.

Hall [14] gives an algorithm for finding a set of distinct representatives in the field of combinatorics. Steward [28] uses a chaining algorithm to obtain an output set in the analysis of systems of linear equations. Hopcroft and Karp [19] gives an algorithm for finding maximum matchings in bipartite graphs. Sussman, et al [4],[27] uses a one step deduction method of constraint propagation in the analysis of electrical networks in artificial intelligence. Gustavson [12] uses a depth first search method together with heuristic techniques to find a maximum assignment. Duff [5],[6],[7] finds a maximun traversal by using a depth first search method to put only non-zeros on the diagonal of a square matrix.

The concept of a complete matching may be illustrated by an example. Given the three equations in three unknowns:

$$f1 = 2x1 = 4 \quad (6.5)$$

$$f2 = x1 + x2 = 5 \quad (6.6)$$

$$f3 = 3x1 + 5x2 - x3 = 17 \quad (6.7)$$

Figure 44 shows its bipartite graph representation and figure 45 shows the complete matching for the bipartite graph.

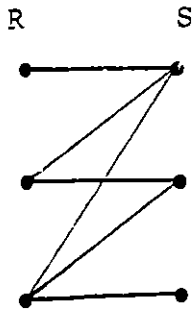


Figure 44, Bipartite graph representation.

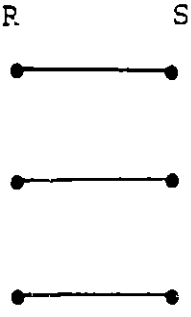


Figure 45, Complete matching.

The variable that is matched with Equation (6.5) is that variable for which Equation (6.5) is to be solved. The variable in this case is x1. The variable which is matched with Equation (6.6) is x2 since this is the variable for which Equation (6.6) is solved. In the same way, variable x3 is matched with Equation (6.7). In this case, the complete matching is unique. In many cases, there may be more than one complete matching because of the simultaneous nature of the set of equations. A non-unique complete matching occurs when an equation cannot be used to solve for a variable uniquely. An example will illustrate. Given the three equations in three unknowns:

$$f1 = x1 + 2x2 = 8 \qquad (6.8)$$

$$f2 = x2 - x3 = 2 \qquad (6.9)$$

$$f3 = 3x1 + x3 = 10 \qquad (6.10)$$

Figure 46 shows its bipartite graph representation. Figure 47 shows its two possible complete matchings.

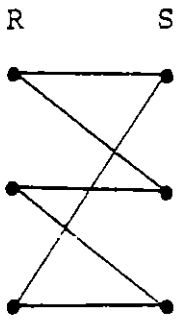


Figure 46, Bipartite graph representation.

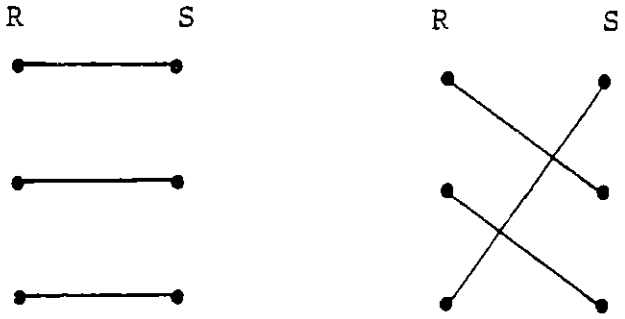


Figure 47, Two complete matchings.

If there does not exist at least one such complete matching, the set of constraint equations are redundant and the Jacobian matrix is singular. The propagation method, therefore, can detect a redundant dimensioning scheme. An example of a set of equations which does not have a complete matching is shown below:

$$f1 = x1 + x2 = 5 \qquad (6.11)$$

$$f2 = x1 = 2 \qquad (6.12)$$

$$f3 = x2 = 3 \qquad (6.13)$$

The algorithm which is used in the current implementation is a depth first search type method. A depth first search is a method of exploring all the vertices and edges of a graph with systematic backtracking. Depth first search explores an undirected graph G(V,E) by following the first edge of $v \in V$, (v,w) which is incident on v. If

vertex w has been visited previously, the search returns to v and the second edge is chosen. If vertex w has not been visited, the process is applied recursively to vertex w. If all the edges incident on vertex v have been explored in the above manner, backtrack along the edge (u,v) that led to vertex v and explore the edges incident on u. The search terminates when backtracking is attempted from the vertex at which the depth first search began.

The algorithm for finding a complete matching is as follows:

Do i = 1,n

1. For each constraint equation i, look for an edge which is incident on an unmatched coordinate j.

2. If found, match equation i with coordinate j. GO TO 6.

3. If none found, do depth first search from the current equation through the set of matched equations accessible from the current equation to find an unmatched coordinate.

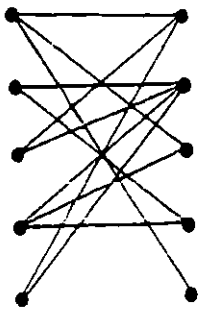
4. If search succeeds, new matching is found and the matching set is extended by one. GO TO 6.

5. If search fails, then i equations were visited and only i-1 independent coordinates were found. Matrix is singular. STOP.

6. CONTINUE

To prevent the algorithm from going into an infinite loop, each equation is marked in the backtracking step of the depth first search so that the equation will not be visited again. An example will illustrate the algorithm.

$$B = \begin{array}{c} \\ r1 \\ r2 \\ r3 \\ r4 \\ r5 \end{array} \begin{array}{c} \begin{array}{ccccc} s1 & s2 & s3 & s4 & s5 \end{array} \\ \begin{bmatrix} 1 & \emptyset & 1 & \emptyset & 1 \\ \emptyset & 1 & \emptyset & 1 & \emptyset \\ 1 & 1 & \emptyset & \emptyset & \emptyset \\ \emptyset & 1 & 1 & 1 & \emptyset \\ 1 & 1 & \emptyset & \emptyset & \emptyset \end{bmatrix} \end{array} \qquad G =$$



1. Match r1 with s1.

2. Match r2 with s2.

3. No edges of r3 incident on an unmatched coordinate, sj.

   Do depth first search from r3.

   Match r3 with s1.

   Backtrack to r1.

   Match r1 with s3.

4. Match r4 with s4.

5. No edges of r5 incident on an unmatched coordinate, sj.

   Do depth first search search from r5.

   Match r5 with s1.

   Backtrack to r3.

   No edges of r3 incident on an unmatched coordinate, sj.

Match r3 with s2.

Backtrack to r2.

No edges of r2 incident on an unmatched coordinate, sj.

Match r2 with s4.

Backtrack to r4.

No edges of r4 incident on an unmatched coordinate, sj.

Match r4 with s3.

Backtrack to r1.

Match r1 with s5.

6. Done. Complete matching: {r1,s5}
{r2,s4}
{r3,s2}
{r4,s3}
{r3,s1}

The second step of the propagation method involves using the complete matching found by the depth first search method and the binary matrix B = [bij] to obtain a subset of the total equations and coordinates which need to be solved for in the modified Newton-Raphson method.

Given a change in a dimension, a constraint equation, fi, will change. Looking in the complete matching set, the corresponding coordinate xj, which we know will change can be found. By looking in column xj of the Jacobian matrix, all non-zero values, aij, signifies that a change in

coordinate xj will cause a change in equation i. The method propagates in this way until no more new equations are found. The set of coordinates xj, and equations, fi, found by this method is the set of coordinates and equations which need to be solved for by the modified Newton-Raphson method. Since each equation has a corresponding matched coordinate the number of coordinates will always be equal to the number of equations. The algorithm is as follows:

1. For all changed equations, fi, find matched coordinate, xj.

2. For all coordinates, xj, find all equations, fi, i=1,2,...,n in which bij = 1

3. If no new equations found, STOP. Otherwise GO TO 1.

There are two cases where the use of the zero-non zero structure of the Jacobian can cause the propagation method to find less than the required number of constraint equation and coordinates given any dimension change. In these cases, the elements in the Jacobian appear as zeros when in fact then should be non-zero. Figure 48 and figure 49 illustrate the first case.

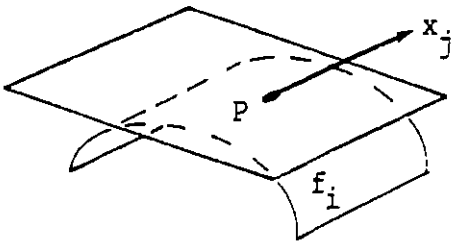


Figure 48, Case 1: Zero slope tangent plane.

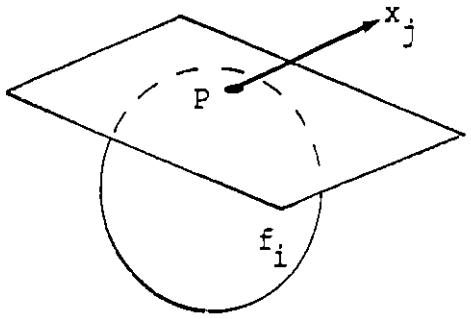


Figure 49, Case 2: Zero slope tangent plane.

In both figures, the slope of the tangent plane to the surface, fi, is zero. In figure 48, a large movement of the point P in the direction of xj will not move P off the surface, fi. In figure 49, a large movement of P in the direction of xj will move P off the surface fi.

The first case occurs when the Jacobian element dfi/dxj = Ø. A change in xj will not affect equation fi. The second case occurs when the Jacobian element dfi/dxj = Ø. A change in xj will cause equation fi to be no longer satisfied. Figure 5Ø and figure 51 illustrate how the two cases can occur in the constraint equations used in variational goemetry.

Figure 5Ø shows four points on a horizontal plane which have been constrained to be coplanar. f is, therefore, a constraint of four points, P1=(x1,y1,z1), P2=(x2,y2,z2), P3=(x3,y3,z3), and P4=(x4,y4,z4) being coplanar. df/dx3 = Ø. A change in x3 will not affect the constraint f, since all four points will still be on the same plane.

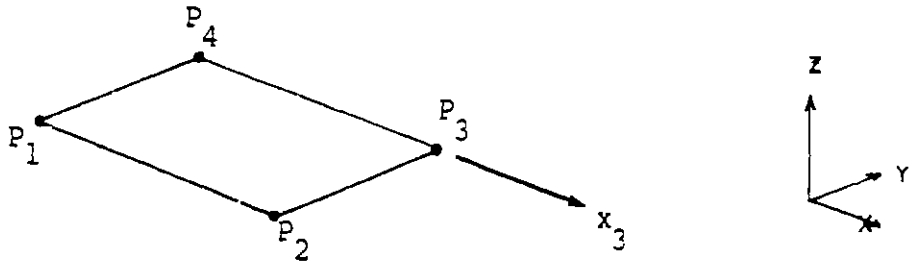


Figure 5Ø, Four points coplanar.

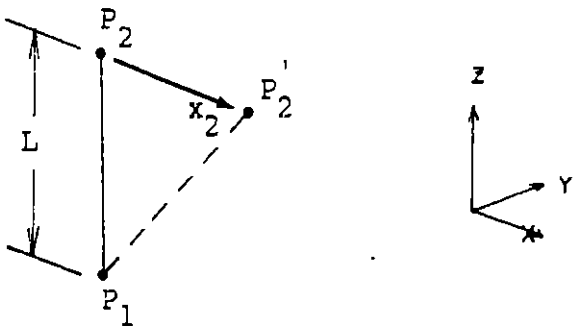


Figure 51, Linear distance between two points.

Figure 51 shows a constraint, f, of the linear distance between two points, P1=(x1,y1,z1) and P2=(x2,y2,z2) which are aligned along the z direction. df/dx2 = Ø. Given a change in x2, the constraint equation f will no longer be satisfied since L is no longer the same if z2 is kept constant.

This situation is found to occur only in this pathological case and a check can be made when forming the Jacobian elements.

The second area where the zero-non zero structure of the Jacobian can cause propagation method to find less than the total number of required constraints and coordinates is that the structure is derived only at the current geometry.

Given another geometry, the zero-non zero structure of the Jacobian can conceivably be altered. For these cases, the Jacobian matrix will need to be recomputed and step one of the propagation method reexecuted.

6.4.4 TIME AND STORAGE REQUIREMENTS

The efficiency of the propagation method depends on (1) the time required to find a complete matching in the first step of the method (2) the time required to find the subset of constraint equation and coordinates which need to be solved in the second step of the propagation method (3) the amount of reduction in the number of equations and coordinates which need to be solved.

The execution time of step one in the propagation method is of O(mn) where m is the number of non-zeros in the Jacobian matrix ($\emptyset \leq m \leq n^2$). Given our three-dimensional constraints, $m \leq 18n$. This must be true because the structure of each constraint equation is such that an individual constraint equation constrains at most six vertices. Since each vertex can have a x, y, and a z component, there can be a maximum of 18 non-zeros in any one row of the Jacobian matrix.

The process of finding a matching between an equation and a variable is of O(m). In a majority of cases, the time involved is considerably less becuse it is very rare that all the edges need to be searched to find a matching.

Since there are a total of n equations and n coordinates, the total execution time of step one is of O(mn). Therefore, the execution time of step one is of $O(n^2)$.

Step one of the propagation method is executed only once for each dimensioning scheme. Because only integer comparisons are involved, there are no computations and therefore no opportunity for numerical or roundoff errors. Figure 52 shows a plot of the time required to execute step one for different orders of the Jacobian matrix. As can be seen, the execution of the algorithm in the cases tested is much faster than the predicted worse case $O(n^2)$ time.

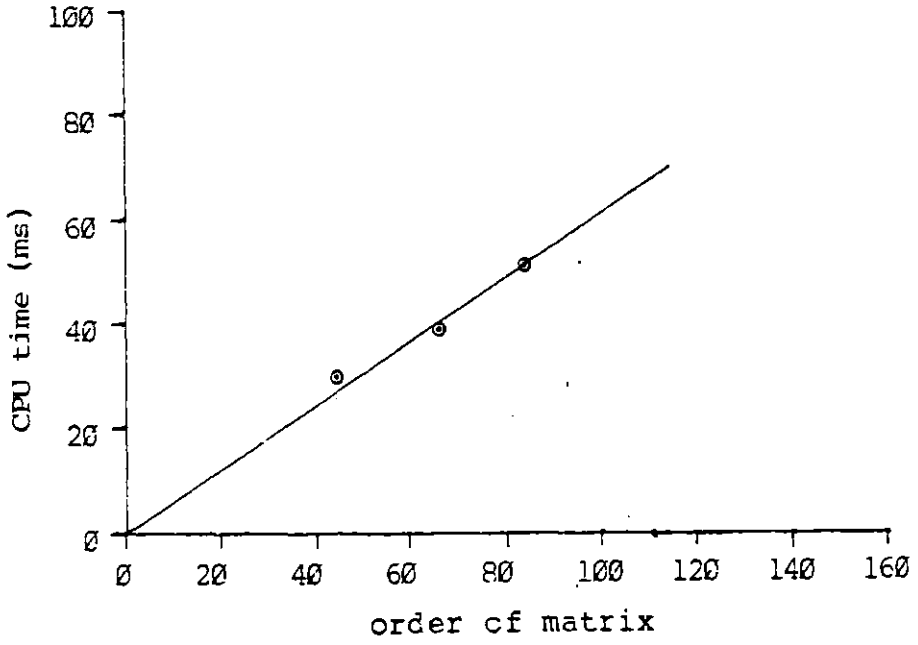


Figure 52, Plot of CPU time required for step one of the propagation method vs order of the matrix.

The execution time of the second step in the propagation method is of O(n). This step must be executed every time a new dimension is altered. The time required to execute this step depends on the number of the equations and coordinates which are found. Figure 53 shows a plot of the time required to execute step two versus the size of the subset of equations and coordinates found by this method.

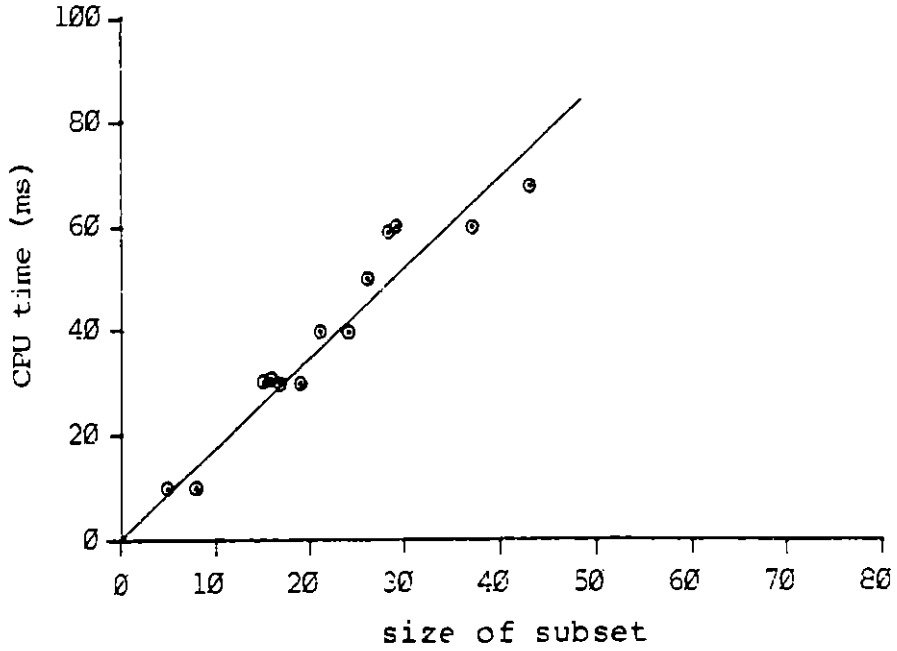


Figure 53, Plot of CPU time required for step two of the propagation method vs the size of the subset of equations and coordinates.

A general relationship for the amount of reduction of the set of equations and coordinates which need to be solved is more difficult to obtain. The amount of reduction depends on the particular combination of dimensions which are changed and the constraining scheme as a whole.

A constraining scheme can exist such that a change in a dimension causes a chain-like effect where a large number of coordinate points change. Figure 54 illustrates a dimensioning scheme which exhibits this chain-like effect.

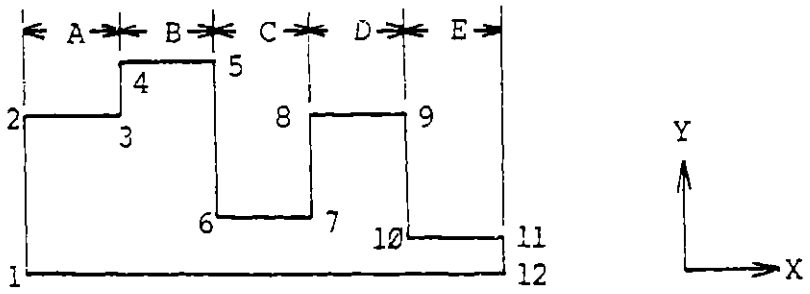


Figure 54, Consecutive dimensioning scheme.

A change in dimension A will cause points 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12 to change in the x direction. The same part may be dimensioned in another way. Figure 55 illustrates an alternate dimensioning scheme where the chain-like effect is avoided. A change in dimension A will

only change points 3 and 4 in the x direction.

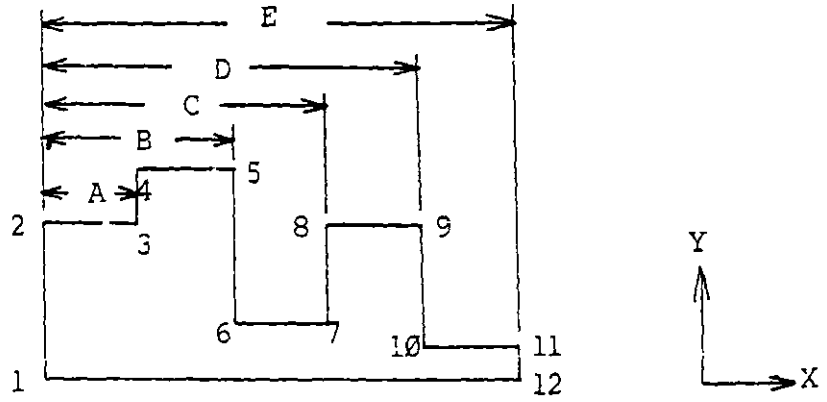


Figure 55, Datum dimensioning scheme.

A good dimensioning scheme reflects the function of a part and minimizes cumulative tolerances in critical areas. Cumulative tolerances are an undesireable condition and result when either the location of a surface or an overall dimension is affected by more than one tolerance dimension. The type of dimensioning scheme illustrated in Figure 54 is a consecutive dimensioning scheme in which each dimension is made relative to the one preceeding it. This is an example of a poorly dimensioned part because of the existence of cumulative tolerances. To avoid the inconsistency of cumulative tolerances, it is preferrable to locate surfaces from a datum plane. The dimensioning scheme shown in Figure 55 is an example of a datum dimensioning scheme. Although this type of dimensioning scheme is encountered in various

applications, some combination of datum dimensioning and consecutive dimensioning is usually used.

Thus, the size of the subsets of equations and coordinates found by the propagation method depends on the type of the dimensioning scheme used. In a dimensioning scheme which exhibits many consecutive dimensions, the size of the subsets would tend to be larger than in a dimensioning scheme in which a fixed datum is used. In a datum type dimensioning scheme, the implicit rigid body constraints should be chosen in such a way as to define the datum.

A numerical measure of the size of the subsets of equations and coordinates which will be found by the propagation method can be found by performing block triangularization on the entire set of equations before execution of the propagation method. The existence of small diagonal blocks represent a decoupled dimenisoning scheme similar to a datum type dimensioning scheme whereas large diagonal blocks indicate the possible existence of consecutive type dimensioning and cumulative tolerances. The size of the diagonal blocks can therefore be used as a measure of the quality of a dimensioning scheme and of the amount of reduction likely to be encountered by executing

the propagation method.

The presence of large diagonal blocks implies that for those sets of equations and coordinates which comprise the block, no equation may be solved independently of the other equations in the block. Unknown information must be passed to all the equations of the block before the equations can be solved. In the propagation method, if one equation in such a diagonal block is altered, all the equations of that block must be used in obtaining the solution.

To give a general idea of the amount of reduction obtained from the propagation method, various three-dimensional parts have been tested. Figure 56 plots the reduction found for various dimension changes. The subsets found are represented as percentages of the total number of equations.

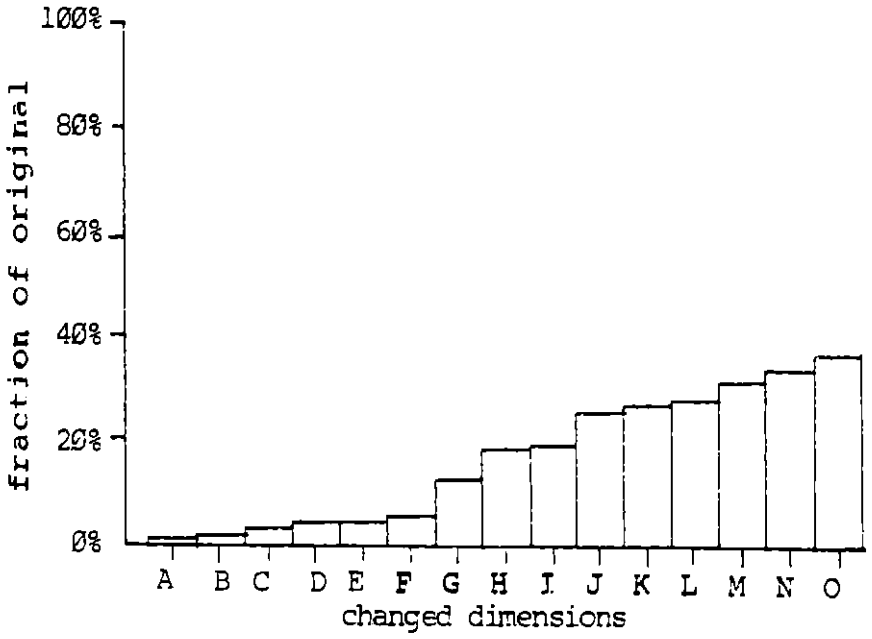


Figure 56, Plot of the relative reduction in the system of equations which need to be solved by using the propagation method.

Storage requirements for the propagation method is of O(n). Given any dimension change, to execute step two of the propagation method, the complete matching set must be stored in row access order in an integer*2 array of length n. Two integer*2 arrays of length m are used to store the row and column indices of the non-zeros in the Jacobian. Two integer*2 arrays of length n are used to store the subset of equations and coordinates which need to be solved given a dimension change. Total storage requirements for the propagation method is therefore 4m + 6n bytes.

7.Ø IMPLEMENTATION

7.1 SOFTWARE

A system of software written entirely in FORTRAN has been developed which demonstrates the concepts of three-dimensional variational geometry. The general program structure is shown in Figure 57.

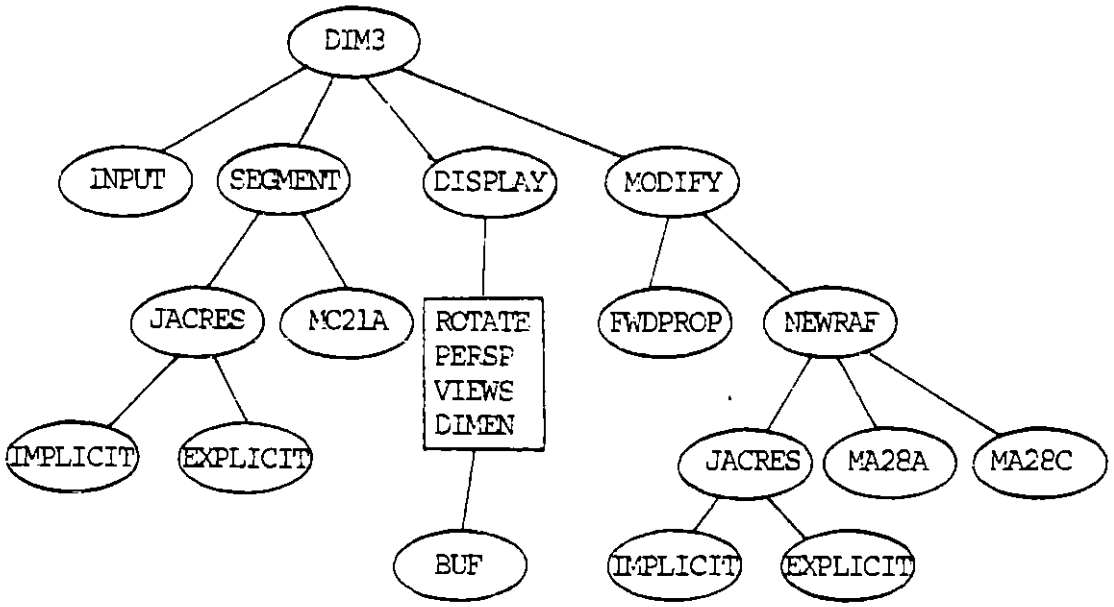


Figure 57, Program structure.

DIM3

This is the main program which calls the various subroutines which make up the variational geometry methods: data input, segmentation of the Jacobian, display of the part, and the numerical methods.

INPUT

The current implementation assumes the input of geometry, topology, and constraints has already been done. This subroutine reads from the DIMENSION system data base. The geometry, topology, constraints, and drawing parameters are read in from various disk files.

SEGMENT

This subroutine is a data handling routine which calls subroutines to activate the propagation method of segmentation.

JACRES

This subroutine computes the Jacobian matrix and residuals from the set of three-dimensional constraint routines.

MC21A

This subroutine finds a complete matching between constraint equations and coordinates for purposes of segmentation.

DISPLAY

This subroutine is a data handling routine which calls subroutines to display, rotate, and dimension the part.

PERSP

This subroutine calls various drawing routines to generate a perspective view of the three-dimensional part given its geometry and topology.

ROTATE

This subroutine rotates a perspective view of the three-dimensional part.

VIEWS

This subroutine generates the three orthogonal views of the three-dimensional part.

DIMEN

This subroutine calls various routines to draw the dimension lines, arrows and dimension label in the three orthogonal views.

BUF

This subroutine is a data transfer routine which transfers data from the main computer to the graphics display for drawing purposes. It also receives data input from the graphics display to execute the various user commands.

MODIFY

This subroutine accepts as input the dimension(s) to be changed, the new value(s) of the dimension(s). and the step number and calls the numerical methods routines to alter the part geometry.

FWDPROP

This subroutine takes as input the matching set produced in MC21A and the position of the non-zeros of the Jacobian in row order and produces a subset of coordinates and constraints which need to be solved for given any dimension change.

NEWRAF

This routine accepts as input the subset of coordinates and constraints from subroutine FWDPROP. It then calls JACRES to compute the Jacobian and the residuals for only the subset of constraints and coordinates from the set of three-dimensional constraint routines. Routine MA28A/MA28C is then called to solve for the points vector at each iteration of the modified Newton-Raphson method.

Currently, input of topology and assignment of constraints are accomplished by reading from data files and not through an interactive procedure.

On entry to the program, various data files, which contain the geometry, topology, constraints, and dimensions for a specific part are read in. Because the geometry satisfies all the constraint equations, no numerical operations are required at this stage. If the topology were to be entered using an approximate interactive input method, then the Newton-Raphson method would need to be performed at this stage to reconcile the geometry with the constraints.

The program then proceeds to find a complete matching set for the constraints and coordinates by a depth first search process. At this stage, if the constraining scheme is redundant, a redundant constraint will be flagged and the program will exit.

Given a permissible dimensioning scheme, the program then displays the part in a perspective view. At this stage, the user may rotate the part about its x-y centroid or display the three orthographic views of the part showing its various dimensions. A perspective view is also provided to aid in visualization.

The user may now select any of the displayed dimensions by use of a digitizing pen on a data tablet with feedback via a screen cursor. A numeric keypad then replaces the perspective view and the user is prompt for the new dimensional value. Up to ten dimensions may be altered at any one time. After entering the new value(s) of the dimension(s) to be changed, the user is prompt for the step number. The step number serves two purposes, to increment the geometry in a number of steps for display purposes and to increase the stability of the Newton-Raphson iterations.

Upon entering the step number, the program performs the second step of the propagation method to determine the subset of constraints and coordinates which need to be solved. These constraints and coordinates are then used in the Newton-Raphson method. The part is incremented dynamically in a full perspective view from its initial to final geometry. The process may then be repeated as the user is prompt for either a rotation of the changed geometry or a display of the three orthographic views.

7.2 HARDWARE

The main program for the DIMENSION system as well as the data base resides on a Digital Equipment VAX 11/780 virtual memory computer located in Massachusetts Institute of Technology's Joint Computer Facility. The drawing routines for the DIMENSION system resides on a PDP 11/34 minicomputer with an internal program memory capacity of 64K bytes.

Communication between the main program and drawing routines is via DECNET, a data network produced by Digital Equipment Corporation. For actual display of the geometry of the part, the PDP 11/34 drives a MEGATEK 7000 vector refresh display.

User interface to the geometry of the part as well as alteration of dimensions is by using a digitizing pen on a Summagraphics BITPAD data tablet. A software screen cursor provides the necessary feedback.

All numerical computations are performed on the VAX 11/780. Display of vector and character data as well as cursor addressing and user prompting are performed on the PDP 11/34.

8.Ø RESULTS

A suitable data structure for three-dimensional variational geometry has been developed. The data structure contains complete geometric and topological information of a part. In addition, a constraint and dimension structure was identified to allow interface to methods of interactive constraining, dimension alteration, and display.

Three-dimensional constraint equations have been derived for objects bounded by planar, cylindrical, spherical, and conical surfaces. Application of the constraints to three-dimensional objects allows for ease of manipulation and modification of three-dimensional geometries. The result is a reduction in user interaction and the creation of a friendlier interface to CAD systems.

The efficiency of the numerical methods were investigated. Various methods to increase the efficiencies of computation and storage have been developed and implemented. To increase computational efficiency, modified Newton-Raphson method as well as sparse elimination were implemented. To further increase computational efficiency, a method to segment the total number of equations and

coordinates into a smaller subset for purposes of the numerical computations was developed. To increase the efficiency of storage, sparse matrix storage methods were used.

Finally, to demonstrate the utility of the variational geometry methods, a system of software has been written and implemented which makes up part of the DIMENSION system. This system of software allows a part to be entered through geometry and topology data files. The part is constrained by the constraint and dimension data files. Interactive methods are then used to allow the user to display and view the part in various views. To alter the geometry, the user simply selects the dimension(s) to be changed and enters the new dimensional value(s).

9.0 CONCLUSIONS AND RECOMMENDATIONS

A system, embodying the concept of variational geometry, has been developed. Three-dimensional shape models consisting of planes, cylinders, spheres, and cones are built up with the use of three-dimensional constraints. Modification of part geometry involves a simple two-step process, selecting the dimension(s) to be changed and entering the new value(s).

One of the current benefits of the system is simplicity of the user interface. By using variational geometry to create and modify a part, the process of design is made easier. Since design is inherently iterative, the techniques of variational geometry greatly facilitates the design process.

The concepts of variational geometry can be expanded to include constraining of clearances between two or more mating parts. By specification of specific constraints in the design of assemblies, part mismatch and interference can be avoided.

Because the current methods of variational geometry operate on a specific class of surfaces, future investigations include extension of these concepts to general quadric surfaces and free form surfaces.

Since the geometry of a part can be modified easily, it is possible that a part can be made to intersect itself. Therefore, methods of detecting self-intersection of a part must be implemented.

Current methods of constraining of the part topology is through the use of data files. To make the DIMENSION system fully interactive, methods of interactively specifying constraints will need to be developed.

REFERENCES

1. Baer,A. Eastman,C. and Henrion,M. "Geometric Modelling, A Survey", Computer-Aided Design, Vol 11, No. 5, Sept 1979, pp 253-272.

2. Beltrami, E.J., An Algorithmic Approach to Non-linear Analysis and Optimization, Academic Press, New York, 1970.

3. Congdon, R.M., Three-Dimensional Shape Input Through Sketch Recognition, M.S. Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1981.

4. DeKleer,J. and Sussman,G.J., Propagation of Constraints Applied to Circuit Synthesis, MIT Artificial Intelligence Laboratory Memo. No. 485, Cambridge, Massachusetts, 1978.

5. Duff, I.S., "A survey of Sparse Matrix Research", Proc. IEEE, Vol 65, No. 4, April 1977, pp 500-535.

6. Duff, I.S., MA28 - A Set of Fortran Subroutines for Sparse Unsymmetric Linear Equations, Harwell Report No. R 8730, Oxfordshire, England, Nov. 1980.

7. Duff, I.S. and Reid, J.K., "Some Design Features of a Sparse Matrix Code", ACM Trans. on Math. Software, Vol 5, No. 1, March 1979, pp 18-35.

8. Dulmage, A.L. and Mendelsoln, N.S., "On the Inversion of Sparse Matrices", Math. Comp, 16, 1962, pp 494-496.

9. Dulmage, A.L. and Mendelsoln, N.S., "Two Algorithms for Bipartite Graphs", J. Soc. Indus. Math., Vol 11, No. 1, Mar 1963, pp 183-194

10. Even, S., Graph Algorithms, Computer Science Press, Maryland, 1979.

11. Fox, L. Introduction to Numerical Linear Algebra, Oxford University Press, London and New York, 1965.

12. Gustavson, F.G., "Finding the Block Lower Triangular Form of a Sparse Matrix", in Sparse Matrix Computations, New York, Academic Press, 1976.

13. Gustavson, F.G., "Some Basic Techniques for Solving Sparse Systems", in Sparse Matrices and Their Applications (D.J.Rose and R.A.Willougby, Eds.) pp 41-52,

Plenum Press, New York, 1972.

14. Hall, M., "An Algorithm for Distinct Representatives", Amer. Math. Monthly, 1956, pp 716-717.

15. Harary, F., "Sparse Matrices and Graph Theory", in Large Sparse Sets of Linear Equations (J.K. Reid Ed) pp 139-150, Academic Press, New York, 1971.

16. Hillyard, R.C., Dimensions and Tolerances in Shape Design., Technical Report No. 8, University of Cambridge Computer Laboratory, Cambridge, England, 1978.

17. Hillyard, R.C. and Braid, I.C., "Analysis of Dimensions and Tolerances in Computer-Aided Mechanical Design", Computer-Aided Design, Vol 10, pp 161-166, June 1978.

18. Hillyard, R.C., and Braid, I.C., "Characterizing Non-ideal Shapes in Terms of Dimensions and Tolerances", Computer Graphics, ACM-Siggraph, Vol 12, No.3, pp 234-238, Aug 1978.

19. Hopcroft, J.E. and Karp, R.M., "An $n^{3/2}$ Algorithm for Maximum Matchings in Bipartite Graphs", SIAM J. Computing, Vol 2, No.4, pp 225-231, Dec 1973

20. Light, R.A., Symbolic Dimensioning in Computer-Aided Design, M.S. Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1980.

21. Lin, V.C., Two Segmentation Methods, Technical Document No. 21, MIT Computer-Aided Design Laboratory, Cambridge, Massachusetts, 1980.

22. Morrill, W.K., Selby, S.M., and Johnson, S.G., Modern Analytic Geometry, Intext Educational Publishers, Scranton, Penn., 1972.

23. Ogbuobiri, E.C., "Dynamic Storage and Data Retrieval in Sparsity Programming", IEEE Transactions on Poower Apparatus and Systems, Vol PAS-89, No. 1, pp 150-155, Jan 1970.

24. Ortega, J.M. and Reinboldt, W.C., Iterative Solutions of Non-linear Equations in Several Variables, Academic Press, New York, 1970.

25. Reid, J.K., "Solution of LInear Systems of Equations: Direct Methods (General)", in Lecture Notes in

Mathematics: Sparse Matrix Techniques, Copenhagen 1976 (Barker, V.A. Ed), pp 102-127.

26. Reingold, E.M., Nievergelt, J. and Deo, J., Combinatorial Algorithms: Theory and Practice, Prentice Hall, New Jersey, 1977.

27. Steele, G.L. and Sussman, G.J., Constraints, MIT Artificial Intelligence Laboratory Memo No. 502, Cambridge, Massachusetts, 1978.

28. Steward, D.V., "On an Approach to Techniques for the Analysis of the Structure of Large Systems of Equations", SIAM Review, Vol 4, No. 4, pp 321-342, Oct 1962.

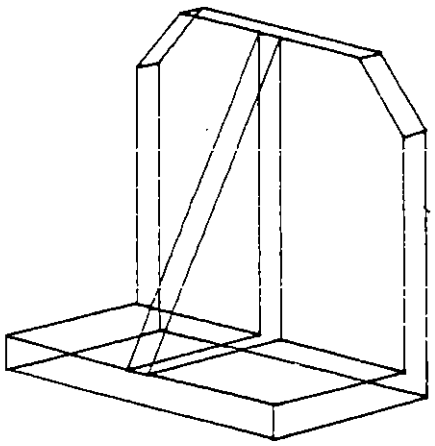
29. Sutherland, I.E., Sketchpad: A Man-machine Graphical Communication System, MIT Lincoln Laboratory Technical Report No. 296, Lexington, Massachusetts, 1963.

30. Tewarson, R.P., Sparse Matrices, Academic Press, New York, 1973.

31. Vandergraft, J.S., Introduction to Numerical Calculations, Academic Press, New York, 1978.

APPENDIX A - OBJECT WITH PLANAR SURFACES

Figure A-1 shows a full perspective view of a three-dimensional object made up of only planar surfaces. Figure A-2 shows the part in rotation. In Figure A-3, orthographic views of the part are generated. To change the height of the flange, the appropriate dimension, 15.ØØ in this case is selected with the digitizing pen. Figure A-4 shows the process of entering the new dimensional value using a numerical keypad which is displayed in place of the perspective view. Figure A-5 shows the entering of the step number. Figure A-6 shows the object incrementing towards its new geometry. Figures A-7 to A-11 shows the process of moving the web of the flange 2.ØØ units to the left.



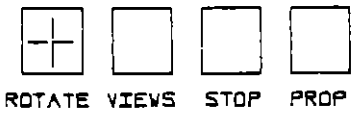
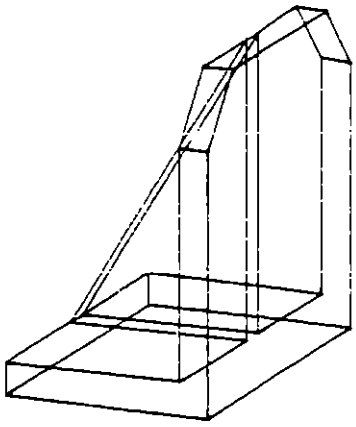


Figure A-1, Object with planar surfaces.



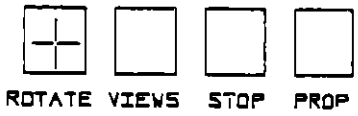


Figure A-2, Object in rotation.

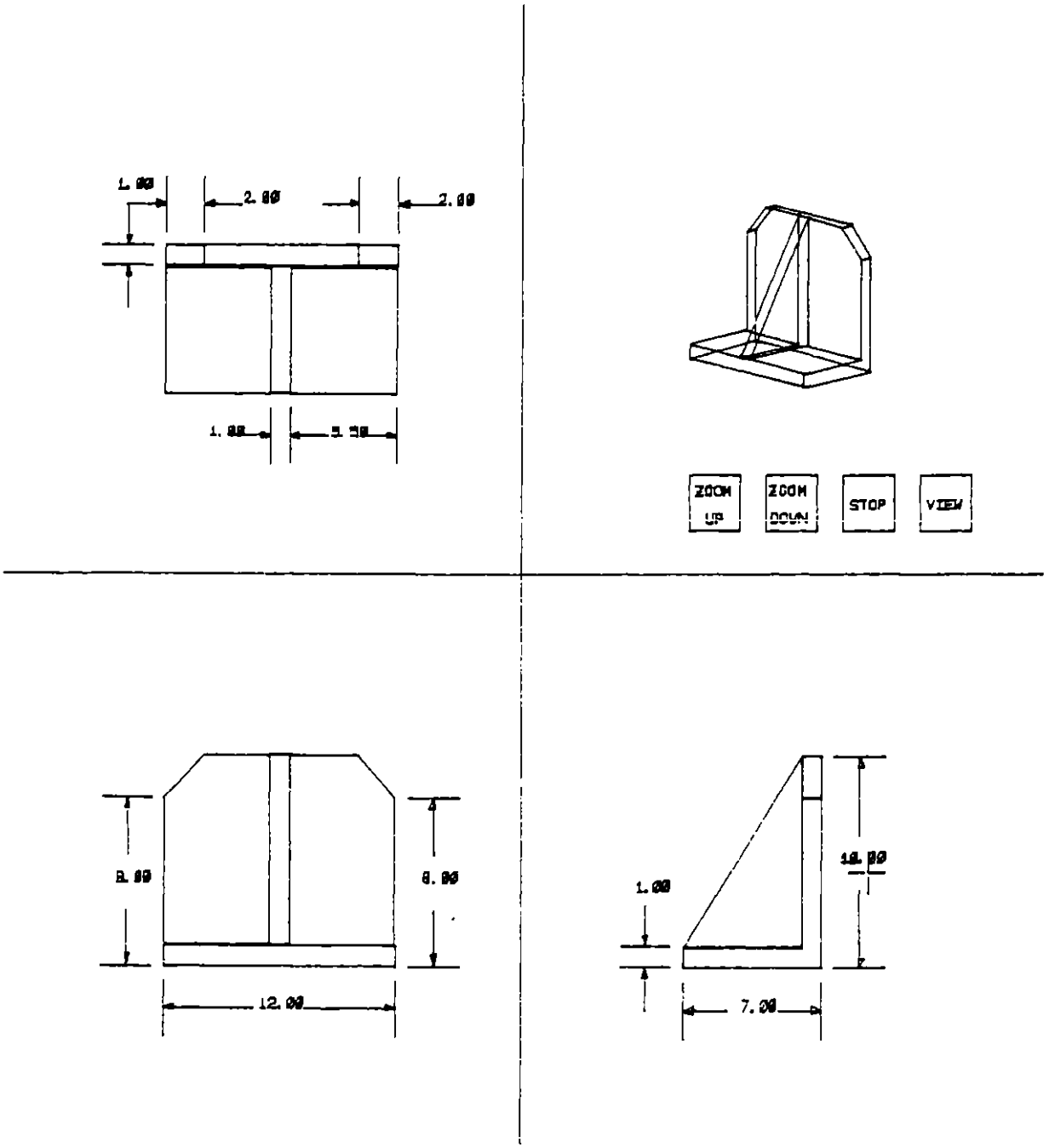


Figure A-3, Selecting dimension to be changed.

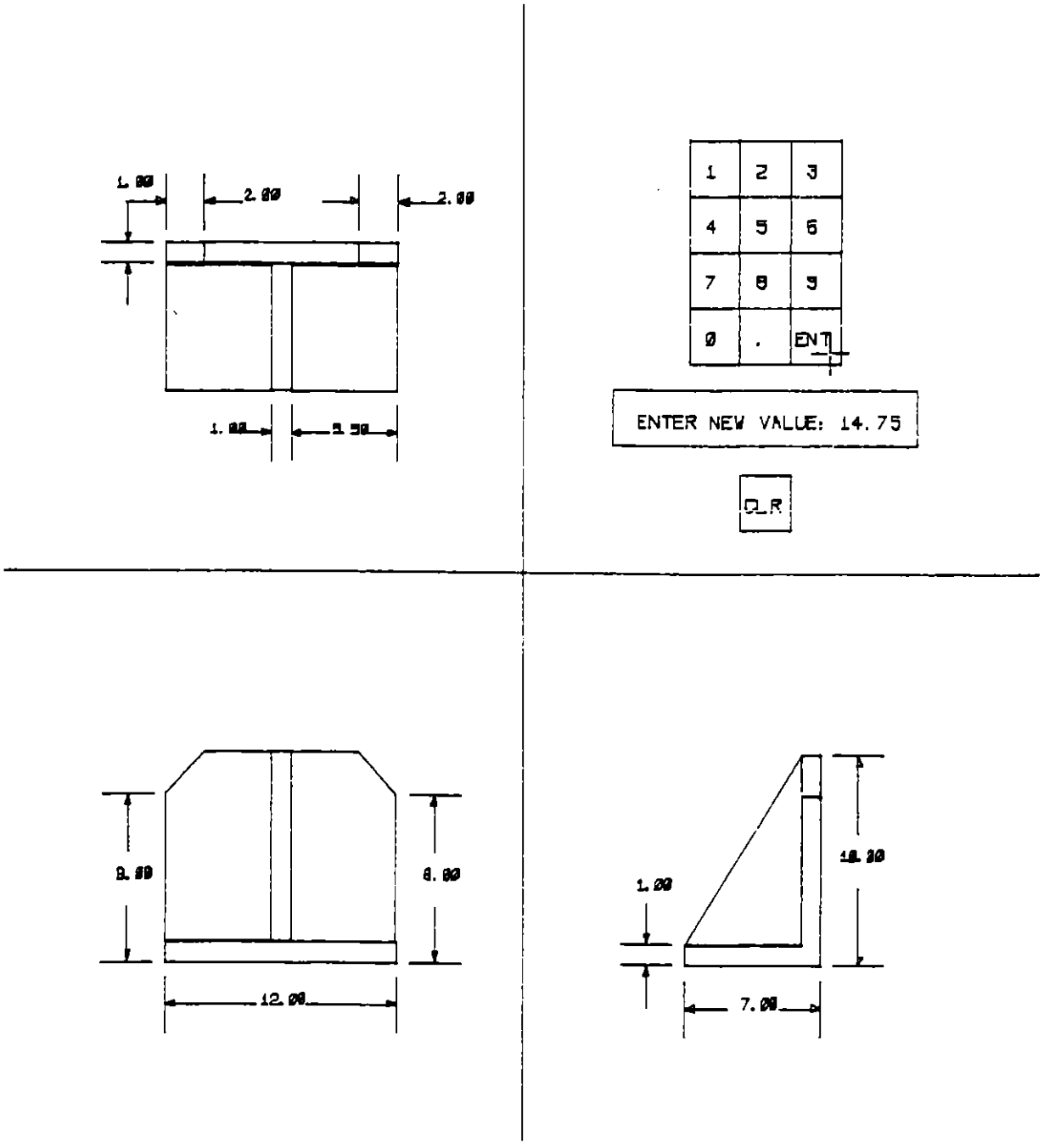


Figure A-4, Entering new value.

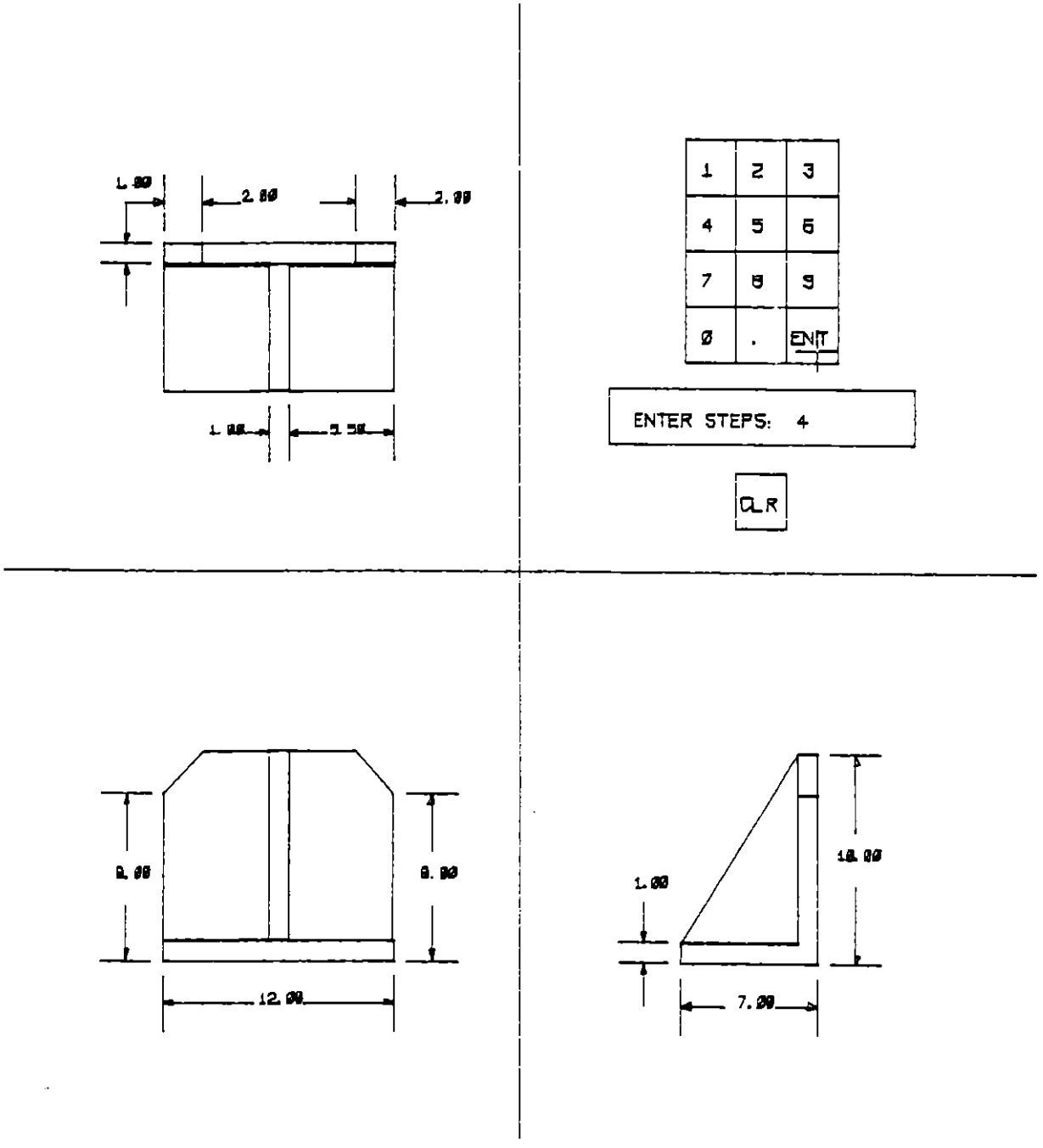


Figure A-5, Entering number of steps.

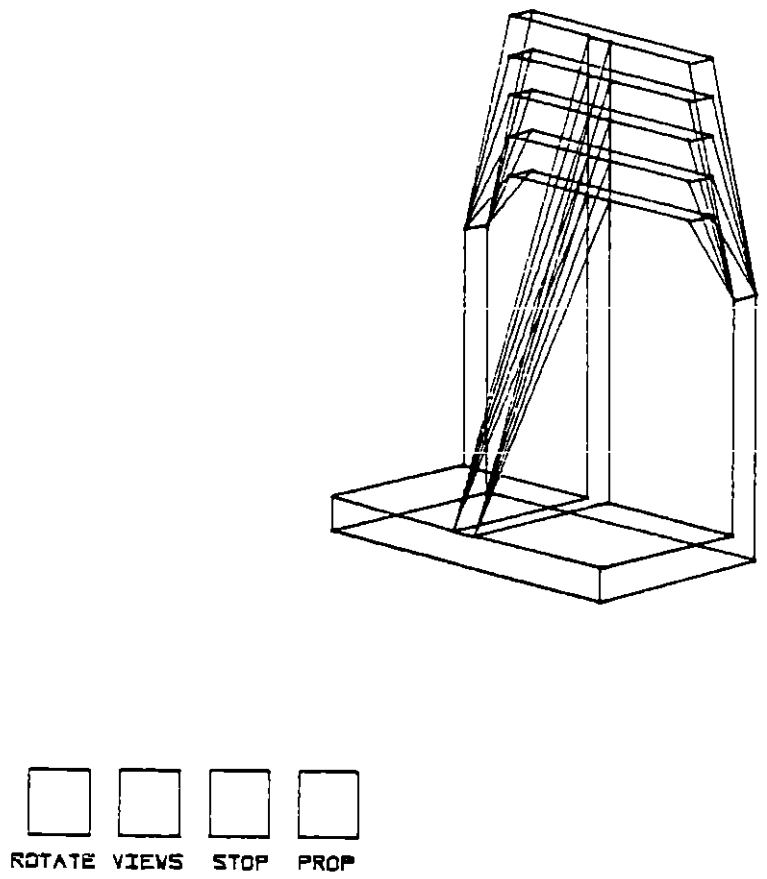


Figure A-6, Alteration of geometry.

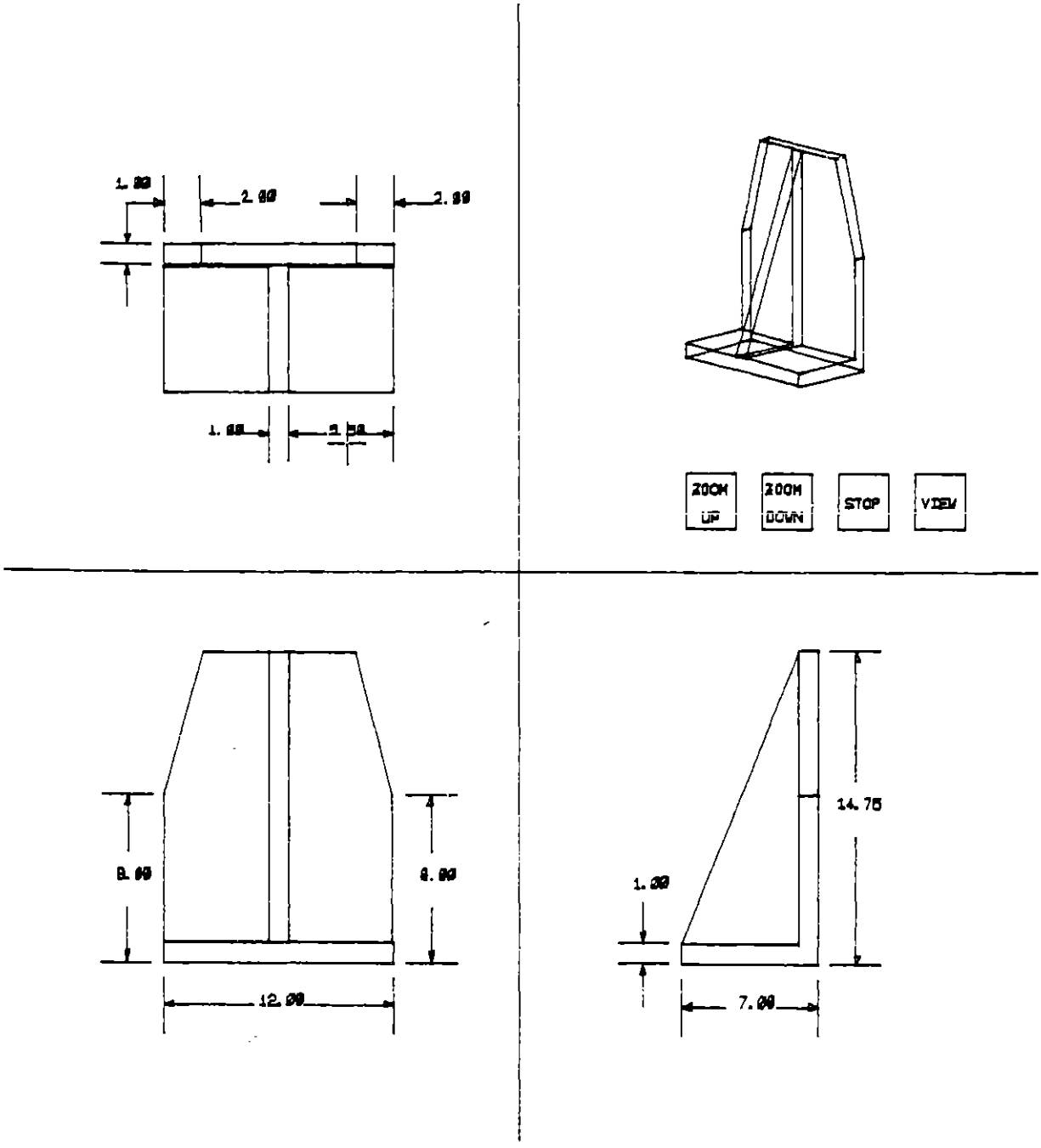


Figure A-7, Selecting dimension to be changed.

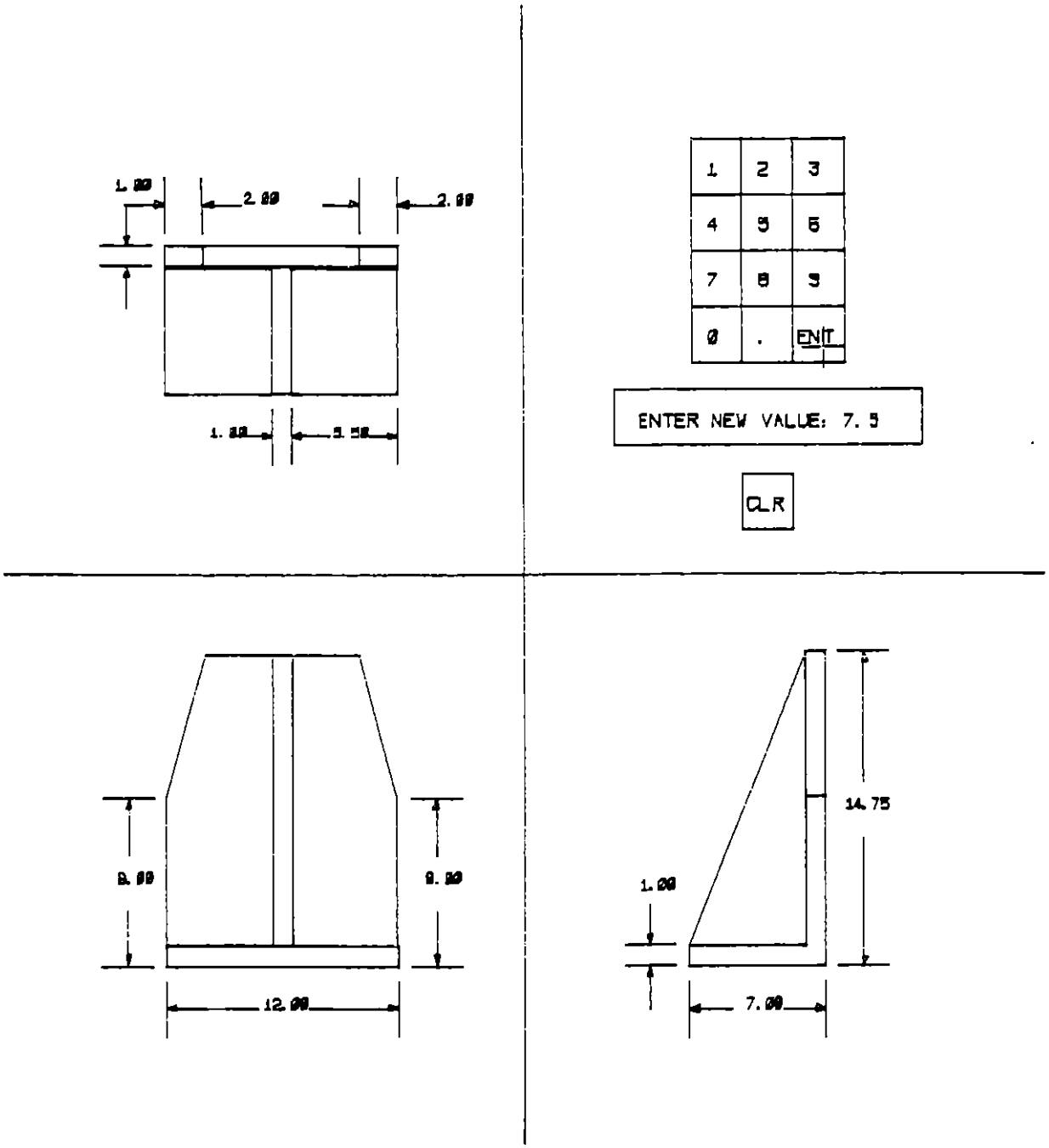


Figure A-8, Entering new value.

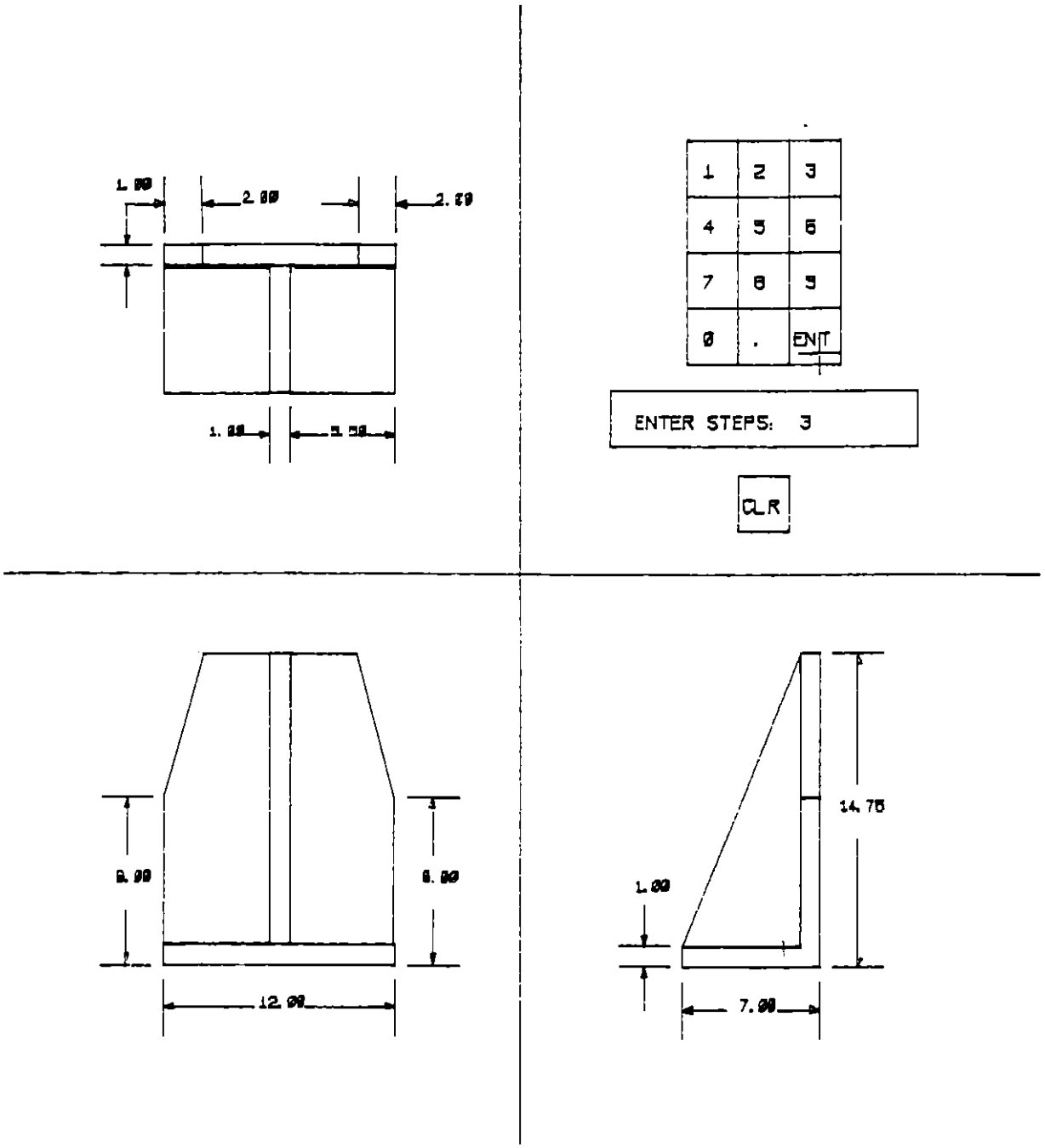


Figure A-9, Entering number of steps.

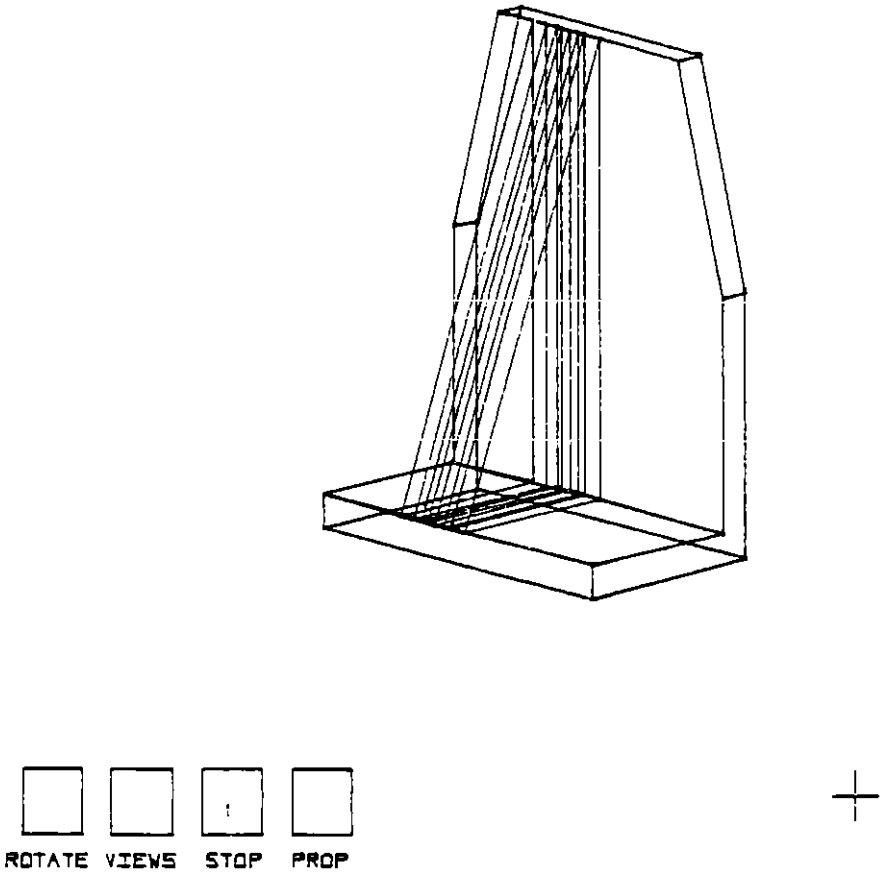


Figure A-1Ø, Alteration of geometry.

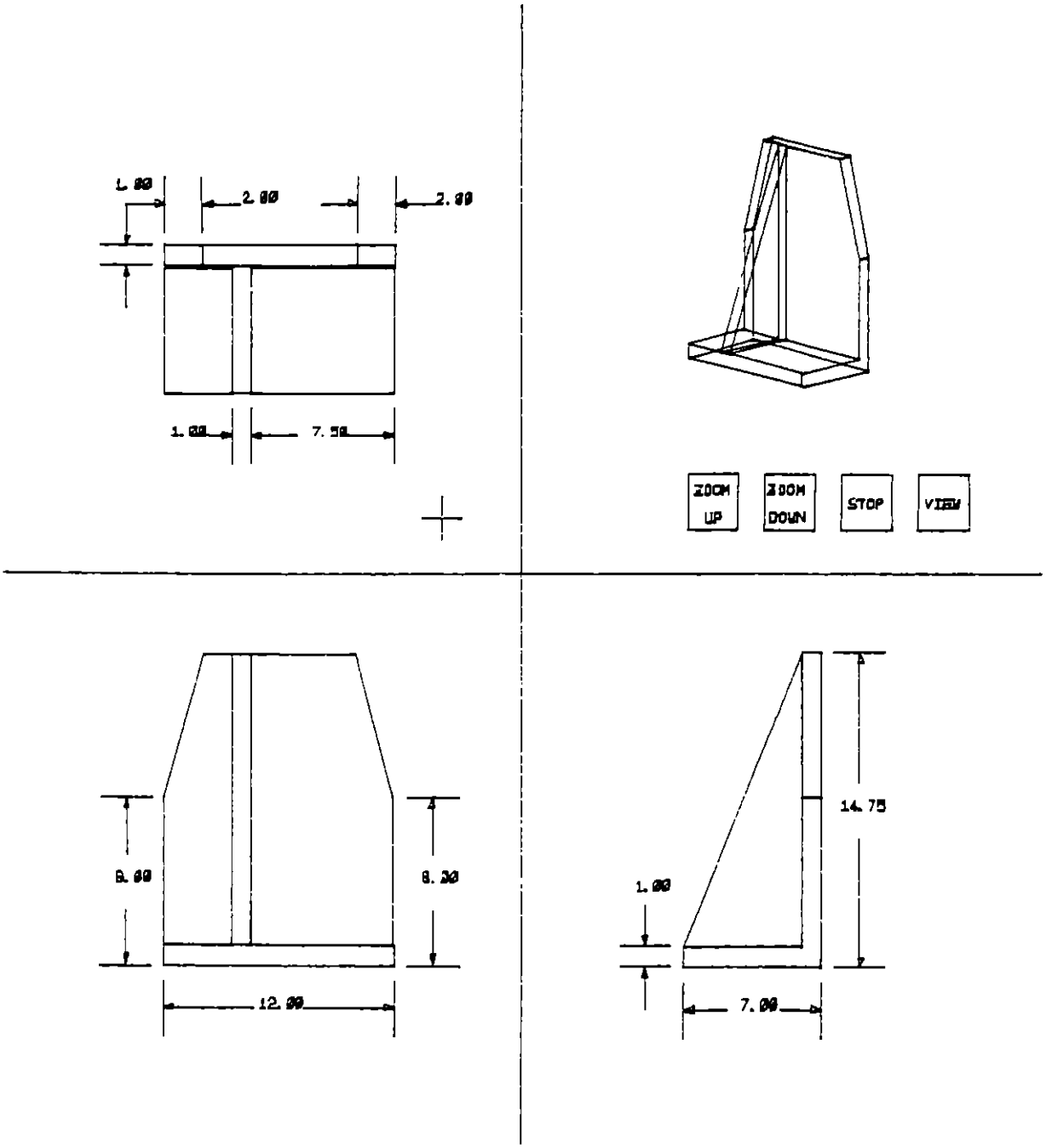
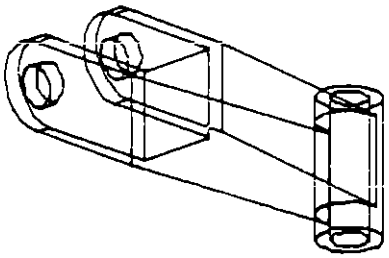


Figure A-11, Views of new geometry.

APPENDIX B - OBJECT WITH CYLINDRICAL SURFACES

Figure B-1 shows a full perspective view of a three-dimensional object made up of planar and cylindrical surfaces. Again, Figure B-2 shows the part in rotation and Figure B-3 shows its three orthographic projections. Figure B-4 to B-7 shows the process of changing the length of the solid portion of the guide bracket. Figures B-8 to B-11 show the process of changing the diameter of the cylindrical surface. In a similar manner, any displayed dimension can be used to modify the geometry of the guide bracket shown.



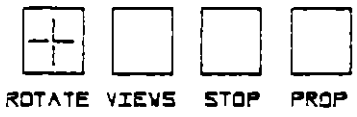


Figure B-1, Object with cylindrical surfaces.

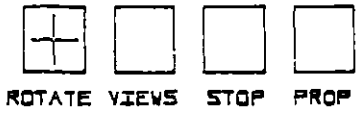


Figure B-2, Object in rotation.

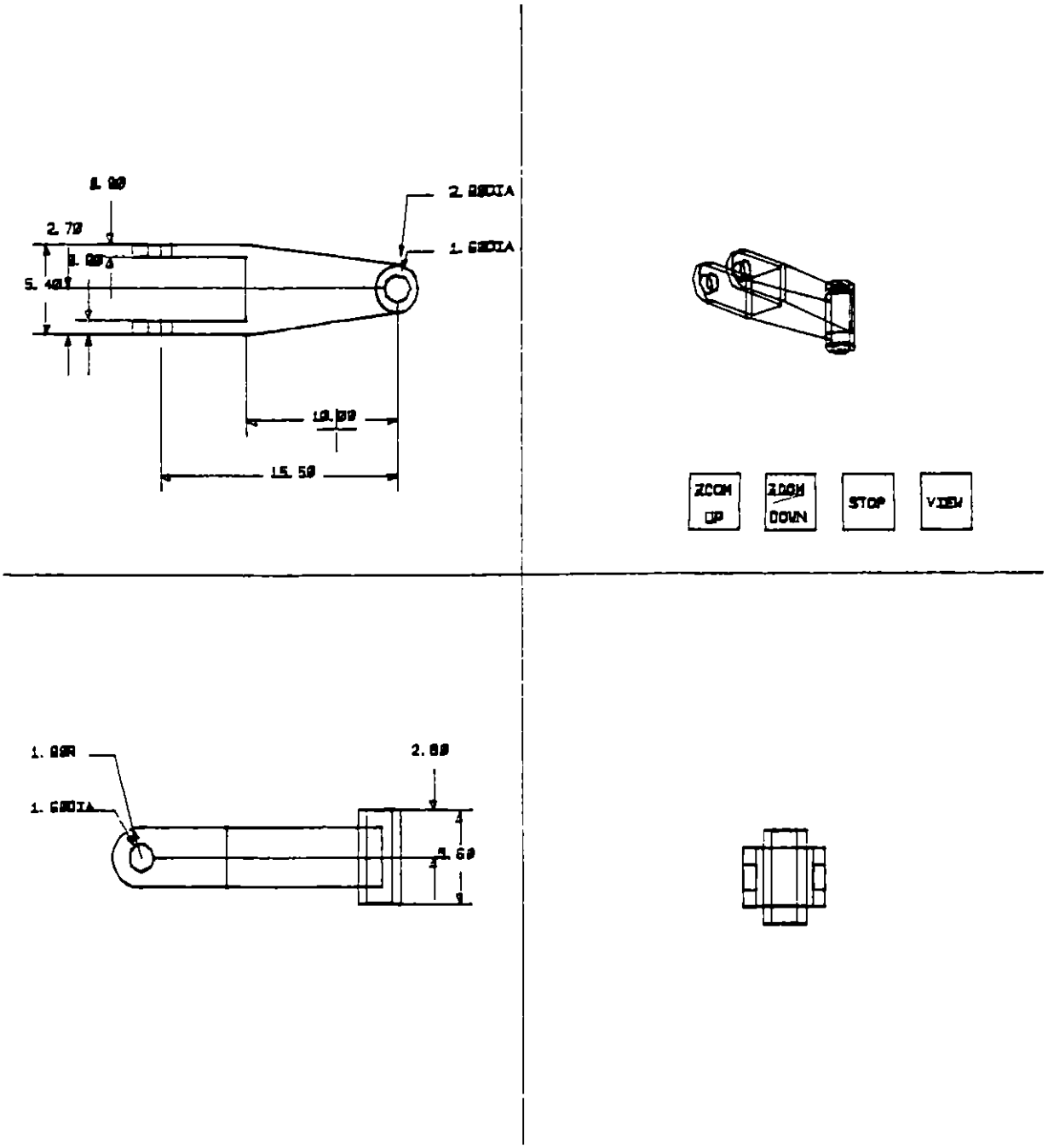


Figure B-3, Selecting dimension to be changed.

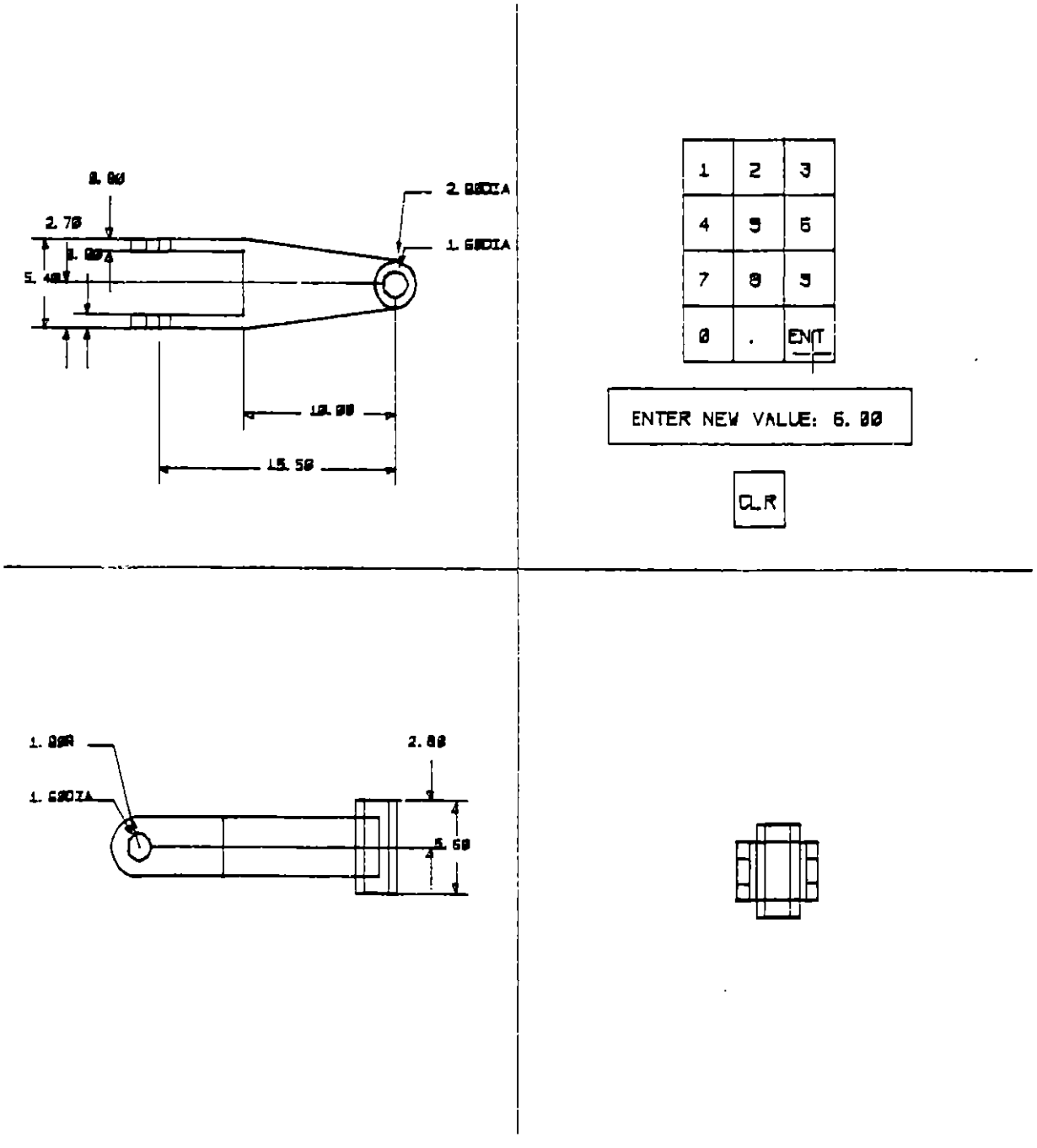


Figure B-4, Entering new value.

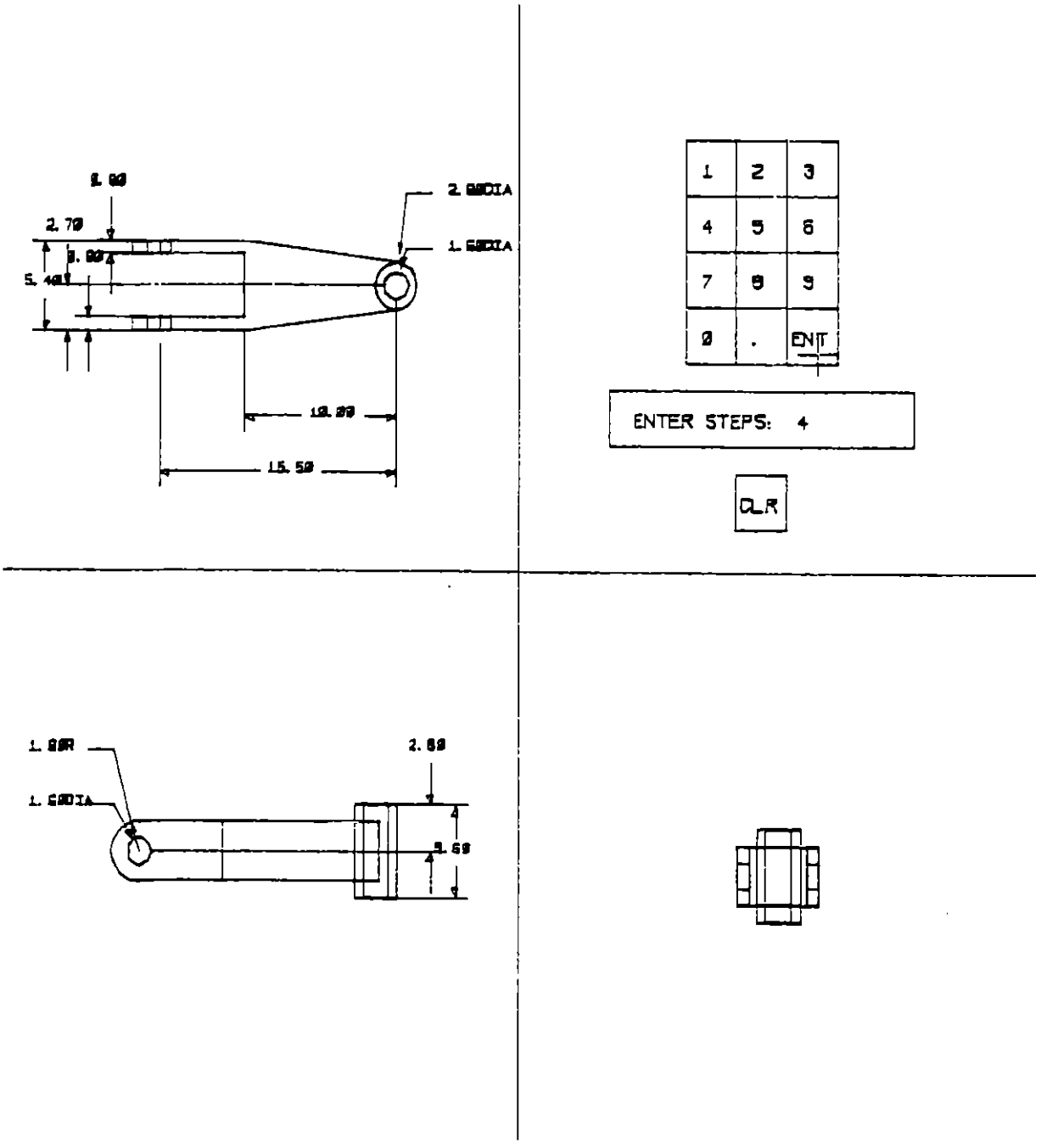


Figure B-5, Entering number of steps.

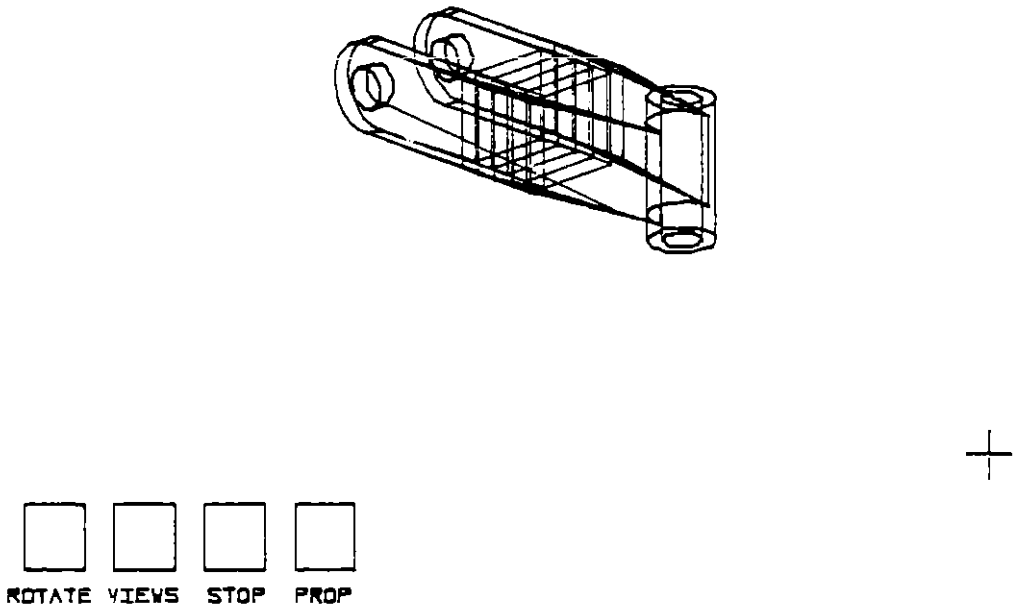


Figure B-6, Alteration of geometry.

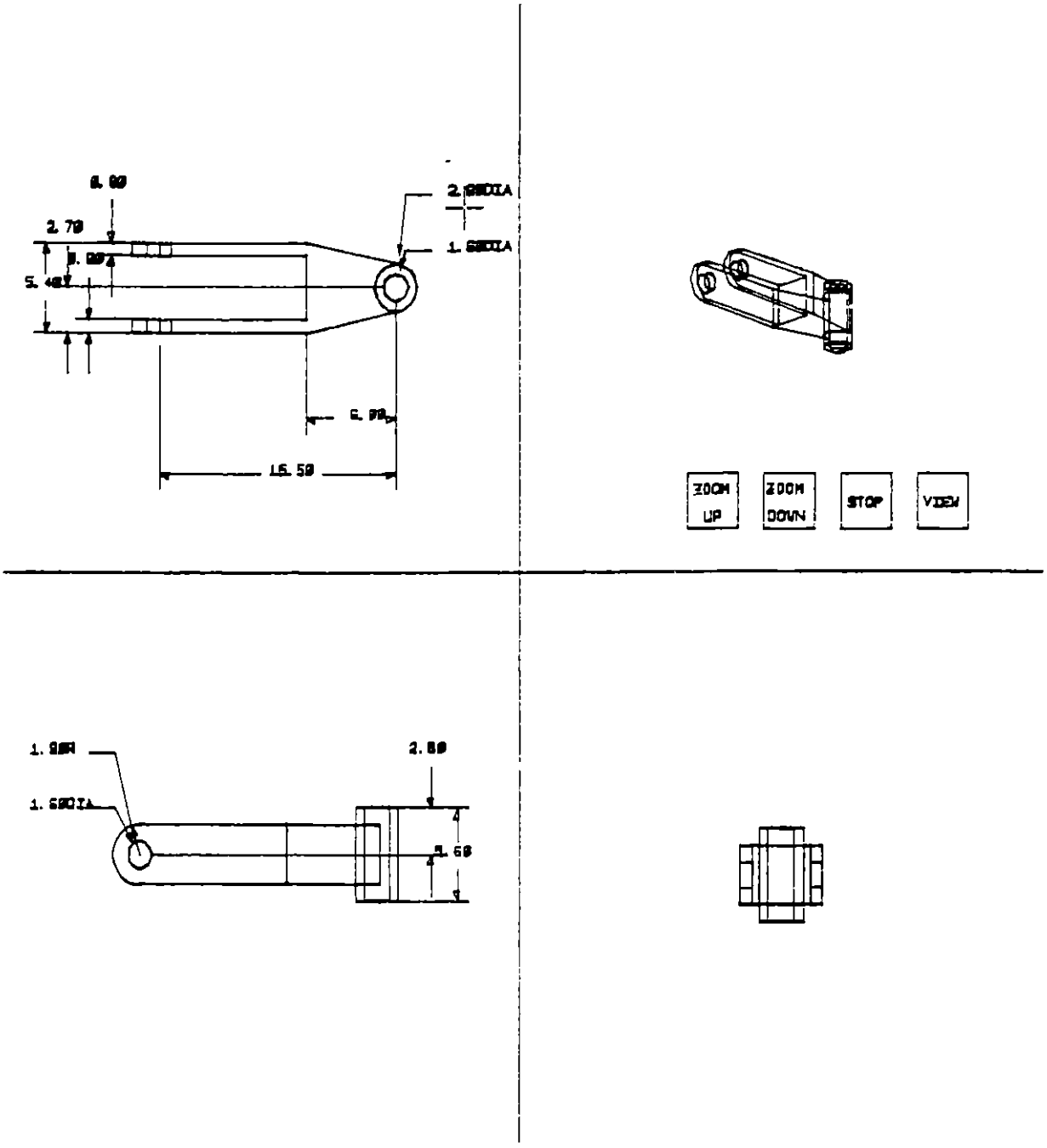


Figure B-7, Selecting dimension to be changed.

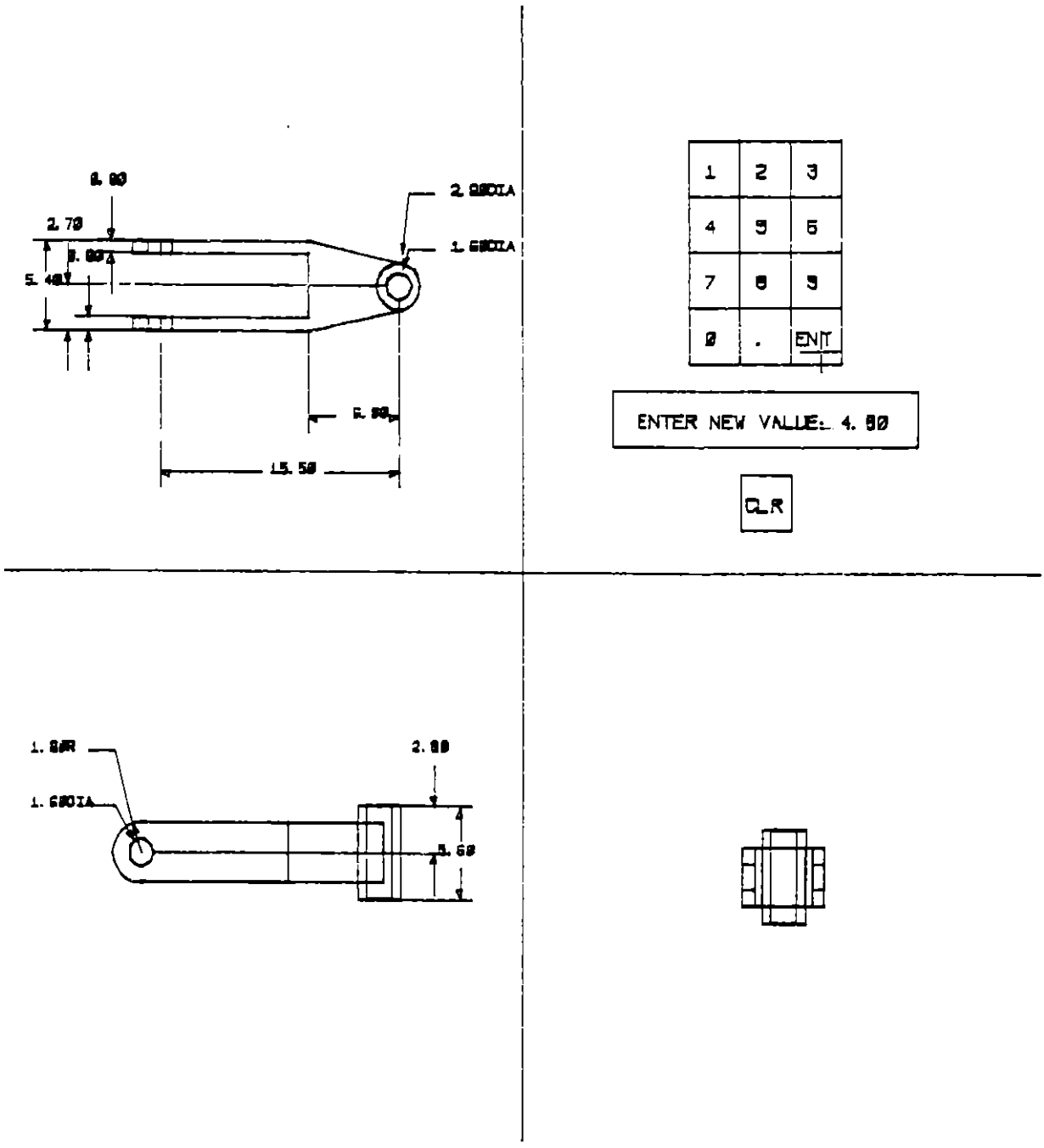


Figure B-8, Entering new value.

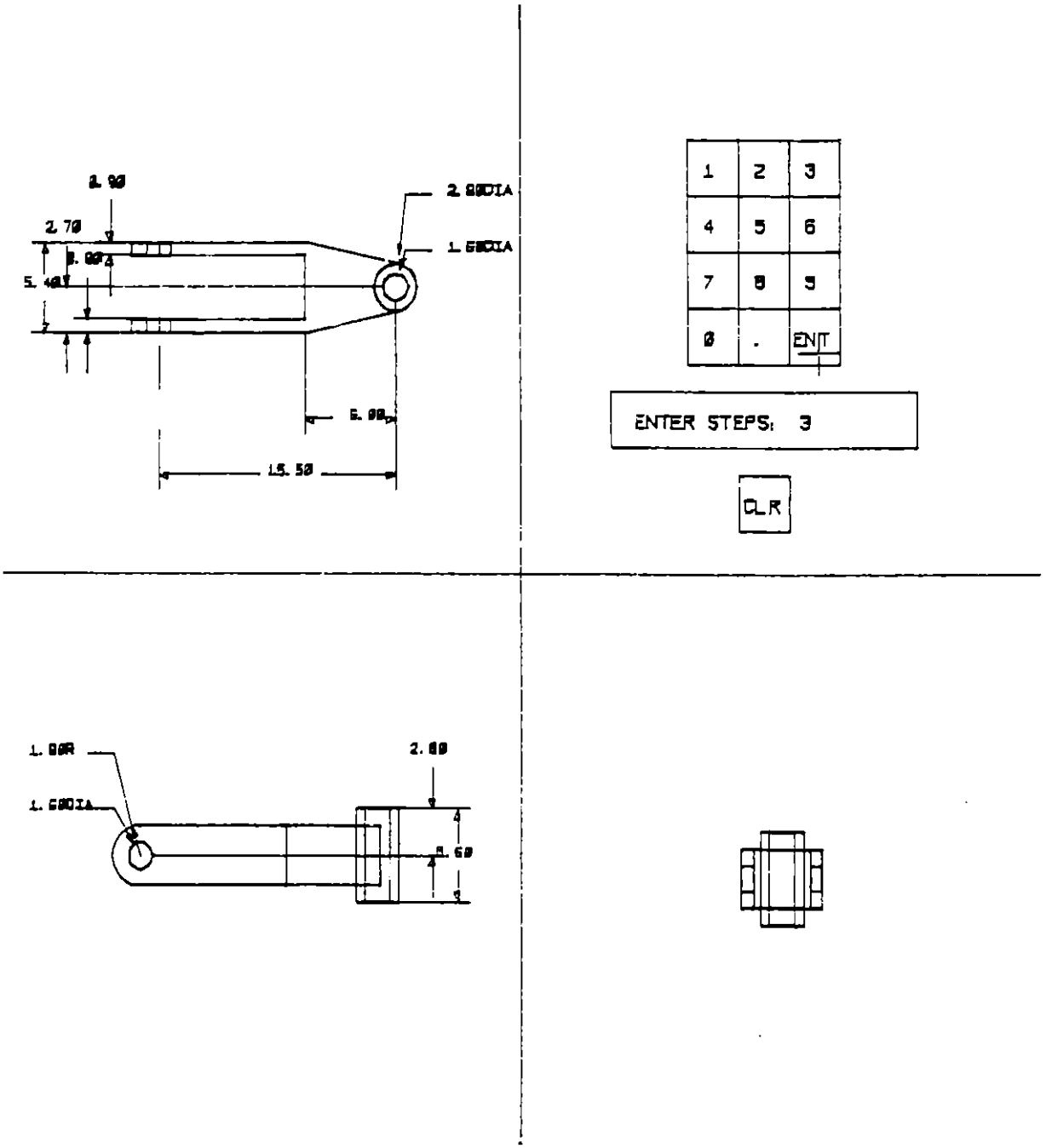


Figure B-9, Entering number of steps.

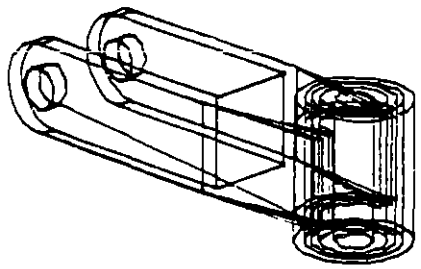


Figure B-1Ø, Alteration of geometry.

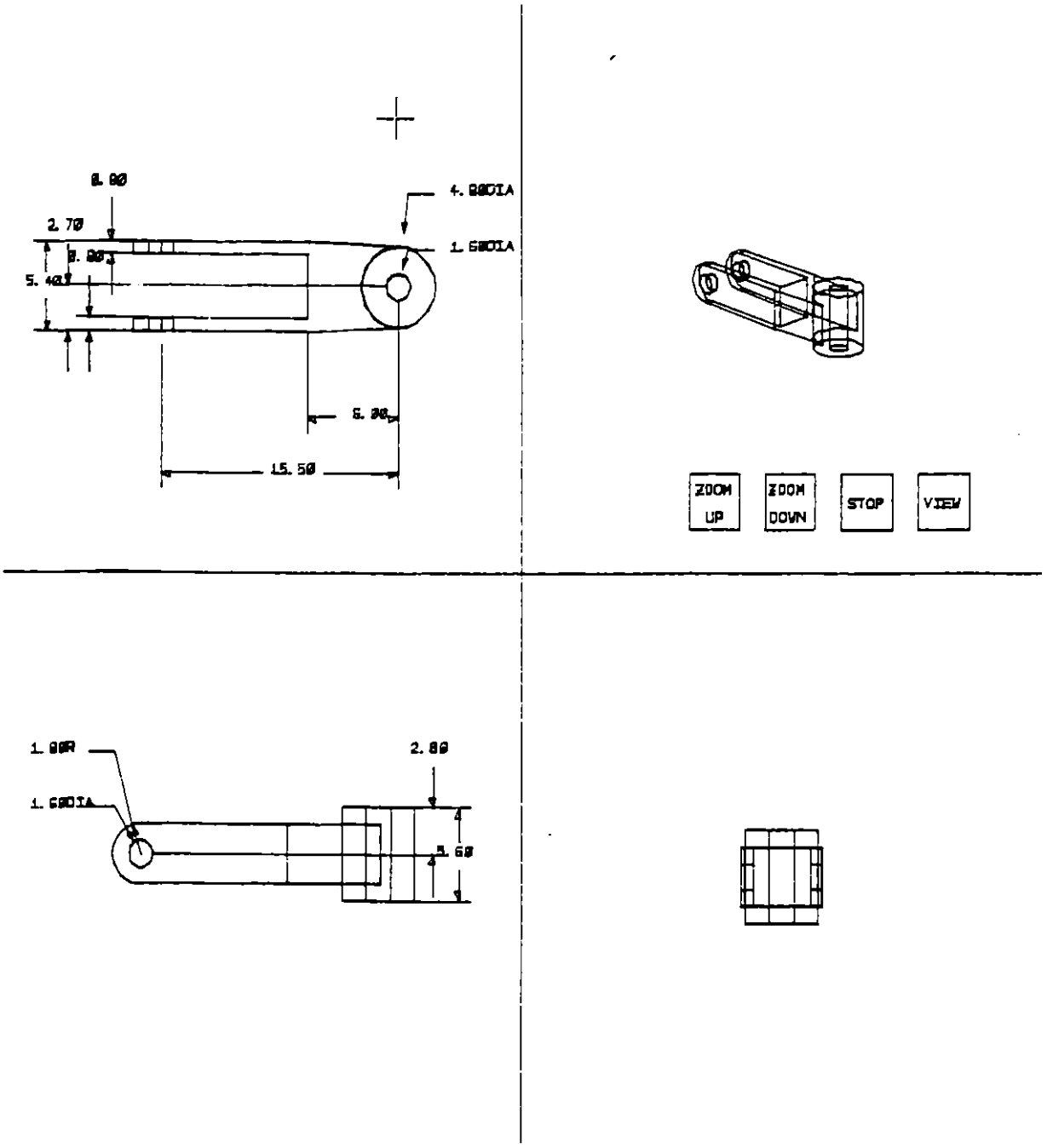


Figure B-11, Views of new geometry.

APPENDIX C - NEWTON-RAPHSON METHOD

The Newton-Raphson method is a generalization of Newton's method for finding the roots of a function, f(x) = Ø. To solve this problem, the function is approximated at some point, xo, by its tangent. The value of x, where the tangent crosses the x-axis is solved for. This process is repeated from the new point, x1, until the algorithm converges. Newton's method defines a sequence of points according to the recurrence relation

$$xn+1 = xn - \frac{f(xn)}{f'(xn)} \qquad (C.1)$$

The convergence depends on the initial estimate of the solution, xo, with faster convergence resulting as xo approaches the actual solution.

The Newton-Raphson algorithm expands Newton's method to n functions. Given n constraining equations in n unknowns:

$$\begin{aligned} f1(\underline{x}j) &= f1(x1,x2,\ldots,xn) = Ø \\ f2(\underline{x}j) &= f2(x1,x2,\ldots,xn) = Ø \\ &\ldots\ldots\ldots\ldots\ldots \\ fN(\underline{x}j) &= fN(x1,x2,\ldots,xn) = Ø \end{aligned} \qquad (C.2)$$

we want to find the solution vector $\underline{x} = [x1, x2, \ldots, xn]$, where $[x1, x2, \ldots, xn]$ are the points defining the geometry of the part.

In the neighborhood of an initial estimate, $\underline{x}j^{(p)}$, we can approximate the above set of equations by the linear terms of a Taylor series

$$fi(\underline{x}j) = \emptyset = fi = \left(\frac{\partial fi}{\partial x1}\right)\Delta x1 + \left(\frac{\partial fi}{\partial x2}\right)\Delta x2 + \ldots + \left(\frac{\partial fi}{\partial xn}\right)\Delta xn \quad (C.3)$$

where $i = 1, 2, \ldots, n$ and fi and $\left(\frac{\partial f_i}{\partial x_j}\right)$ represents the functions and their partial derivatives evaluated at point $\underline{x}j$. $\Delta\underline{x}j = \underline{x}j^{(p+1)} - \underline{x}j^{(p)}$ are the displacements of the points. From $\Delta\underline{x}j$, the next estimate, $\underline{x}j^{(p+1)}$ can be calculated. The Taylor series expansions can be written in matrix form:

$$[J]\,[x] = -[f] \quad (C.4)$$

where the Jacobian matrix [J] given by

$$[J] = \left[\frac{\partial fi}{\partial xj}\right] = \begin{bmatrix} \frac{\partial f1}{\partial x1} & \frac{\partial f1}{\partial x2} & \cdots & \frac{\partial f1}{\partial xn} \\ \frac{\partial f2}{\partial x1} & \frac{\partial f2}{\partial x2} & \cdots & \frac{\partial f2}{\partial xn} \\ \vdots & \vdots & & \\ \frac{\partial fn}{\partial x1} & \frac{\partial fn}{\partial x2} & \cdots & \frac{\partial fn}{\partial xn} \end{bmatrix} \quad (C.5)$$

is evaluated at $\underline{x}j^{(p)}$. $\underline{f}$ = [f1,f2,...,fn] are termed the residuals and are evaluated at $\underline{x}j^{(p)}$.

If we begin with an initial estimate $\underline{x}j^{(1)}$, $fi(\underline{x}j^{(1)}) \neq \emptyset$ because $\underline{x}j^{(1)}$ is not an exact solution. By solving [J] [x] = -[f], we get a set of displacements, $\underline{x}j$, which will lead to a better estimate of the solution: $\underline{x}j^{(2)} = \underline{x}j^{(1)} + \underline{x}j$ provided [J] is non-singular. The process is repeated with the new estimate, $\underline{x}j^{(2)}$, $\underline{x}j^{(3)}$,... and so forth until either the residuals or displacements are sufficiently small in which case, the geometry is consistent with the constraint equations.

APPENDIX D - DOOLITTLE'S METHOD

Doolittle's method solves a system of linear equation, $A\underline{x} = \underline{b}$. In the DIMENSION system, Doolittle's method decomposes the Jacobian matrix, J into an upper triangular matrix, U, and a lower triangular matrix, L, and solving by backsubstitution the systems $L\underline{y} = -\underline{f}$ and $U\underline{dx} = \underline{y}$ for $\underline{y}$ and $\underline{dx}$ respectively.

Denoting the components of J, L, and U by Aij, Lij, and Uij respectively, where $1 \leq i, j \leq n$, $Lij = 0$ for $i < j$, $Lii = 1$ for $1 \leq i \leq n$, and $Uij = 0$ for $i > j$, Doolittle's algorithm can be defined in scalar form as follows. To obtain factors L and U of J for any given m, m = 1,2,...,n:

$$Lim = (Aim - \sum_{q=1}^{m-1} Liq\ Uqm) \ / \ Umm \qquad (D.1)$$

for i = m+1,...,n

$$Umj = Amj - \sum_{q=1}^{m-1} Lmq\ Uqj \qquad (D.2)$$

for j = m,m+1,...,n

In backsubstitution:

$$Yi = -fi - \sum_{q=1}^{i-1} Liq\ Yq \qquad (D.3)$$

for $i = 1,2,\ldots,n$

$$dXi = (Yi - \sum_{q=i+1}^{n} Uiq\ Xq)\ /\ Uii \qquad (D.4)$$

for $i = n,n-1,\ldots,1$

APPENDIX E - SPARSE MATRIX METHODS

E.1 GENERAL THEORY

Matrices having a small percentage of non-zeros are called sparse. One indicator of the sparsity of a matrix is S = number of non-zeros / n^2 where n is the order of the matrix. For the matrices used in the numerical methods of variational geometry, S is often $\leq$.Ø5. It is therefore advantageous from a storage and computational point of view to implement sparse matrix techniques.

In the Newton-Raphson method,

$$A \underline{x} = \underline{b} \qquad (E.1)$$

where A is a non-singular real n X n sparse matrix, $\underline{x}$ a 1 X n vector of variables, and $\underline{b}$ a vector of constraints.

One method by which this can be accomplished is to change the original matrix into lower block triangular form by a set of row permutations, P, and column permutations, Q, such that

$$(PAQ) \cdot Q^T \underline{x} = P\underline{b} \qquad (E.2)$$

A permutation matrix is a square matrix obtained by interchanging rows or columns of the unit matrix.

$$PAQ = \begin{bmatrix} A11 & & & \\ & A22 & & \\ & & . & \\ Aij & & & \\ & & & Aii \end{bmatrix} \qquad (E.3)$$

Aii are the diagonal blocks which cannot be further reduced by row and column permutations. The off diagonal blocks, Aij (j<i), are rectangular.

To find P and Q, the first step is to find a row permutation, P1, so that the entries of the diagonal of A are all non-zero, $\hat{A}$ = P1 A. Then, a symmetric permutation is found to reduce $\hat{A}$ to lower block triangular form, $Q^T P1AQ$, where $Q^T P1 = P$.

The large system of equations is now replaced by several smaller systems. The smaller submatrices can now be inverted using a variation of Gaussian elimination. Since the diagonal blocks are much smaller than the whole matrix, inversion proceeds faster and the growth of non-zeros are

limited to the diagonal blocks only.

Performing a LU decomposition within each diagonal block, Aii,

$$PiAiiQi = LiUi \quad i = 1,2,\ldots,k$$

$$P'AQ' = \begin{bmatrix} L1U1 & & & \\ & L2U2 & & \\ & & LiUi & \\ Aij & & & \ddots \\ & & & LkUk \end{bmatrix} \qquad (E.4)$$

where Li is unit lower triangular and Ui is upper triangular.

The solution of the system can be found by block forward substitution.

$$\begin{aligned} A11x1 &= b1 \\ A22x2 &= b2 - A21x1 \\ &\cdots \\ Aiixi &= bi - \sum_{q=1}^{i-1} Aiq\, xq \quad i=1,2,\ldots,k \end{aligned} \qquad (E.5)$$

E.2 HARWELL ROUTINES

HARWELL has produced a subroutine package which solves a set of sparse linear equations. The subroutine package can be used in the matrix inversion step of the Newton-Raphson method. Single as well as double precision can be obtained. The set of subroutines used are illustrated in Figure E-1.

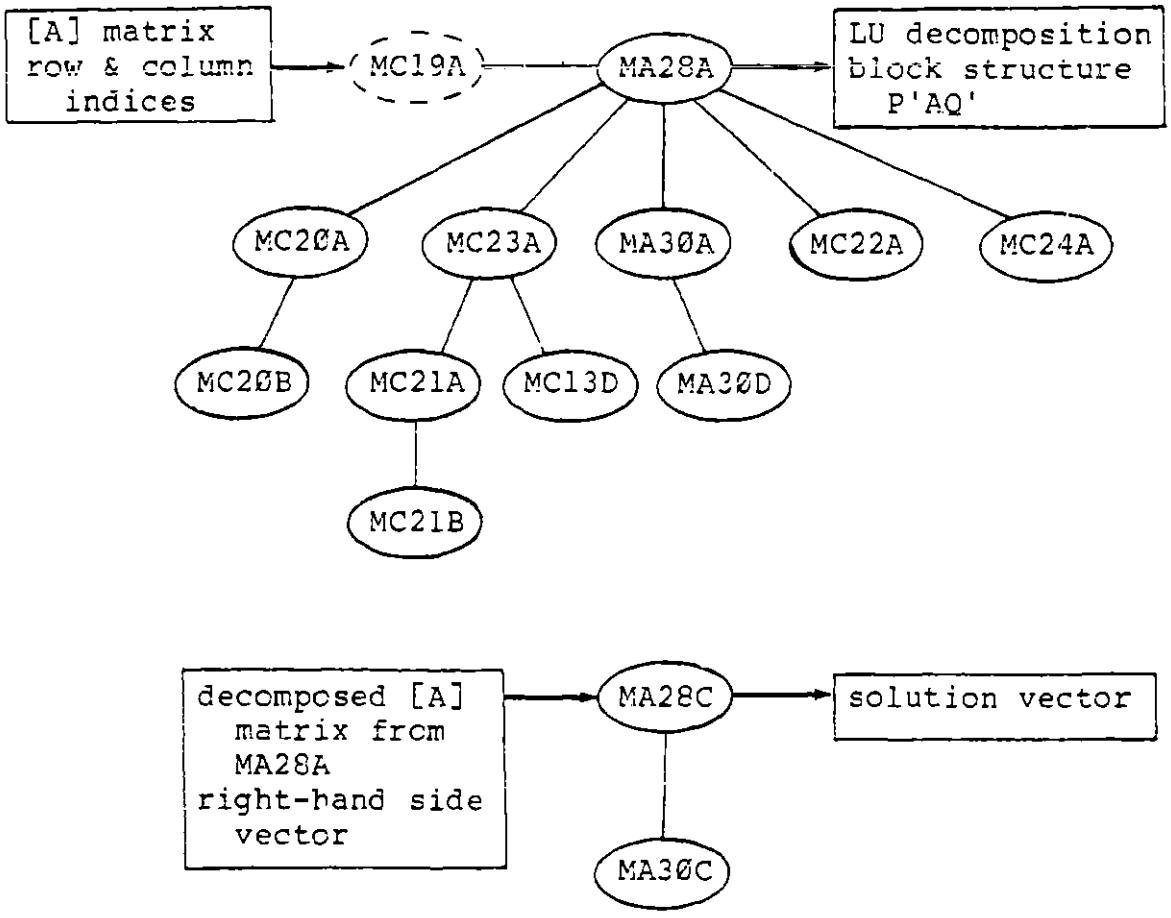


Figure E-1, HARWELL program structure.

MA28

This routine is a data management routine which calls other routines to do block triangularization and LU decomposition. This routine takes as input the matrix, A, and the row and column indices of matrix A. The row and column indices can be arranged in any order.

MC20A AND MC20B

These subroutines reorders the matrix, A to row order form.

MC23A

This subroutine is a data handling routine which calls other routines to perform lower block triangularization.

MC21A AND MC21B

These routines finds a row permutation, P1, which puts non-zeros on the diagonal.

MC13D

This subroutine finds a symmetric permutation, P and Q to permute the matrix outputted from MC21A to block lower triangular form. The routine then reorganizes and reorders

the matrix according to the row and column permutations, P and Q.

MA28C

This subroutine is for the purpose of interfacing with MA30C. It takes as input the decomposed A matrix from MA28A and the right hand side vector. It outputs the solution vector, $\underline{x}$.

MA30C

This subroutine solves $A\underline{x} = \underline{b}$ or $A\,\underline{x} = \underline{b}$, given the decomposition from MA30A by:

1. performing forward elimination corresponding to rows of L1 (the first diagonal block).
2. performing backsubstitution using appropriate elements of U1.
3. repeat steps 1. and 2. for subsequent blocks.

E.3 STORAGE AND TIME REQUIREMENTS

For a matrix of order n, with m non-zeros in the original matrix and r non-zeros in the decomposed matrix, HARWELL estimates the time and storage requirements for routines MA28A and MA28C (on an IBM 370/168).

	MA28A	MA28C
STORAGE (bytes)	42n + 32m	26n + 30m
TIME (microsec)	25/4 r^2/n	4r

An estimate for the value of r is 5/2 m.

The number of Fortran statements require 37K bytes of storage for the compiled code. This package of routines is therefore too large to be run on a minicomputer without using overlays.

E.4 IMPLEMENTATION OF ROUTINES

Interface to the HARWELL routines consists of a call to subroutine MA28A and a call to MA28C. These two subroutines call all the other routines referenced.

For MA28A, the calling statement is as follows:

```
CALL MA28A (N,NZ,A,LICN,IRN,LIRN,ICN,U,IKEEP,IW,W,IFLAG)
```

	ENTRY	EXIT
N [int*4 var]	set = n, the order of the matrix	same
NZ [int*4 var]	set = m, the number of non-zeros in A	same
A(LICN) [real*4 array]	non-zeros of A	non-zeros of factors of A
LICN [int*4 var]	set = 3m, length of A and ICN	same
IRN(LIRN) [int*2 array]	row indices of non-zeros of A	changed
LIRN [int*4 var]	set = 1.5m, length of IRN	same
ICN(LICN) [int*2 array]	column indices of non-zeros in A	column indices of factors of A
U	set = Ø.1Ø, controls	same

[real*4 var]	choice of pivots	
IKEEP(5N) [int*2 array]	internal process	internal process
IW(5N) [int*4 array]	workspace	workspace
W(N) [real*4 array]	workspace	W(1)-largest element encountered during LU
IFLAG [real*4 array]	error flag	Ø if successful

On exit from MA28A, IFLAG should be checked against the error codes. The calling statement to MA28C is as follows:

```
CALL MA28C(N,A,LICN,ICN,IKEEP,RHS,W,MTYPE)
```

	ENTRY	EXIT
N [int*4 var]	same as MA28A output	same
A(LICN) [real*4 array]	same as MA28A output	same
LICN [int*4 var]	same as MA28A output	same
ICN(LICN) [int*2 array]	same as MA28A output	same
IKEEP(5N) [int*2 array]	same as MA28A output	same

RHS(N) [real*4 array]	right hand side	solution vector
W(N) [real*4 array]	workspace	workspace
MYTYPE [int*4 var]	set=1 if solve $A\underline{x}=\underline{b}$ set=0 if solve $A^{-}\underline{x}=\underline{b}$	same

To scale the A matrix, lines of code are added and a call to subroutine MC19A is made before calling MA28A.

APPENDIX F - CONSTRAINT SUBROUTINES

```
      SUBROUTINE RBTX (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR THE
C  CONSTRAINT OF RIGID BODY TRANSLATION IN THE X DIRECTION.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3) - 2
      X1 = XVECT(I)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -X1
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  PARTIALS.
C-----------------------------------------------------------------------
100   II = 0
      DO 10 JJ = 1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = 1.
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE=NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE RBTY (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR THE
C  CONSTRAINT OF RIGID BODY TRANSLATION IN THE Y DIRECTION.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3) - 1
      Y1 = XVECT(I)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -Y1
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  PARTIALS.
C-----------------------------------------------------------------------
100   II = 0
      DO 10 JJ=1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = 1.
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE = NTELE + 1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE RBTZ (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR THE
C  CONSTRAINT OF RIGID BODY TRANSLATION IN THE Z DIRECTION.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3)
      Z1 = XVECT(I)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -Z1
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  PARTIALS.
C-----------------------------------------------------------------------
100   II = 0
      DO 10 JJ=1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = 1.
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE=NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```

```
        SUBROUTINE RBRX (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR THE
C  CONSTRAINT OF RIGID BODY ROTATION IN THE X DIRECTION.
C-----------------------------------------------------------------------
C
        INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
        INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
        INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
        I = 3 * CONSTR(KONS,3)
        Z1 = XVECT(I)
        IF (LCODE .EQ. Ø) GO TO 1ØØ
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
        RESID(NROW) = -Z1
        IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  PARTIALS.
C-----------------------------------------------------------------------
1ØØ     II = Ø
        DO 1Ø JJ=1,NEQS
           AMAT = Ø.
           IF (NVAR(JJ) .EQ. Ø) GO TO 1Ø
           II = II + 1
           IF (JJ .EQ. I) AMAT = 1.
C
           IF (ABS(AMAT) .GT. .Ø2) THEN
              NTELE=NTELE+1
              AVAL(NTELE) = AMAT
              IRN(NTELE) = NROW
              ICN(NTELE) = II
           END IF
1Ø         CONTINUE
C
        RETURN
        END
```

```
      SUBROUTINE RBRY (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR THE
C  CONSTRAINT OF RIGID BODY ROTATION IN THE Y DIRECTION.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3)
      Z1 = XVECT(I)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -Z1
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  PARTIALS.
C-----------------------------------------------------------------------
100   II = 0
      DO 10 JJ=1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = 1.
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE=NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE RBRZ (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR THE
C  CONSTRAINT OF RIGID BODY ROTATION IN THE Z DIRECTION.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3) - 1
      Y1 = XVECT(I)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -Y1
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  PARTIALS.
C-----------------------------------------------------------------------
100   II = 0
      DO 10 JJ=1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = 1.
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE=NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE ANG2P (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF ANGLE BETWEEN INTERSECTING PLANES.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
      DATA PI /3.1415927/
C
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      K = 3 * CONSTR(KONS,5) - 2
      L = 3 * CONSTR(KONS,6) - 2
      M = 3 * CONSTR(KONS,7) - 2
      N = 3 * CONSTR(KONS,8) - 2
      X1 = XVECT(I)
      Y1 = XVECT(I+1)
      Z1 = XVECT(I+2)
      X2 = XVECT(J)
      Y2 = XVECT(J+1)
      Z2 = XVECT(J+2)
      X3 = XVECT(K)
      Y3 = XVECT(K+1)
      Z3 = XVECT(K+2)
      X4 = XVECT(L)
      Y4 = XVECT(L+1)
      Z4 = XVECT(L+2)
      X5 = XVECT(M)
      Y5 = XVECT(M+1)
      Z5 = XVECT(M+2)
      X6 = XVECT(N)
      Y6 = XVECT(N+1)
      Z6 = XVECT(N+2)
C
      A1 =  Y2*Z3 - Y2*Z1 - Y1*Z3 - Y3*Z2 + Y3*Z1 + Y1*Z2
      B1 = -X2*Z3 + X2*Z1 + X1*Z3 + X3*Z2 - X3*Z1 - X1*Z2
      C1 =  X2*Y3 - X2*Y1 - X1*Y3 - X3*Y2 + X3*Y1 + X1*Y2
      A2 =  Y5*Z6 - Y5*Z4 - Y4*Z6 - Y6*Z5 + Y6*Z4 + Y4*Z5
      B2 = -X5*Z6 + X5*Z4 + X4*Z6 + X6*Z5 - X6*Z4 - X4*Z5
      C2 =  X5*Y6 - X5*Y4 - X4*Y6 - X6*Y5 + X6*Y4 + X4*Y5
      SQT1 = SQRT(A1*A1 + B1*B1 + C1*C1)
      SQT2 = SQRT(A2*A2 + B2*B2 + C2*C2)
      THETA = DIM(CONSTR(KONS,2),3)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -1*(A1*A2 + B1*B2 + C1*C2 - SQT1*SQT2*COS(PI-THETA))
      IF (LCODE .EQ. 1) RETURN
```

```
C-----------------------------------------------------------------------
C  ONLY CALCULATE PARTIALS FOR SENSITIVE CONSTRAINTS.
C-----------------------------------------------------------------------
100      D = COS(PI - THETA)*SQT2*.5/SQT1
         E = COS(PI - THETA)*SQT1*.5/SQT2
         II = 0
         DO 10 JJ=1,NEQS
            AMAT = 0.
            IF (NVAR(JJ) .EQ. 0) GO TO 10
            II = II + 1
            IF (JJ .EQ. I) AMAT = AMAT + B2*(Z3-Z2) +
        1       C2*(Y2-Y3) - D * (2*B1*(Z3-Z2) + 2*C1*(Y2-Y3))
            IF (JJ .EQ. I+1) AMAT = AMAT + A2*(Z2-Z3) +
        1       C2*(X3-X2) -D * (2*A1*(Z2-Z3) + 2*C1*(X3-X2))
            IF (JJ .EQ. I+2) AMAT = AMAT + A2*(Y3-Y2) +
        1       B2*(X2-X3) -D * (2*A1*(Y3-Y2) + 2*B1*(X2-X3))
            IF (JJ .EQ. J) AMAT=AMAT + B2*(Z1-Z3) +
        1       C2*(Y3-Y1) - D * (2*B1*(Z1-Z3) + 2*C1*(Y3-Y1))
            IF (JJ .EQ. J+1) AMAT=AMAT + A2*(Z3-Z1) +
        1       C2*(X1-X3) -D * (2*A1*(Z3-Z1) + 2*C1*(X1-X3))
            IF (JJ .EQ. J+2) AMAT=AMAT + A2*(Y1-Y3) +
        1       B2*(X3-X1) -D * (2*A1*(Y1-Y3) + 2*B1*(X3-X1))
            IF (JJ .EQ. K) AMAT=AMAT + B2*(Z2-Z1) +
        1       C2*(Y1-Y2) - D * (2*B1*(Z2-Z1) + 2*C1*(Y1-Y2))
            IF (JJ .EQ. K+1) AMAT=AMAT + A2*(Z1-Z2) +
        1       C2*(X2-X1) -D * (2*A1*(Z1-Z2) + 2*C1*(X2-X1))
            IF (JJ .EQ. K+2) AMAT=AMAT + A2*(Y2-Y1) +
        1       B2*(X1-X2) -D * (2*A1*(Y2-Y1) +2*B1*(X1-X2))
            IF (JJ .EQ. L) AMAT=AMAT + B1*(Z6-Z5) +
        1       C1*(Y5-Y6) - E * (2*B2*(Z6-Z5) + 2*C2*(Y5-Y6))
            IF (JJ .EQ. L+1) AMAT=AMAT + A1*(Z5-Z6) +
        1       C1*(X6-X5) -E * (2*A2*(Z5-Z6) + 2*C2*(X6-X5))
            IF (JJ .EQ. L+2) AMAT=AMAT + A1*(Y6-Y5) +
        1       B1*(X5-X6) -E * (2*A2*(Y6-Y5) + 2*B2*(X5-X6))
            IF (JJ .EQ. M) AMAT=AMAT + B1*(Z4-Z6) +
        1       C1*(Y6-Y4) - E * (2*B2*(Z4-Z6) + 2*C2*(Y6-Y4))
            IF (JJ .EQ. M+1) AMAT=AMAT + A1*(Z6-Z4) +
        1       C1*(X4-X6) -E * (2*A2*(Z6-Z4) + 2*C2*(X4-X6))
            IF (JJ .EQ. M+2) AMAT=AMAT + A1*(Y4-Y6) +
        1       B1*(X6-X4) -E * (2*A2*(Y4-Y6) + 2*B2*(X6-X4))
            IF (JJ .EQ. N) AMAT=AMAT + B1*(Z5-Z4) +
        1       C1*(Y4-Y5) - E * (2*B2*(Z5-Z4) + 2*C2*(Y4-Y5))
            IF (JJ .EQ. N+1) AMAT=AMAT + A1*(Z4-Z5) +
        1       C1*(X5-X4) -E * (2*A2*(Z4-Z5) + 2*C2*(X5-X4))
            IF (JJ .EQ. N+2) AMAT=AMAT + A1*(Y5-Y4) +
        1       B1*(X4-X5) -E * (2*A2*(Y5-Y4) + 2*B2*(X4-X5))
C
            IF (ABS(AMAT) .GT. .02) THEN
               NTELE = NTELE + 1
               AVAL(NTELE) = AMAT
               IRN(NTELE) = NROW
```

```
               ICN(NTELE) = II
            END IF
10          CONTINUE
C
         RETURN
         END
```

```
      SUBROUTINE COPLAN (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES
C  FOR THE CONSTRAINT OF FOUR POINTS COPLANAR.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      K = 3 * CONSTR(KONS,5) - 2
      L = 3 * CONSTR(KONS,6) - 2
      X1 = XVECT(I)
      Y1 = XVECT(I+1)
      Z1 = XVECT(I+2)
      X2 = XVECT(J)
      Y2 = XVECT(J+1)
      Z2 = XVECT(J+2)
      X3 = XVECT(K)
      Y3 = XVECT(K+1)
      Z3 = XVECT(K+2)
      X4 = XVECT(L)
      Y4 = XVECT(L+1)
      Z4 = XVECT(L+2)
C
      A1 = Y2*Z3 - Y2*Z1 - Y1*Z3 - Y3*Z2 + Y3*Z1 + Y1*Z2
      B1 = -X2*Z3 + X2*Z1 + X1*Z3 + X3*Z2 - X3*Z1 - X1*Z2
      C1 = X2*Y3 - X2*Y1 - X1*Y3 - X3*Y2 + X3*Y1 + X1*Y2
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      D = -X1*A1 - Y1*B1 - Z1*C1
      RESID(NROW) = -1* (A1*X4 + B1*Y4 + C1*Z4 + D)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY CALCULATE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------------
100   II = 0
      DO 10 JJ=1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = Y4*Z3 - Y4*Z2 - Y3*Z4 + Y2*Z4
     1        - A1 - Y1*Z3 + Y1*Z2 + Y3*Z1 - Y2*Z1
         IF (JJ .EQ. I+1) AMAT = -X4*Z3 + X4*Z2 - X2*Z4 + X3*Z4
     1          + X1*Z3 - X1*Z2 - B1 + Z1*X2 - Z1*X3
         IF (JJ .EQ. I+2) AMAT = -Y2*X4 + Y3*X4 + X2*Y4 - X3*Y4
     1          + X1*Y2 - X1*Y3 - Y1*X2 + Y1*X3 - C1
```

```
            IF (JJ .EQ. J) AMAT = -Z3*Y4 + Z1*Y4 + Y3*Z4 - Y1*Z4
           1           + Y1*Z3 - Z1*Y3
            IF (JJ .EQ. J+1) AMAT = Z3*X4 - Z1*X4 - X3*Z4 + X1*Z4
           1           - X1*Z3 + Z1*X3
            IF (JJ .EQ. J+2) AMAT = -Y3*X4 + Y1*X4 + X3*Y4 -X1*Y4
           1           + X1*Y3 - Y1*X3
            IF (JJ .EQ. K) AMAT = Z2*Y4 - Z1*Y4 - Y2*Z4 + Y1*Z4
           1           - Y1*Z2 + Z1*Y2
            IF (JJ .EQ. K+1) AMAT = -Z2*X4 + Z1*X4 + X2*Z4 - X1*Z4
           1          + X1*Z2 - Z1*X2
            IF (JJ .EQ. K+2) AMAT = Y2*X4 - Y1*X4 - X2*Y4 + X1*Y4
           1          - X1*Y2 + Y1*X2
            IF (JJ .EQ. L) AMAT = A1
            IF (JJ .EQ. L+1) AMAT = B1
            IF (JJ .EQ. L+2) AMAT = C1
C
            IF (ABS(AMAT) .GT. .Ø2) THEN
               NTELE = NTELE + 1
               AVAL(NTELE) = AMAT
               IRN(NTELE) = NROW
               ICN(NTELE) = II
            END IF
1Ø          CONTINUE
C
        RETURN
        END
```

```
      SUBROUTINE PARALP (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  CONSTRAINT OF A LINE PARALLEL TO A PLANE.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      K = 3 * CONSTR(KONS,5) - 2
      L = 3 * CONSTR(KONS,6) - 2
      M = 3 * CONSTR(KONS,7) - 2
      X1 = XVECT(I)
      Y1 = XVECT(I+1)
      Z1 = XVECT(I+2)
      X2 = XVECT(J)
      Y2 = XVECT(J+1)
      Z2 = XVECT(J+2)
      X3 = XVECT(K)
      Y3 = XVECT(K+1)
      Z3 = XVECT(K+2)
      X4 = XVECT(L)
      Y4 = XVECT(L+1)
      Z4 = XVECT(L+2)
      X5 = XVECT(M)
      Y5 = XVECT(M+1)
      Z5 = XVECT(M+2)
C
      A1 = Y2*Z3 - Y2*Z1 - Y1*Z3 - Y3*Z2 + Y3*Z1 + Y1*Z2
      B1 = -X2*Z3 + X2*Z1 + X1*Z3 + X3*Z2 - X3*Z1 - X1*Z2
      C1 = X2*Y3 - X2*Y1 - X1*Y3 - X3*Y2 + X3*Y1 + X1*Y2
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -1 * ((X5-X4)*A1 + (Y5-Y4)*B1 + (Z5-Z4)*C1)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------------
100   II = 0
      DO 10 JJ=1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
      IF (JJ .EQ. I) AMAT  =  (Y5-Y4)*(Z3-Z2) + (Z5-Z4)*(Y2-Y3)
      IF (JJ .EQ. I+1) AMAT = (X5-X4)*(Z2-Z3) + (Z5-Z4)*(X3-X2)
      IF (JJ .EQ. I+2) AMAT = (X5-X4)*(Y3-Y2) + (Y5-Y4)*(X2-X3)
```

```
      IF (JJ .EQ. J) AMAT  =  (Y5-Y4)*(Z1-Z3) + (Z5-Z4)*(Y3-Y1)
      IF (JJ .EQ. J+1) AMAT = (X5-X4)*(Z3-Z1) + (Z5-Z4)*(X1-X3)
      IF (JJ .EQ. J+2) AMAT = (X5-X4)*(Y1-Y3) + (Y5-Y4)*(X3-X1)
      IF (JJ .EQ. K) AMAT  =  (Y5-Y4)*(Z2-Z1) + (Z5-Z4)*(Y1-Y2)
      IF (JJ .EQ. K+1) AMAT = (X5-X4)*(Z1-Z2) + (Z5-Z4)*(X2-X1)
      IF (JJ .EQ. K+2) AMAT = (X5-X4)*(Y2-Y1) + (Y5-Y4)*(X1-X2)
      IF (JJ .EQ. L) AMAT  =  -A1
      IF (JJ .EQ. L+1) AMAT = -B1
      IF (JJ .EQ. L+2) AMAT = -C1
      IF (JJ .EQ. M) AMAT  =  A1
      IF (JJ .EQ. M+1) AMAT = B1
      IF (JJ .EQ. M+2) AMAT = C1
C
      IF (ABS(AMAT) .GT. .Ø2) THEN
         NTELE = NTELE + 1
         AVAL(NTELE) = AMAT
         IRN(NTELE) = NROW
         ICN(NTELE) = II
      END IF
1Ø       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE PERPP (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF PERPENDICULAR PLANES.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      K = 3 * CONSTR(KONS,5) - 2
      L = 3 * CONSTR(KONS,6) - 2
      M = 3 * CONSTR(KONS,7) - 2
      N = 3 * CONSTR(KONS,8) - 2
      X1 = XVECT(I)
      Y1 = XVECT(I+1)
      Z1 = XVECT(I+2)
      X2 = XVECT(J)
      Y2 = XVECT(J+1)
      Z2 = XVECT(J+2)
      X3 = XVECT(K)
      Y3 = XVECT(K+1)
      Z3 = XVECT(K+2)
      X4 = XVECT(L)
      Y4 = XVECT(L+1)
      Z4 = XVECT(L+2)
      X5 = XVECT(M)
      Y5 = XVECT(M+1)
      Z5 = XVECT(M+2)
      X6 = XVECT(N)
      Y6 = XVECT(N+1)
      Z6 = XVECT(N+2)
C
      A1 =  Y2*Z3 - Y2*Z1 - Y1*Z3 - Y3*Z2 + Y3*Z1 + Y1*Z2
      B1 = -X2*Z3 + X2*Z1 + X1*Z3 + X3*Z2 - X3*Z1 - X1*Z2
      C1 =  X2*Y3 - X2*Y1 - X1*Y3 - X3*Y2 + X3*Y1 + X1*Y2
      A2 =  Y5*Z6 - Y5*Z4 - Y4*Z6 - Y6*Z5 + Y6*Z4 + Y4*Z5
      B2 = -X5*Z6 + X5*Z4 + X4*Z6 + X6*Z5 - X6*Z4 - X4*Z5
      C2 =  X5*Y6 - X5*Y4 - X4*Y6 - X6*Y5 + X6*Y4 + X4*Y5
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -1*(A1*A2 + B1*B2 + C1*C2)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY CALCULATE PARTIALS FOR SENSITIVE CONSTRAINTS.
C-----------------------------------------------------------------------
100   II = 0
```

```
      DO 1Ø JJ=1,NEQS
         AMAT = Ø.
         IF (NVAR(JJ) .EQ. Ø) GO TO 1Ø
         II = II + 1
         IF (JJ .EQ. I) AMAT=AMAT+B2*(Z3-Z2) + C2*(Y2-Y3)
         IF (JJ .EQ. I+1) AMAT=AMAT+A2*(Z2-Z3) + C2*(X3-X2)
         IF (JJ .EQ. I+2) AMAT=AMAT+A2*(Y3-Y2) + B2*(X2-X3)
         IF (JJ .EQ. J) AMAT=AMAT+B2*(Z1-Z3) + C2*(Y3-Y1)
         IF (JJ .EQ. J+1) AMAT=AMAT+A2*(Z3-Z1) + C2*(X1-X3)
         IF (JJ .EQ. J+2) AMAT=AMAT+A2*(Y1-Y3) + B2*(X3-X1)
         IF (JJ .EQ. K) AMAT=AMAT+B2*(Z2-Z1) + C2*(Y1-Y2)
         IF (JJ .EQ. K+1) AMAT=AMAT+A2*(Z1-Z2) + C2*(X2-X1)
         IF (JJ .EQ. K+2) AMAT=AMAT+A2*(Y2-Y1) + B2*(X1-X2)
         IF (JJ .EQ. L) AMAT=AMAT+B1*(Z6-Z5) + C1*(Y5-Y6)
         IF (JJ .EQ. L+1) AMAT=AMAT+A1*(Z5-Z6) + C1*(X6-X5)
         IF (JJ .EQ. L+2) AMAT=AMAT+A1*(Y6-Y5) + B1*(X5-X6)
         IF (JJ .EQ. M) AMAT=AMAT+B1*(Z4-Z6) + C1*(Y6-Y4)
         IF (JJ .EQ. M+1) AMAT=AMAT+A1*(Z6-Z4) + C1*(X4-X6)
         IF (JJ .EQ. M+2) AMAT=AMAT+A1*(Y4-Y6) + B1*(X6-X4)
         IF (JJ .EQ. N) AMAT=AMAT+B1*(Z5-Z4) + C1*(Y4-Y5)
         IF (JJ .EQ. N+1) AMAT=AMAT+A1*(Z4-Z5)+ C1*(X5-X4)
         IF (JJ .EQ. N+2) AMAT=AMAT+A1*(Y5-Y4)+ B1*(X4-X5)
C
         IF (ABS(AMAT) .GT. .Ø2) THEN
            NTELE = NTELE + 1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
1Ø       CONTINUE
C
      RETURN
      END
C-----------------------------------------------------------------------
C  NOTE:  SINCE SAME VARIABLE CAN APPEAR MORE THAN ONCE IN THE SAME
C         EQUATION, MUST ADD JACOBIAN ELEMENTS TOGETHER.
C  EG.   I = 2   M = 2   JJ = 2
C         MUST ADD A(NROW,II) OBTAINED FROM I TO A(NROW,II) OBTAINED
C         FROM M TO GET NEW TOTAL A(IROW,II).  MUST DO SAME FOR I+1,
C         I+2, M+1, M+2.
C-----------------------------------------------------------------------
```

```
      SUBROUTINE EQLDIS (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF TWO LINEAR DISTANCES BEING EQUAL.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      K = 3 * CONSTR(KONS,5) - 2
      L = 3 * CONSTR(KONS,6) - 2
      X1 = XVECT(I)
      Y1 = XVECT(I+1)
      Z1 = XVECT(I+2)
      X2 = XVECT(J)
      Y2 = XVECT(J+1)
      Z2 = XVECT(J+2)
      X3 = XVECT(K)
      Y3 = XVECT(K+1)
      Z3 = XVECT(K+2)
      X4 = XVECT(L)
      Y4 = XVECT(L+1)
      Z4 = XVECT(L+2)
      E = SQRT((X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1) + (Z2-Z1)*(Z2-Z1))
      F = SQRT((X4-X3)*(X4-X3) + (Y4-Y3)*(Y4-Y3) + (Z4-Z3)*(Z4-Z3))
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = F - E
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------------
100   G = 1 / E
      H = 1 / F
      II = 0
      DO 10 JJ =1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = AMAT - G*(X2-X1)
         IF (JJ .EQ. I+1) AMAT = AMAT - G*(Y2-Y1)
         IF (JJ .EQ. I+2) AMAT = AMAT - G*(Z2-Z1)
         IF (JJ .EQ. J) AMAT = AMAT + G*(X2-X1)
         IF (JJ .EQ. J+1) AMAT = AMAT + G*(Y2-Y1)
         IF (JJ .EQ. J+2) AMAT = AMAT + G*(Z2-Z1)
         IF (JJ .EQ. K) AMAT = AMAT + H*(X4-X3)
         IF (JJ .EQ. K+1) AMAT = AMAT + H*(Y4-Y3)
```

```
            IF (JJ .EQ. K+2) AMAT = AMAT + H*(Z4-Z3)
            IF (JJ .EQ. L) AMAT = AMAT - H*(X4-X3)
            IF (JJ .EQ. L+1) AMAT = AMAT - H*(Y4-Y3)
            IF (JJ .EQ. L+2) AMAT = AMAT - H*(Z4-Z3)
C
            IF (ABS(AMAT) .GT. .Ø2) THEN
               NTELE=NTELE+1
               AVAL(NTELE) = AMAT
               IRN(NTELE) = NROW
               ICN(NTELE) = II
            END IF
1Ø          CONTINUE
C-------------------------------------------------------------------------
C  NOTE: SINCE SAME VARIABLE CAN APPEAR MORE THAN ONCE IN IN THE
C  SAME EQUATION, MUST ADD JACOBIAN ELEMENTS TOGETHER.
C-------------------------------------------------------------------------
C
         RETURN
         END
```

```
      SUBROUTINE EQXDIS (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF THE TWO EQUAL X DISTANCES.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      K = 3 * CONSTR(KONS,5) - 2
      L = 3 * CONSTR(KONS,6) - 2
      X1 = XVECT(I)
      X2 = XVECT(J)
      X3 = XVECT(K)
      X4 = XVECT(L)
      E = SQRT((X2-X1)*(X2-X1))
      F = SQRT((X4-X3)*(X4-X3))
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = F - E
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------------
100   G = 1 / E
      H = 1 / F
      II = 0
      DO 10 JJ=1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = AMAT - G*(X2-X1)
         IF (JJ .EQ. J) AMAT = AMAT + G*(X2-X1)
         IF (JJ .EQ. K) AMAT = AMAT + H*(X4-X3)
         IF (JJ .EQ. L) AMAT = AMAT - H*(X4-X3)
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE = NTELE + 1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C-----------------------------------------------------------------------
C  NOTE: SINCE SAME VARIABLE CAN APPEAR MORE THAN ONCE IN THE
C        SAME EQUATION, MUST ADD JACOBIAN ELEMENTS TOGETHER.
C-----------------------------------------------------------------------
```

```
C
      RETURN
      END
```

```
      SUBROUTINE DIS2PT (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF THE DISTANCE BETWEEN TWO POINTS IN 3 DIMENSIONS.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      X1 = XVECT(I)
      Y1 = XVECT(I+1)
      Z1 = XVECT(I+2)
      X2 = XVECT(J)
      Y2 = XVECT(J+1)
      Z2 = XVECT(J+2)
      D = SQRT((X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1) + (Z2-Z1)*(Z2-Z1))
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      DIST = DIM(CONSTR(KONS,2),3)
      RESID(NROW) = -1 * (D - DIST)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENEITIVE VARIABLES.
C-----------------------------------------------------------------------
100   E = 1 / D
      II = 0
      DO 10 JJ=1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = -E*(X2-X1)
         IF (JJ .EQ. I+1) AMAT = -E*(Y2-Y1)
         IF (JJ .EQ. I+2) AMAT = -E*(Z2-Z1)
         IF (JJ .EQ. J) AMAT = E*(X2-X1)
         IF (JJ .EQ. J+1) AMAT = E*(Y2-Y1)
         IF (JJ .EQ. J+2) AMAT = E*(Z2-Z1)
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE = NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE DISPTP (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF THE DISTANCE BETWEEN A POINT AND A PLANE.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      K = 3 * CONSTR(KONS,5) - 2
      L = 3 * CONSTR(KONS,6) - 2
      X1 = XVECT(I)
      Y1 = XVECT(I+1)
      Z1 = XVECT(I+2)
      X2 = XVECT(J)
      Y2 = XVECT(J+1)
      Z2 = XVECT(J+2)
      X3 = XVECT(K)
      Y3 = XVECT(K+1)
      Z3 = XVECT(K+2)
      X4 = XVECT(L)
      Y4 = XVECT(L+1)
      Z4 = XVECT(L+2)
C
      A1 = Y2*Z3 - Y2*Z1 - Y1*Z3 - Y3*Z2 + Y3*Z1 + Y1*Z2
      B1 = -X2*Z3 + X2*Z1 + X1*Z3 + X3*Z2 - X3*Z1 - X1*Z2
      C1 = X2*Y3 - X2*Y1 - X1*Y3 - X3*Y2 +X3*Y1 +X1*Y2
      D = -X1*A1 - Y1*B1 - Z1*C1
      SQT = SQRT(A1*A1 + B1*B1 + C1*C1)
      SIGN = 1.
      E = A1*X4 + B1*Y4 + C1*Z4 + D
      IF (E .GE. 0.) SIGN = -1.
      DIST = DIM(CONSTR(KONS,2),3)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      RESID(NROW) = -1 * (E + SIGN*DIST*SQT)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------------
100   F = SIGN*DIST*.5/SQT
      II = 0
      DO 10 JJ = 1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = Y4*(Z3-Z2) + Z4*(Y2-Y3) - A1
```

```
     1      - Y1*(Z3-Z2) - Z1*(Y2-Y3) + F*(2*B1*(Z3-Z2)
     2      + 2*C1*(Y2-Y3))
            IF (JJ .EQ. I+1) AMAT = X4*(Z2-Z3) + Z4*(X3-X2) -
     1      X1*(Z2-Z3) - B1 - Z1*(X3-X2) + F*(2*A1*(Z2-Z3)
     2      + 2*C1*(X3-X2))
            IF (JJ .EQ. I+2) AMAT = X4*(Y3-Y2) + Y4*(X2-X3) -
     1      X1*(Y3-Y2) - Y1*(X2-X3) - C1 + F*(2*A1*(Y3-Y2)
     2      + 2*B1*(X2-X3))
            IF (JJ .EQ. J) AMAT = Y4*(Z1-Z3) + Z4*(Y3-Y1) -
     1      Y1*(Z1-Z3) - Z1*(Y3-Y1) + F*(2*B1*(Z1-Z3)
     2      + 2*C1*(Y3-Y1))
            IF (JJ .EQ. J+1) AMAT = X4*(Z3-Z1) + Z4*(X1-X3)
     1      - X1*(Z3-Z1) - Z1*(X1-X3) + F*(2*A1*(Z3-Z1)
     2      + 2*C1*(X1-X3))
            IF (JJ .EQ. J+2) AMAT = X4*(Y1-Y3) + Y4*(X3-X1)
     1      - X1*(Y1-Y3) - Y1*(X3-X1) + F*(2*A1*(Y1-Y3)
     2      + 2*B1*(X3-X1))
            IF (JJ .EQ. K) AMAT = Y4*(Z2-Z1) + Z4*(Y1-Y2) -
     1      Y1*(Z2-Z1) - Z1*(Y1-Y2) + F*(2*B1*(Z2-Z1)
     2      + 2*C1*(Y1-Y2))
            IF (JJ .EQ. K+1) AMAT = X4*(Z1-Z2) + Z4*(X2-X1) -
     1      X1*(Z1-Z2) - Z1*(X2-X1) + F*(2*A1*(Z1-Z2)
     2      + 2*C1*(X2-X1))
            IF (JJ .EQ. K+2) AMAT = X4*(Y2-Y1) + Y4*(X1-X2) -
     1      X1*(Y2-Y1) - Y1*(X1-X2) + F*(2*A1*(Y2-Y1)
     2      + 2*B1*(X1-X2))
            IF (JJ .EQ. L) AMAT = A1
            IF (JJ .EQ. L+1) AMAT = B1
            IF (JJ .EQ. L+2) AMAT = C1
C
            IF (ABS(AMAT) .GT. .Ø2) THEN
               NTELE = NTELE+1
               AVAL(NTELE) = AMAT
               IRN(NTELE) = NROW
               ICN(NTELE) = II
            END IF
1Ø          CONTINUE
C
            RETURN
            END
```

```
      SUBROUTINE HDIST (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF THE HORIZONTAL DISTANCE BETWEEN TWO POINTS.
C-----------------------------------------------------------------------
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3 * CONSTR(KONS,3) - 2
      J = 3 * CONSTR(KONS,4) - 2
      X1 = XVECT(I)
      Y1 = XVECT(I+1)
      X2 = XVECT(J)
      Y2 = XVECT(J+1)
      D = SQRT((X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1))
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      DIST = DIM(CONSTR(KONS,2),3)
      RESID(NROW) = -1*(D - DIST)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------------
100   E = 1 / D
      II = 0
      DO 10 JJ = 1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I) AMAT = -E*(X2-X1)
         IF (JJ .EQ. I+1) AMAT = -E*(Y2-Y1)
         IF (JJ .EQ. J) AMAT = E*(X2-X1)
         IF (JJ .EQ. J+1) AMAT = E*(Y2-Y1)
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE = NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE VDIST (LCODE,KONS,NROW)
C-----------------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF THE VERTICAL DISTANCE BETWEEN TWO POINTS.
C-----------------------------------------------------------------------
C
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
      I = 3*CONSTR(KONS,3) - 2
      J = 3*CONSTR(KONS,4) - 2
C
      Z1 = XVECT(I+2)
      Z2 = XVECT(J+2)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------------
      DIST = DIM(CONSTR(KONS,2),3)
      RESID(NROW) = -1 * (Z2 - Z1 - DIST)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------------
100   II = 0
      DO 10 JJ= 1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I+2) AMAT = -1.
         IF (JJ .EQ. J+2) AMAT = 1.
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE = NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE XDIST(LCODE,KONS,NROW)
C-----------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF THE X DISTANCE BETWEEN TWO POINTS.
C-----------------------------------------------------------------
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3*CONSTR(KONS,3)-2
      J = 3*CONSTR(KONS,4)-2
      X1 = XVECT(I)
      X2 = XVECT(J)
      IF (LCODE .EQ. Ø) GO TO 1ØØ
C-----------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------
      DIST = DIM(CONSTR(KONS,2),3)
      RESID(NROW) = -1*(X2-X1-DIST)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------
1ØØ   II = Ø
      DO 1Ø JJ = 1,NEQS
         AMAT = Ø.
         IF (NVAR(JJ) .EQ. Ø) GO TO 1Ø
         II = II + 1
         IF (JJ .EQ. I) AMAT = -1.
         IF (JJ .EQ. J) AMAT = 1.
C
         IF (ABS(AMAT) .GT. .Ø2) THEN
            NTELE = NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
1Ø       CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE YDIST(LCODE,KONS,NROW)
C-----------------------------------------------------------------
C  THIS SUBROUTINE COMPUTES THE RESIDUALS AND PARTIAL DERIVATIVES FOR
C  THE CONSTRAINT OF THE Y DISTANCE BETWEEN TWO POINTS.
C-----------------------------------------------------------------
      INCLUDE 'RES1:[P3453.VLIN.COMMON]CONSTRAIN.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]NUMERIC.CMN'
      INCLUDE 'RES1:[P3453.VLIN.COMMON]SEGMENT.CMN'
C
      I = 3*CONSTR(KONS,3)-2
      J = 3*CONSTR(KONS,4)-2
      Y1 = XVECT(I+1)
      Y2 = XVECT(J+1)
      IF (LCODE .EQ. 0) GO TO 100
C-----------------------------------------------------------------
C  RESIDUALS.
C-----------------------------------------------------------------
      DIST = DIM(CONSTR(KONS,2),3)
      RESID(NROW) = -1*(Y2-Y1-DIST)
      IF (LCODE .EQ. 1) RETURN
C-----------------------------------------------------------------
C  ONLY COMPUTE PARTIALS FOR SENSITIVE VARIABLES.
C-----------------------------------------------------------------
100   II = 0
      DO 10 JJ = 1,NEQS
         AMAT = 0.
         IF (NVAR(JJ) .EQ. 0) GO TO 10
         II = II + 1
         IF (JJ .EQ. I+1) AMAT = -1.
         IF (JJ .EQ. J+1) AMAT = 1.
C
         IF (ABS(AMAT) .GT. .02) THEN
            NTELE = NTELE+1
            AVAL(NTELE) = AMAT
            IRN(NTELE) = NROW
            ICN(NTELE) = II
         END IF
10       CONTINUE
C
      RETURN
      END
```