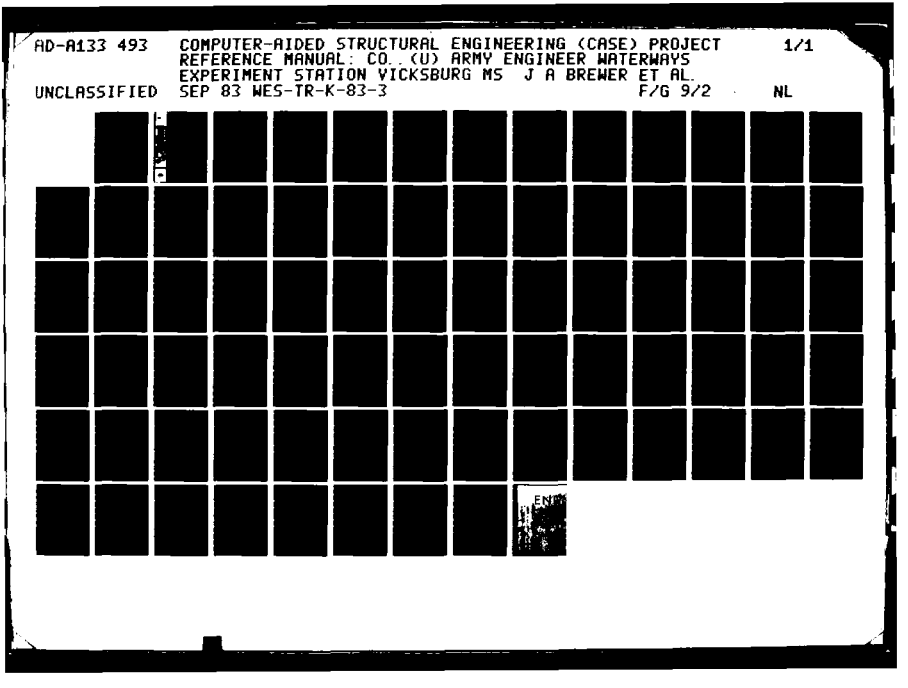
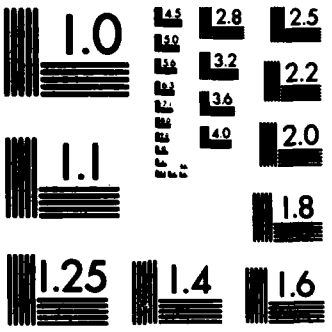

AD-A133 493
COMPUTER-AIDED STRUCTURAL ENGINEERING (CASE) PROJECT REFERENCE MANUAL: CO.. (U) ARMY ENGINEER WATERWAYS EXPERIMENT STATION VICKSBURG MS J A BREWER ET AL.
1/1
UNCLASSIFIED
SEP 83 WES-TR-K-83-3
F/G 9/2
NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

US Army Corps of Engineers

COMPUTER-AIDED STRUCTURAL ENGINEERING (CASE) PROJECT

TECHNICAL REPORT K-83-3

REFERENCE MANUAL: COMPUTER GRAPHICS PROGRAM FOR GENERATION OF ENGINEERING GEOMETRY (SKETCH)

by

John A. Brewer III, Jeffrey N. Jortner
Warren N. Waggenspack, Jr.

Computer Graphics Research & Applications Laboratory
Mechanical Engineering Department
Louisiana State University, Baton Rouge, La. 70803

September 1983
Final Report

Approved For Public Release; Distribution Unlimited

Prepared for Office, Chief of Engineers, U. S. Army
Washington, D. C. 20314

Monitored by Automatic Data Processing Center
U. S. Army Engineer Waterways Experiment Station
P. O. Box 631, Vicksburg, Miss. 39180

Destroy this report when no longer needed. Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

The covers of U. S. Army Engineer Waterways Experiment Station (WES) engineering and scientific reports have been redesigned. Each WES Laboratory and support organization will have its own distinctive color imprinted on white coverstock. This standardizes WES publications and enhances their professional appearance.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: Technical Report K-83-3	2. GOVT ACCESSION NO.: AD-A133 493
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): REFERENCE MANUAL: COMPUTER GRAPHICS PROGRAM FOR GENERATION OF ENGINEERING GEOMETRY (SKETCH)	5. TYPE OF REPORT & PERIOD COVERED: Final report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): John A. Brewer III, Jeffrey N. Jortner, Warren N. Waggenspack, Jr.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Graphics Research & Applications Laboratory, Mechanical Engineering Department, Louisiana State University, Baton Rouge, La. 70803	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office, Chief of Engineers, U. S. Army, Washington, D. C. 20314	12. REPORT DATE: September 1983
	13. NUMBER OF PAGES: 71
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): U. S. Army Engineer Waterways Experiment Station, Automatic Data Processing Center, P. O. Box 631, Vicksburg, Miss. 39180	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Available from National Technical Information Service, Springfield, Va. 22161.

This report was prepared under the Computer-Aided Structural Engineering (CASE) Project. A list of published CASE reports is printed on the inside of the back cover.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer graphics
Computer programs
Engineering mathematics
Geometry
Mathematical models
SKETCH (Computer program)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report provides a reference manual for version 3, modification 1, of SKETCH, a computer program that can be used for generation of engineering geometry. The primary objective of program SKETCH is to assist the user in creation of precise mathematical models of engineering geometry from roughly drawn pictures. A picture or sketch is constructed in an on-line fashion using interactive graphics techniques. The data base which results consists of three-dimensional (3-D) coordinates, lines, polygonal faces, (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

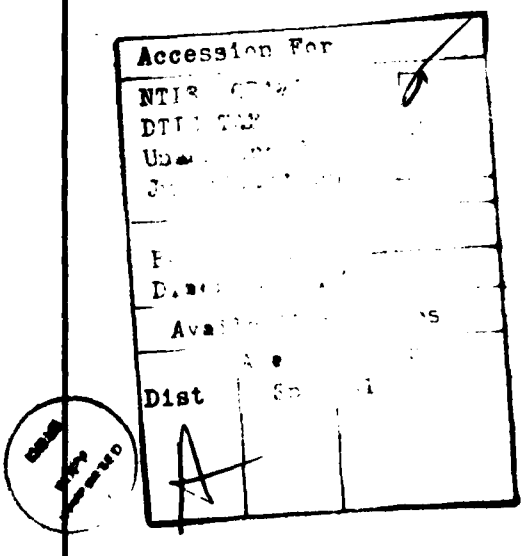
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (Continued).

curves, and surfaces. Curved surfaces, however, are not supported in version 3, modification 1, of SKETCH.

A user of SKETCH first interactively creates a schematic representation of the intended geometry. The schematic can be a two-dimensional (2-D) cross section or a 3-D pictorial. Straight lines alone are used in the initial sketch. Polygons or polyhedra defined by the lines are dimensioned by the user and automatically converted to a 3-D mathematical model by SKETCH. Arcs, circles, ellipses, and splines can be added after the 3-D conversion takes place. The original straight lines therefore serve as control information and boundary conditions for the curve forms.

SKETCH is designed to allow the definition of any engineering geometry, but is particularly advantageous when creating asymmetric geometry. The result of a sketching session is an output file consisting of coordinates, lines, faces, isolated curve segments, and splines. The files can be created or modified with the system editor, if desired.



PREFACE

This is the reference manual for version 3, modification 1, of SKETCH, a computer program that can be used for generation of engineering geometry. The work in writing the computer program and the manual was accomplished with funds provided to the U. S. Army Engineer Waterways Experiment Station (WES), Vicksburg, Miss., by the Civil Works Directorate of the Office, Chief of Engineers, U. S. Army (OCE), under the Computer-Aided Structural Engineering (CASE) Project.

Specifications for the program were provided by the members of the CASE Task Group on 3D Stability. The members of the task group during the period of development (though not all served the entire period) were:

Mr. Charles W. Kling, Mobile District (Chairman)
Mr. Robert Haavisto, Sacramento District
Mr. John Hoffmeister, Nashville District
Mr. William Holtham, New England Division
Mr. Gerrett Johnson, Seattle District
Mr. Tom Leicht, St. Louis District
Mr. Thomas J. Mudd, St. Louis District
Mr. John Tang, Sacramento District

The program was developed at Louisiana State University (LSU) in the Computer Graphics Research and Applications Laboratory by Dr. John A. Brewer III. This manual was written by Dr. Brewer, Mr. Jeffrey N. Jortner, and Mr. Warren N. Waggenspack, Jr. A number of graduate students contributed to the development and debugging of SKETCH and to refinement of the documentation. Mr. Colin B. Selleck worked on the command interpreter, Mr. David E. Wilbanks coded the in-core data management system, and Mr. Waggenspack developed the merge capability. Mr. Jortner worked on the circular arc, circle, quadratic, and cubic curve capabilities. A mechanical engineering undergraduate, Ms. Christine M. Behrmann, helped with the documentation and assisted in debugging the program.

The work was performed under the direction of Dr. N. Radhakrishnan, Special Technical Assistant, Automatic Data Processing (ADP) Center, WES. Mr. Fred T. Tracy, Chief, Research and Development Software Group, ADP Center, served as technical point of contact at WES. Mr. Donald R. Dressler, Structures Division, Civil Works Directorate, was the OCE point of contact.

Commanders and Directors of WES during the preparation and publication

of this report were COL N. P. Conover, CE, and COL T. C. Creel, CE. Technical Director was Mr. F. R. Brown.

CONTENTS

REFERENCE MANUAL: COMPUTER GRAPHICS PROGRAM FOR GENERATION OF ENGINEERING GEOMETRY (SKETCH)

INTRODUCTION

Objectives

The primary objective of program SKETCH is to assist the user in creation of precise mathematical models of engineering geometry from roughly drawn pictures. A picture or sketch is constructed in an on-line fashion using interactive graphics techniques. The data base which results consists of three-dimensional (3-D) coordinates, lines, polygonal faces, curves, and surfaces. Curved surfaces, however, are not supported in version 3, modification 1, of SKETCH.

A user of SKETCH first interactively creates a schematic representation of the intended geometry. The schematic can be a two-dimensional (2-D) cross section or a 3-D pictorial. Straight lines alone are used in the initial sketch. Polygons or polyhedra defined by the lines are dimensioned by the user and automatically converted to a 3-D mathematical model by SKETCH. Arcs, circles, ellipses, and splines can be added after the 3-D conversion takes place. The original straight lines therefore serve as control information and boundary conditions for the curve forms.

SKETCH is designed to allow the definition of any engineering geometry, but is particularly advantageous when creating asymmetric geometry. The result of a sketching session is an output file consisting of coordinates, lines, faces, isolated curve segments, and splines. The files can be created or modified with the system editor, if desired.

Features

SKETCH extensively uses interactive graphics as a communication mode between the user and the computer. In an environment capable of responsive interaction, SKETCH is very effective. Typed commands are processed through a user-friendly command interpreter which preprocesses commands for syntax errors.

Sketched objects using graphic input techniques do not have to be

carefully constructed. Lines which are intended to be parallel to principal axes, for example, can be sketched very inaccurately. Alignment is not a criterion for sketching. Only the general shape and topology need be correct. However, line names are part of the topological information and must be specified correctly.

Although precise alignment is not required, a principal line sketched to within 5% of the proper slope will be automatically named "X", "Y", or "Z". In the case of 2-D cross sections, the line naming problem is solved completely as lines not parallel to the two principal axes involved are assumed to be frontal lines ("F" lines) and are so named by SKETCH. If the program incorrectly labels a line, the user should rename the line using graphic input methods.

All data are managed in a single FORTRAN array by the use of five in-core, data management routines. This memory management method optimizes the use of available primary storage. Multiple objects are managed within the data structure so that complex objects or scenes can be created in a piecemeal fashion. The suggested approach is to complete a set of fully dimensioned subobjects and then merge them (using the MERGE command) into a single object or scene.

FUNDAMENTAL ALGORITHMS

Development of 3-D Models

This section is a description of the logic involved in the automatic creation of a 3-D data base from a rough, 2-D sketch through the use of the SKETCH program environment.

The fundamental user-generated information consists of a table of 2-D coordinates (the geometry) and a table of lines (the topology) which define the intended geometric model. Sketches are created through a menu-driven process with the user selecting various options and entering the data with appropriate graphical input devices.

The user generates a sequence of straight lines and coordinate information which form a polygonal representation of the intended polyhedra. Polygons may be open or closed, and polyhedra may be convex or concave. Also, 2-D cross sections can be sketched and dimensioned. Such data will simply become 3-D data in a common principal plane after automatic conversion to a refined geometric model.

In addition to providing a topological sketch, the user is also required to supply information about the 3-D orientation of lines. He must specify line types for all lines according to their position in three space relative to the principal axes and principal planes. A line can be in one of seven possible positions: parallel to any of three principal axes, parallel to any of three principal planes, or in an oblique position. The naming convention used by SKETCH is as follows:

"X", "Y", "Z" - Refers to the fact that the line is parallel to one of the principal axes, X, Y, or Z, respectively.

"H", "P", "F" - Refers to a Horizontal, Profile, or Frontal line which is parallel to one of the principal planes, XY, XZ, or YZ. A line in one of these three orientations is termed an Inclined line.

"O" - Refers to a line which is not parallel to a principal axis or principal plane. This is an Oblique line.

The topological description plus the line type information provides substantial, but in general not complete, information about the 3-D shape of an object. Ambiguous objects or portions of objects can occur if too many oblique lines or, in some cases, inclined lines are connected.

Dimensioning Planes

Every coordinate in three space can be located at the intersection of three mutually orthogonal planes (Figure 1). If the three planes are respectively perpendicular to principal axes, they are considered principal planes. The class of planes selected is special in that each plane is perpendicular to one of the three coordinate axes, and for this reason each is called a principal plane or "dimensioning" plane.

To locate a single point in three space, all that is required is the location of the respective dimensioning planes. One might wonder why not directly establish the Cartesian components instead? The answer to this lies in the fact that a plane can contain an infinite number of points, and therefore has the ability to establish a single component for a large number of points with the specification of a single value.

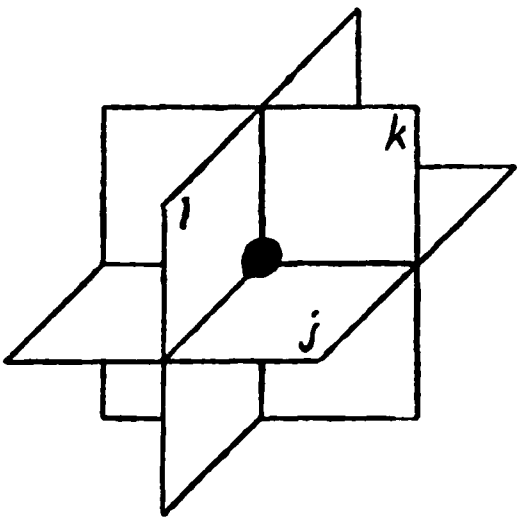


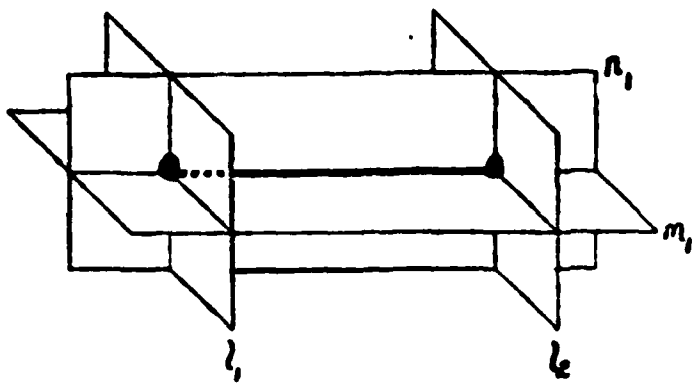
Figure 1. Dimensioning planes at a point

The benefit of the dimensioning planes concept is demonstrated effectively in SKETCH. Before the actual input of dimensions is requested, an attempt is made to assign each coordinate in an object to a triad of dimensioning planes (one in each of the three principal directions). This is accomplished using information contained in the line table.

A single straight-line segment will involve a minimum of 4 to a maximum of 6 dimensioning planes depending on the line type (see Figure 2).

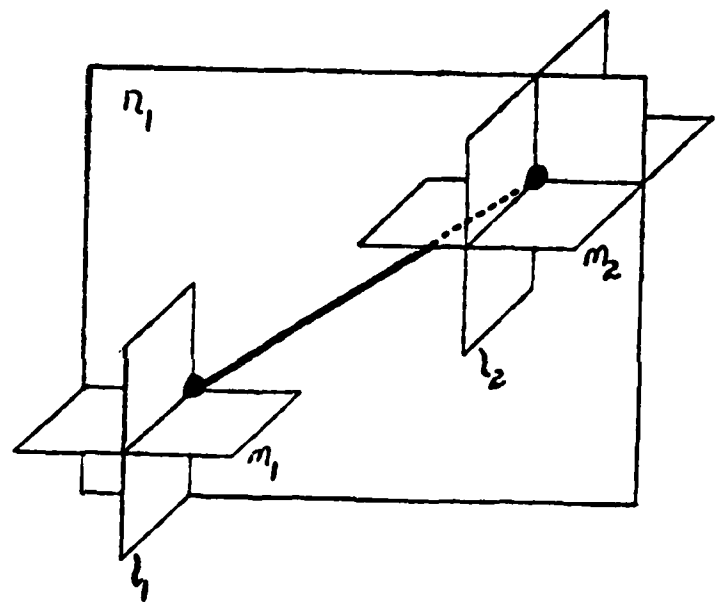
The endpoint of a line parallel to a principal axis (e.g., the X axis) will have the same coordinate values in two or three directions; i.e., Y1 = Y2 and Z1 = Z2. Thus, to locate the relative position of the endpoints of the line, a single incremental value ΔX is required. By similar reasoning, line types H, P, and F require only two incremental values and O lines require three.

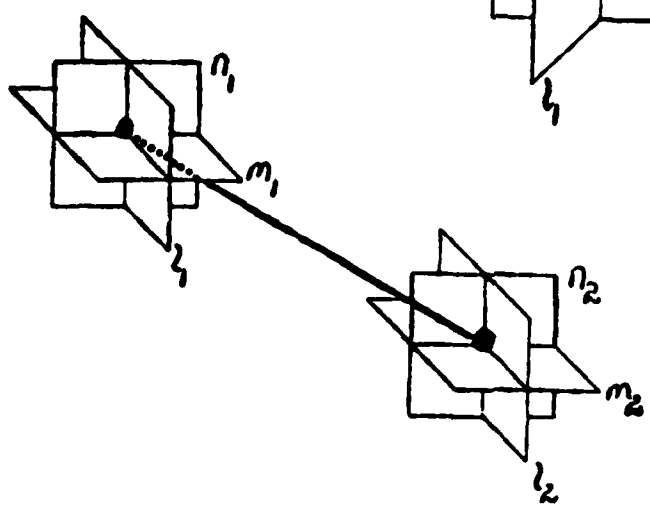
Establishing a single point from which all others may be referenced, SKETCH progresses through the line table and eventually assigns all endpoints to a set of dimensioning planes. It should be pointed out that no



a. Principal line (X, Y, Z)

b. Inclined line (H, P, F)





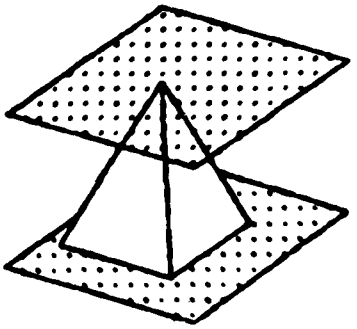
c. Oblique line (O)

Figure 2. Dimensioning planes of the different line types

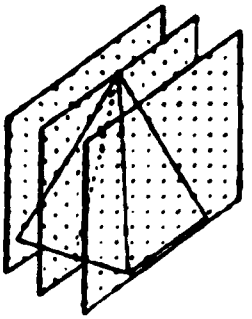
free-standing points are allowed to exist in an object definition; however, they could be accommodated with only minor modifications. In addition to locating the endpoints in a plane, the algorithm also maintains a counter of the required number of dimensioning planes in each of the principal directions as this information is useful later on in the dimensioning process. An example object and the resulting dimensioning planes are shown in Figure 3.

Dimensioning Process

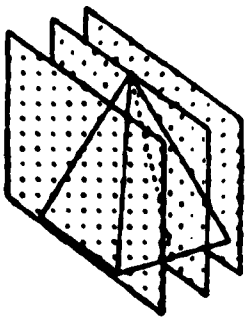
After all coordinates are assigned to the proper dimensioning planes,



Y Dimensional Planes



X Dimensional Planes



Z Dimensional Planes

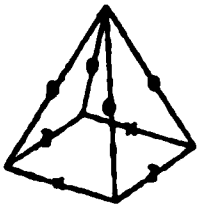


Figure 3. Sketched object with resulting dimensioning planes

the actual specification of dimensions commences. Looking at a single direction (see Figure 4), it is evident that to establish the relative position of the coordinates, the user must specify N - 1 dimensions where N is the number of dimensioning planes in that direction. The actual or absolute coordinate values are set using the coordinate value of the reference plane established by the user upon entering the dimensioning stages of SKETCH.

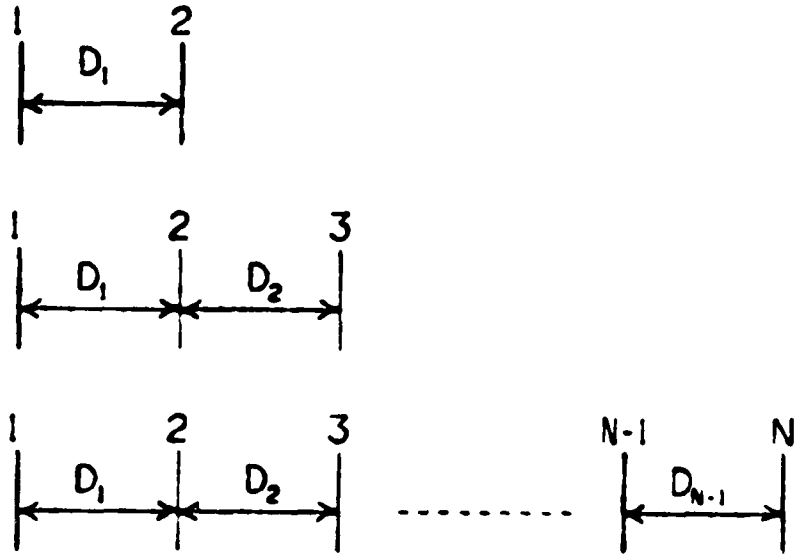


Figure 4. Schematic of the dimensioning plane/ dimension requirement relationship

Extension lines provide another dimensioning aid in SKETCH (see Figure 5).

A single extension line for each dimensioning plane is projected from the first coordinate located that exists within that plane. Extension lines for the X, Y, Z directions are projected parallel to the Z, X, Y directions, respectively.

To enter a dimension, the user locates a pair of extension lines, the first of which is assumed to have a lower relative coordinate value in order for the information to be processed properly. The user is only allowed to specify component dimensions, and a simple check for redundant information is made before accepting a given dimension. There are no other rules governing which dimensions should be given or the order in which they should be specified. A series or counters maintains the number of dimensions given and the number required in each direction.

As the dimensions are accepted, the program builds a set of tables with

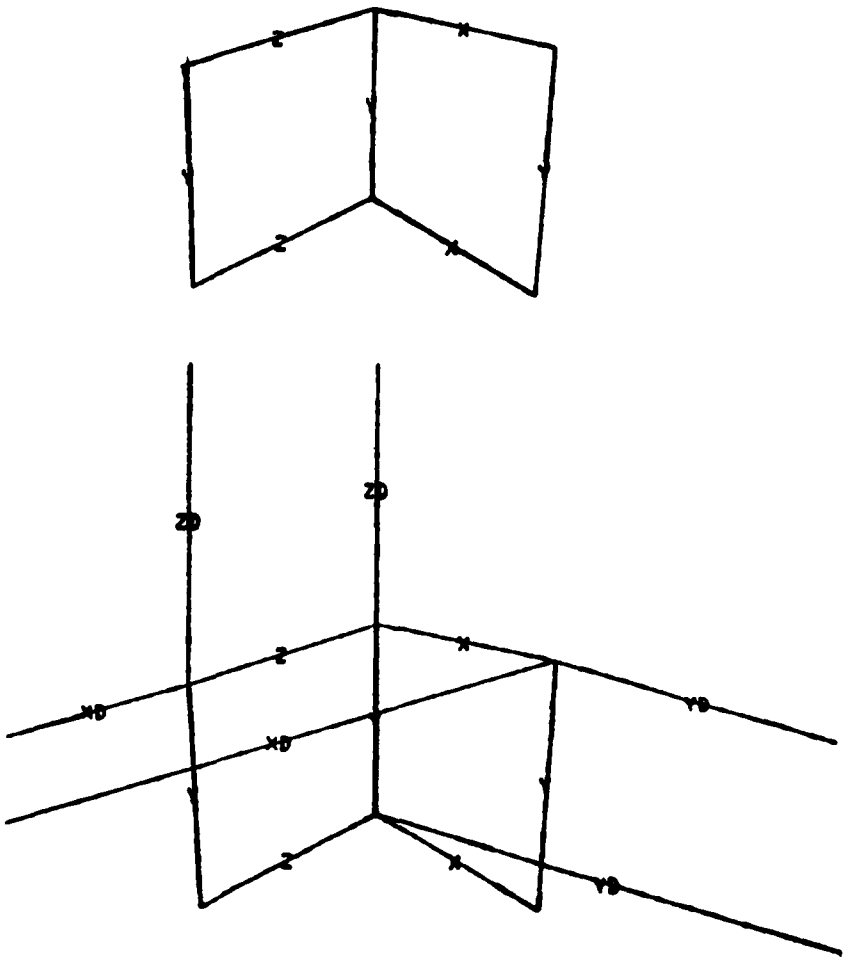


Figure 5. Sketched object with generated extension lines

extensions "XD", "YD", "ZD". These are described more fully in the next chapter under "Elements of Object Definition." When enough information is collected, the user is notified and the resulting tables are passed to another subroutine CVTSTR. This subroutine processes the information in the dimension tables and generates the proper 3-D coordinate values for each of the points of an object.

Working out from the reference plane, dimension values are successively added or subtracted from the reference value. In this manner, the coordinate value of each dimensioning plane is calculated. When this is done, the information is then simply transferred into the coordinate table.

Observation Transformation

The observation transformation is designed to be user-friendly in an

engineering environment. In creating pictorial sketches, the engineer thinks in terms of major axes laid in arbitrary directions, rather than in terms of observers, picture planes, etc. The transformation technique used in SKETCH provides a user interface free of concern for observer vantage points, picture planes, associated scale factors, and rotation angles.

The transformation is fundamentally a shear transformation,

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ \cos\beta & \sin\beta & 0 & 0 \\ \cos\gamma & \sin\gamma & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Principal coordinate axes x, y, and z are mapped into arbitrary positions with angles α, β, and γ defined as shown in Figure 6. Examples using various values for α, β, and γ are given in Figure 7.

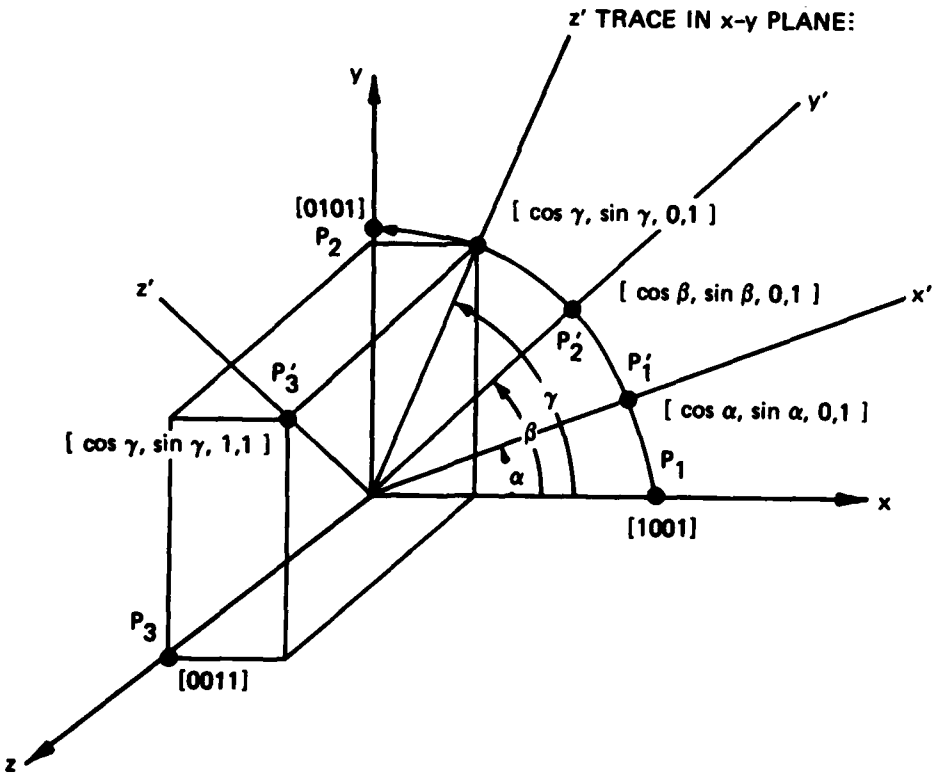


Figure 6. Mapping principal axes into arbitrary positions on the x-y plane

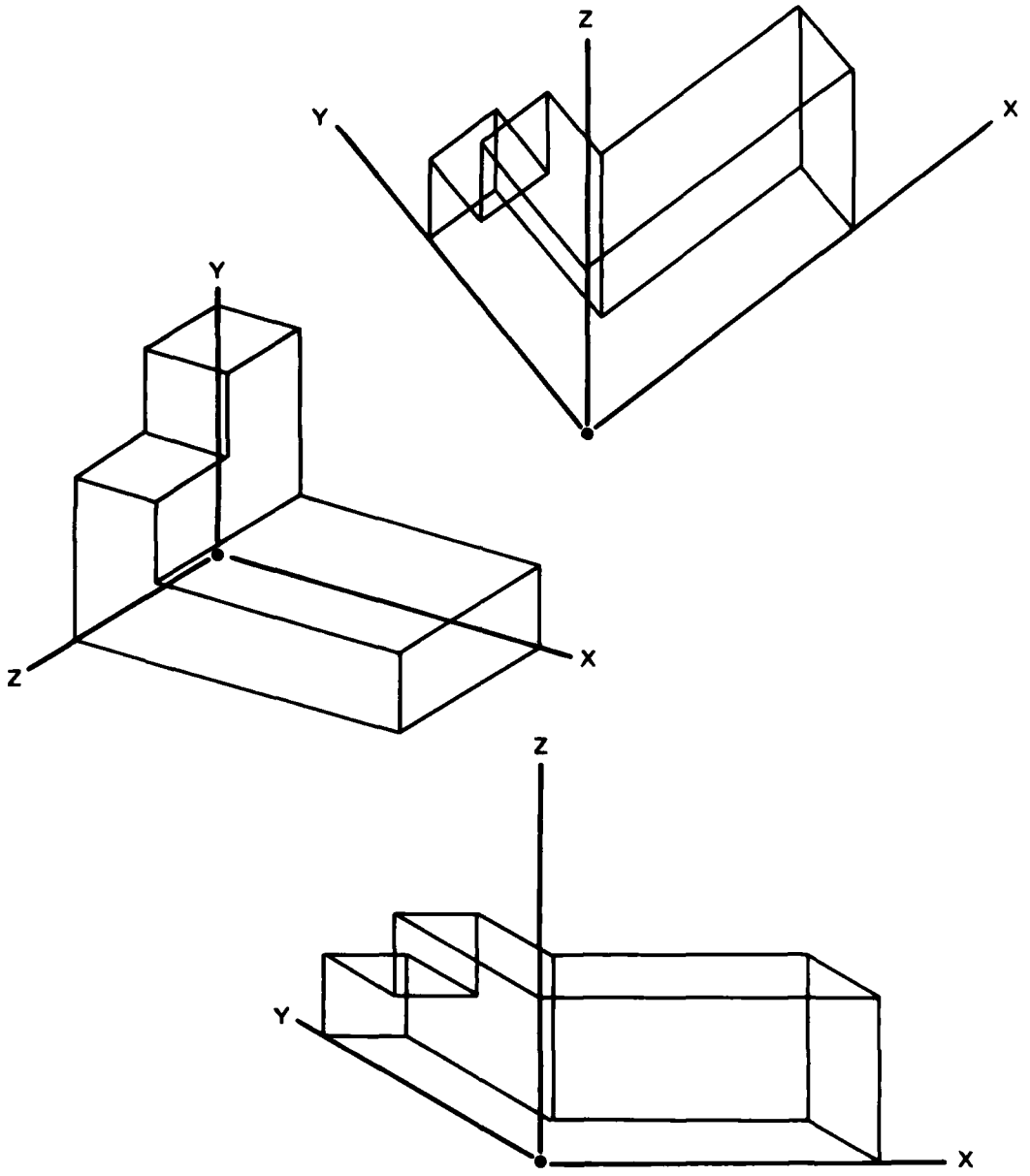


Figure 7. Example views with arbitrary placement of axes

Face Generation

The face generation algorithm is currently limited to creation of planar faces of polyhedra bodies. A candidate object for automatic face generation must be a fully defined 3-D polyhedron. In response to the CREATE FACES or GENERATE FACES command, the algorithm erases the screen, redisplays the object with dashed lines, and proceeds to retrace each possible face, one at a time.

The user is provided the opportunity to reverse the order of sides, to reject, to accept, or to repeat the display of each retraced face.

The algorithm is based on the fact that an edge of a face will not be used more than twice. In other words, a single edge is expected to be common to no more than two faces. The user must, therefore, be careful not to accept an internal polygon, such as the "neck" shown in Figure 8, as an external face. The face generation logic does not distinguish external faces from internal polygons. In fact, a particular internal polygon may be chosen as a face candidate numerous times, and the user should reject such a choice each time it appears.

The order in which the sides of the first face are defined by the algorithm is arbitrary. If the user reverses the order of the face, each face thereafter will be ordered consistent with the first.

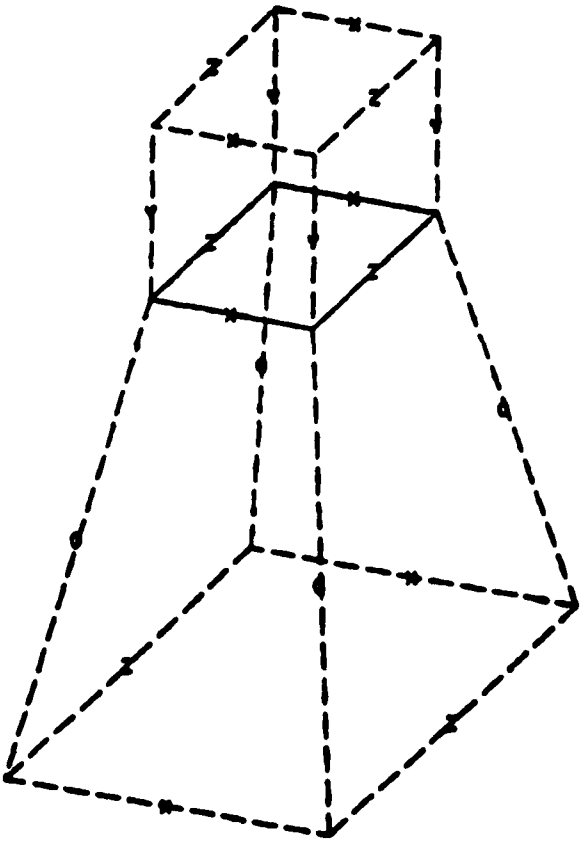


Figure 8. An internal polygon of a polyhedron

Curve Definition

Parametric Quadratic Curve

$$[x\ y\ z] = [t^2\ t\ 1] \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

$$[x\ y\ z] = [t^2\ t\ 1]\ A \qquad (1)$$

where

$$A = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$$

The general matrix form of the quadratic curve is given by Equation 1. The A matrix is found by multiplying a constant matrix and a point matrix together. The three points required are obtained by pointing at two intersecting lines with a common endpoint or pointing at any three points in an object. The A matrix is saved in the matrix table and is not regenerated each time the curve is drawn. To draw the curve, points are generated by varying the parameter t from 0 to 1 using increments of 0.05.

Overhauser Cubic Curve

$$[x\ y\ z\ 1] = [t^3\ t^2\ t\ 1] \begin{bmatrix} a_1 & a_2 & a_3 & 0 \\ b_1 & b_2 & b_3 & 0 \\ c_1 & c_2 & c_3 & 0 \\ d_1 & d_2 & d_3 & 1 \end{bmatrix} = [t^3\ t^2\ t\ 1]\ A \qquad (2)$$

where

$$A = \begin{bmatrix} -1/2 & 3/2 & -3/2 & 1/2 \\ 1 & -5/2 & 2 & -1/2 \\ -1/2 & 0 & 1/2 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{bmatrix}$$

The general matrix form for the curve is given by Equation 2. With this formulation, four points are necessary to define a cubic curve between the second and third points.

Within the SKETCH environment, the four points may be selected arbitrarily in three space or by selecting three contiguously intersecting lines. The

A matrix is calculated by multiplying together the constant matrix and the point matrix. The resulting matrix (A) is then stored in the matrix table and not regenerated each time the curve is drawn. To draw the curve, points are generated by varying the parameter t from 0 to 1 using an incremental value of 0.05.

Circular Arc

A radius and two lines with a common endpoint are specified. The common endpoint is P2 as shown in Figure 9. Unit vectors, A and B, are calculated for each line in the direction away from P2. The unit vector perpendicular to the plane containing vectors A and B is C (see Figure 10). From this vector, two angles, α and β, are found. These angles are the rotations about the X and Y axes, respectively, necessary to orient the curve into the XY planes.

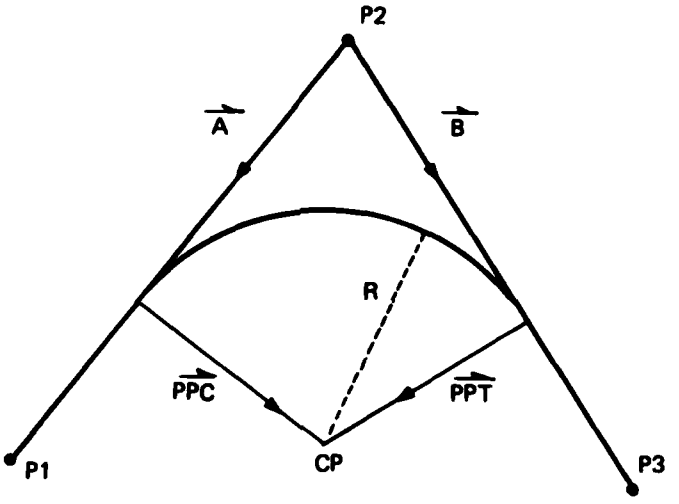


Figure 9. Circular arc definition in the XY plane

Figure 9 shows PPC and PPT which are unit vectors perpendicular to A and B. Using the radius specified, the center point (CP) of the arc is calculated. The curve is then drawn using transformations which effect a rotation about an axis through CP and perpendicular to the plane of the curve.

This transformation is recursively applied to the initial point of the arc, thus producing the points on the arc. The total transformation is composed of several separate transformations:

$$T = -T_{t1} * T_{ry} * T_{rx} * T_{rot} * -T_{rx} * -T_{ry} * T_{t1}$$

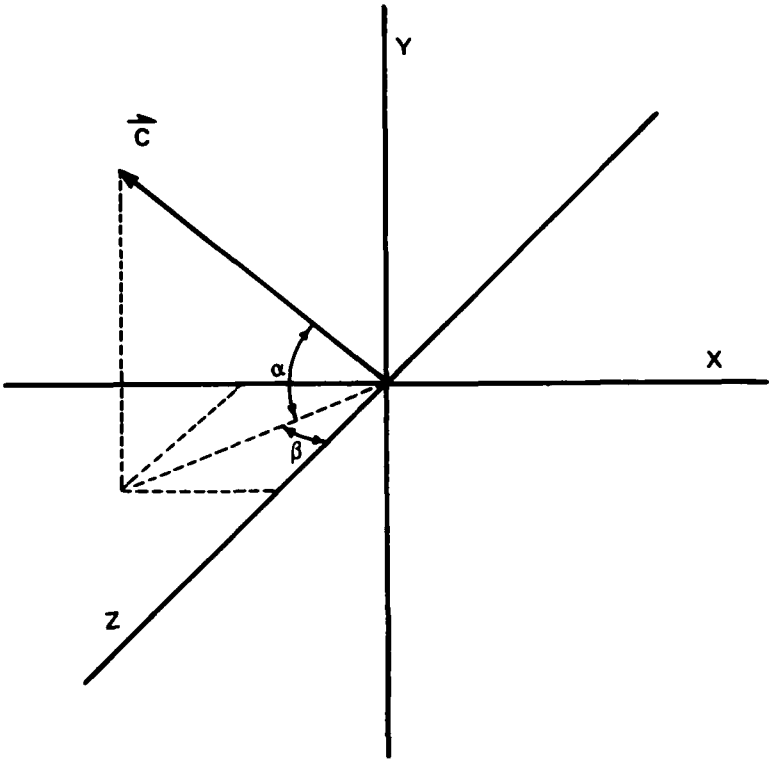


Figure 10. Rotation angles α and β

T_{t1} = translation from point (0,0,0) to CP

T_{ry} = rotation about the Y axis of β radians

T_{rx} = rotation about the X axis of α radians

T_{rot} = rotation about the Z axis of 5 degrees

[x' y' z' 1] = [x y z 1] T

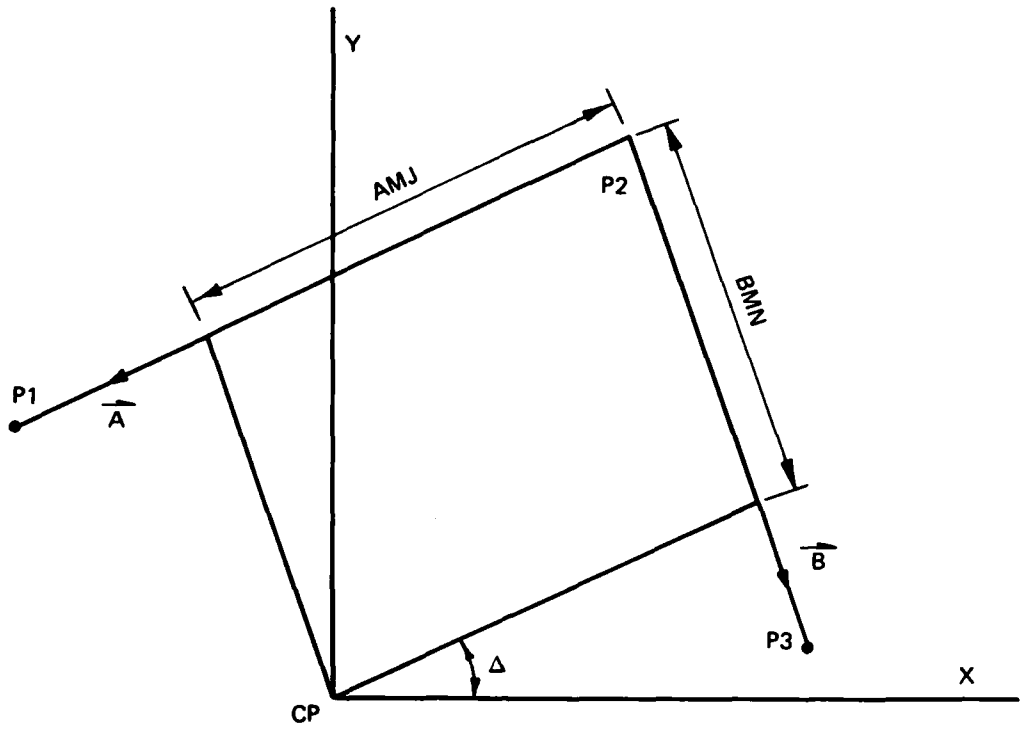
The total transformation is stored in the matrix table and is then a constant for that particular arc.

Circle

A full circle is drawn in the same manner as the circular arc, with one exception. The transformation is applied through 360 degrees. Also, if no radius is initially given, then half the length of the first line specified is used as the radius.

Elliptical Arc

Two distances, AMJ and BMN, and two lines with a common endpoint are specified (see Figure 11a). The common endpoint is P2. Unit vectors (A and



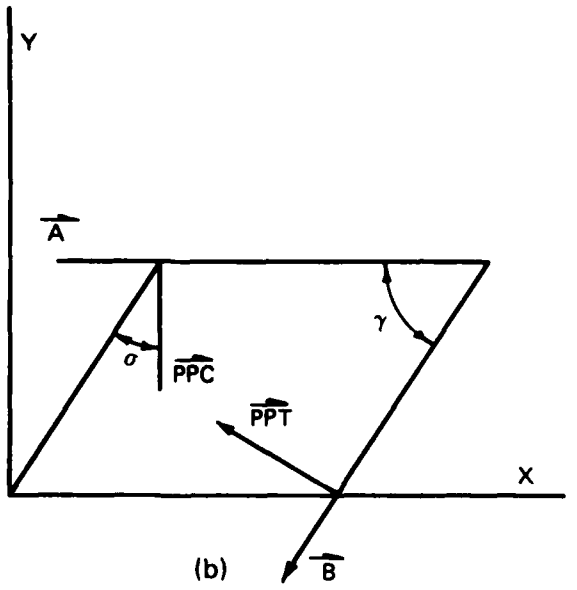


Figure 11. Elliptical arc definition in the XY plane

B) are calculated for each line in the direction away from P2. The unit vector perpendicular to the plane containing vectors A and B is C. From this vector, two angles (α and β) are found. These angles are the rotations about the X and Y axes, respectively, necessary to orient the curve into the XY plane (see Figure 10).

PPC and PPT are unit vectors perpendicular to A and B in the XY plane (see Figure 11b). A parallelogram with sides of length AMJ and BMN is then defined which locates the focus of the arc CP (see Figure 11a).

The parallelogram is transformed into a rectangle with the vector A parallel to the X axis and CP at the origin. This is accomplished with a translation, a rotation about the Z axis, and a shear in the XY plane. A circular arc is then drawn in the rectangle. When the inverse transformations are applied, an elliptical arc is generated. Again, the curve is generated by recursive application of the rotation transformation to the initial point, similar to that done with the circle. All the transformations can be concatenated into a single matrix T:

$$T = -T_{t1} * -T_{ry} * T_{rx} * T_{rz} * -T_{shx} * T_{rot} * T_{shx} * -T_{rz} * -T_{rx} * T_{ry} * T_{t1}$$

T_{t1} = translation from point (0,0,0) to CP

T_{ry} = rotation of β radians about the Y axis

T_{rx} = rotation of α radians about the X axis

T_{rz} = rotation of Δ radians about the Z axis

T_{shx} = shear of σ radians in the X direction

T_{rot} = rotation of 5 degrees about the Z axis

This total transformation is stored in the matrix table and becomes a constant for the ellipse.

Full Ellipse

A full ellipse is generated in the same manner as described for the elliptical arc with the recursive application of the transformation through 360 degrees. Also, if no distances are specified in the command, then the lengths of the specified lines are used for AMJ and BMN.

DATA STRUCTURE AND MANAGEMENT

Overview of Data Organization

The data management routines are designed to make efficient use of primary memory. In addition, the management of multiple objects is facilitated. A more detailed description of the fundamental routines may be found in Appendix C.

An object or scene consists of a combination of tables relating to geometry and topology. The data management system affords the programmer the ability to reference all the tables or constructs by name. The program uses this capability to link all the proper tables to a single object. SKETCH builds its own higher level data structure using the data management routines described in Appendix C.

Users of the SKETCH program are given the liberty of working with a single, unnamed object or multiple objects with name consisting of one to six characters. Internally, the data management system references all tables or constructs with an eight-character label. With the additional two-character suffix, SKETCH is able to distinguish the various objects as separate entities and also to manipulate the individual tables that comprise the structure of a single object.

Elements of Object Definition

An object named (xxxxxx) may consist of one or several of the following classes of tables:

OBJECT DEFINITION	xxxxxxØØ
COORDINATE TABLE	xxxxxxCT
LINE TABLE	xxxxxxLT
FACE TABLE	xxxxxxFT
CURVE TABLE	xxxxxxKT
MATRIX TABLE	xxxxxxMT
Extension Line Tables	
COORDINATE TABLE	xxxxxxEC
LINE TABLE	xxxxxxEL
Dimension Tables	
X	xxxxxxXD
Y	xxxxxxYD
Z	xxxxxxZD
TEMPORARY	xxxxxxPP
TEMPORARY	xxxxxxVV

Upon creating an object using the sketch command, an OBJECT DEFINITION construct is built. It provides a complete list of all the information tables that combine to "define" a given object. As tables are added or deleted, the entries in the object definition are modified to reflect the change. A typical OBJECT DEFINITION construct appears in Figure 12.

The first element is the display status flag indicating if the object is currently being displayed (0-no; 1-yes). The second and third elements contain the labels of the last coordinate and line, respectively.

The next portions of the object definitions are blocked off in groups of three words each. As new information is added to the object geometry, various

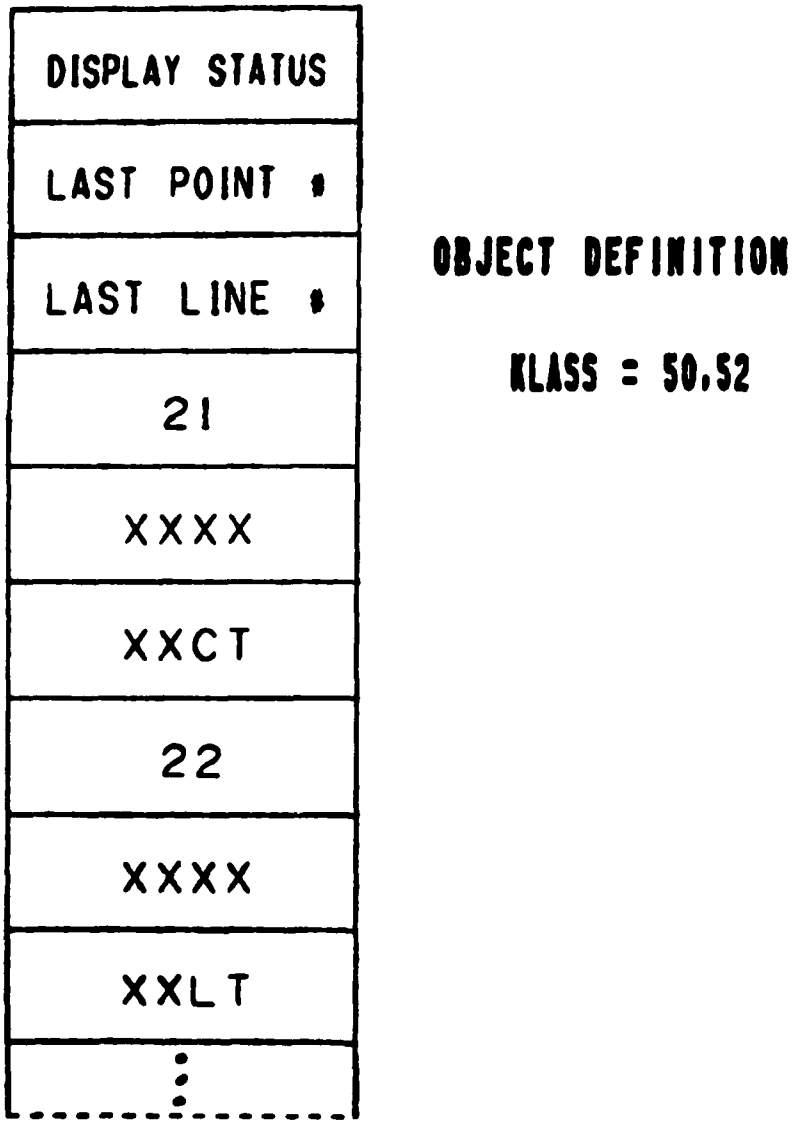


Figure 12. Organization of object definition

tables are added to the object definition. The first word in each three-word block contains the KLASS of the table of information that comprises a portion of the object description. The remaining two words in each block contain the name used by the data management routines to access the table. A given object definition may contain several of these construct blocks, and as tables are added or removed from an object, the corresponding entry in the object definition is entered or deleted.

The coordinate table consists of several blocks. Each block contains the information shown in Figure 13 when the object is still 2-D. The first word in the block is the label used to distinguish individual points. The next two words hold the actual location in screen coordinates where the point will be displayed. The remaining pair of words holds the initial screen location of the point when it was entered in sketching. The screen and sketch coordinate pairs differ only when the ZOOM option is active.

When conversion to 3-D is initiated, the xxxxxxCT construct is expanded to accommodate three additional records. The space is first used for the dimensioning plane information (see page 7) and then the actual 3-D coordinate value (see Figure 14).

The line table information is also organized into sectioned blocks, each

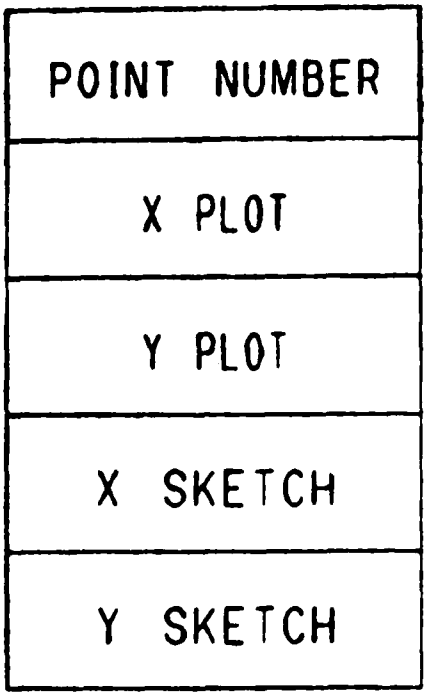
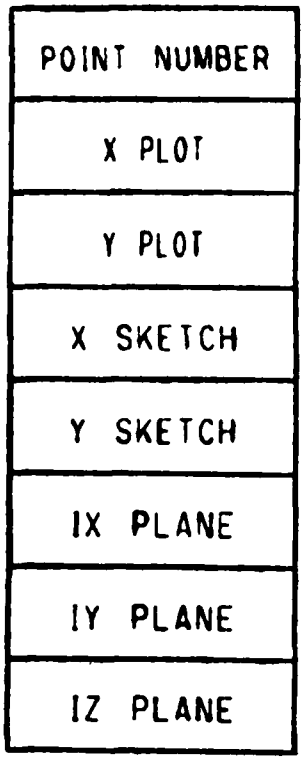


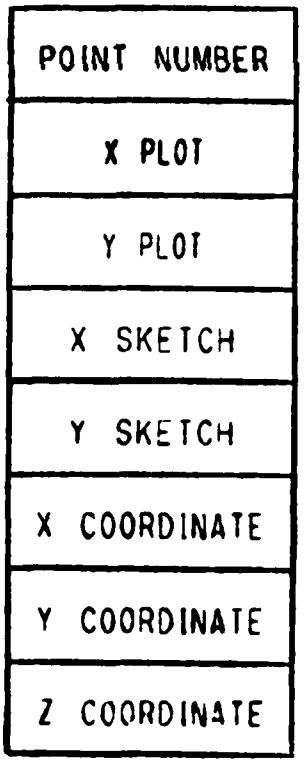
Figure 13. Coordinate table for a 2-D sketch



a)

COORDINATE TABLE
2-D SKETCH
WITH
DIMENSIONING PLANES

KLASS = 21



b)

COORDINATE TABLE
3-D SKETCH
KLASS = 21

Figure 14. Expanded coordinate table

one maintaining the information shown in Figure 15. The first word of the block is again an identification label. It distinguishes this line from all others. The second and third entries hold offset values for locating the proper endpoints in the coordinate table. The offsets are necessary as points may be added or deleted from an object at any given time. The line type, described on page 7, is contained in the fourth element of the block. The projected line length and sine and cosine of an angle, measured from the horizontal on the screen, fill the remaining three words that comprise a single block.

The face tables, referenced by xxxxxxFT, have a variable block size. The basic structure for one is depicted in Figure 16. The first word is a counter of the total number of face definitions contained in the table. Following the counter is a series of blocks that contain individual face descriptions. Each definition begins with an indicator of the number of words required for that particular face description. The next series of words holds offsets into the coordinate table for addressing the vertices of a face.

For processing curve information, a matrix table for assisting display and a curve table for holding the curve definition are maintained. The curve table is divided into seven word blocks (see Figure 17). The first word is an integer value identifying the curve type: 1 - circular arc; 2 - full circle; 3 - quadratic curve; 4 - cubic curve; 5 - elliptical arc; and 6 - full ellipse. The next three words in the block hold offsets into the coordinate table for locating the three points used to define the curve. Word five of the block holds the number of incremental steps used to approximate the curve. The sixth element may contain either a radius of curvature, another coordinate table offset, or nothing. The last word in the block is a pointer into the matrix table indicating which matrix is used to display this curve. The matrix table construct (see Figure 18) is sectioned into twelve word blocks. It stores by column the first twelve elements of the 4×4 matrix required to generate the curve display. The last column of the matrix is a constant [0 0 0 1].

When the dimensioning process is initiated, the coordinate table is expanded, as stated before, and several new constructs are created. These are: EXTENSION LINE COORDINATE TABLE; EXTENSION LINE TABLE; and X, Y, AND Z-DIMENSION TABLES. The format for the xxxxxxEC and xxxxxxEL constructs is the same as that for the coordinate and line tables described previously and will not be repeated here.

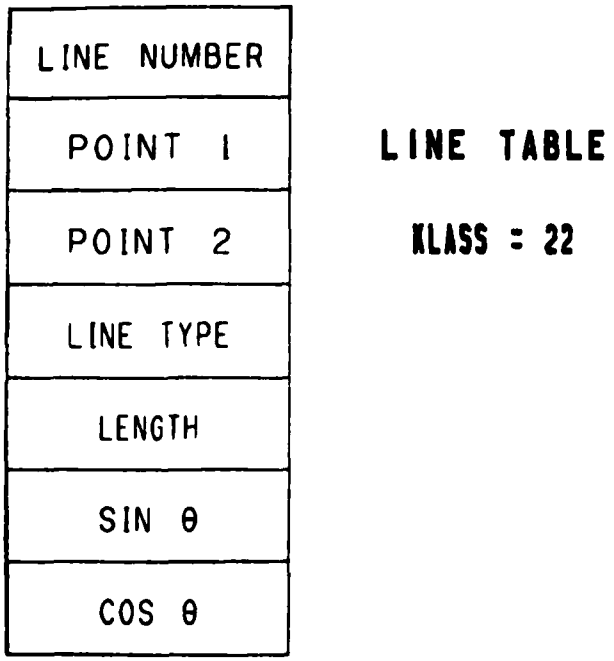


Figure 15. Organization of line table information

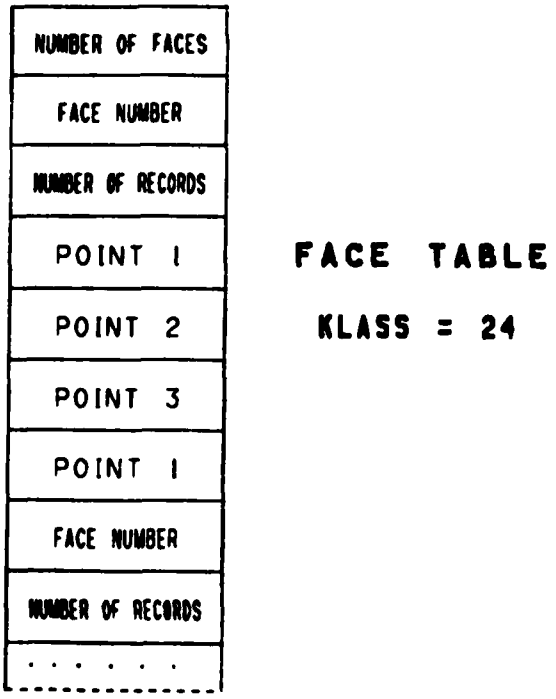


Figure 16. Format for face table information

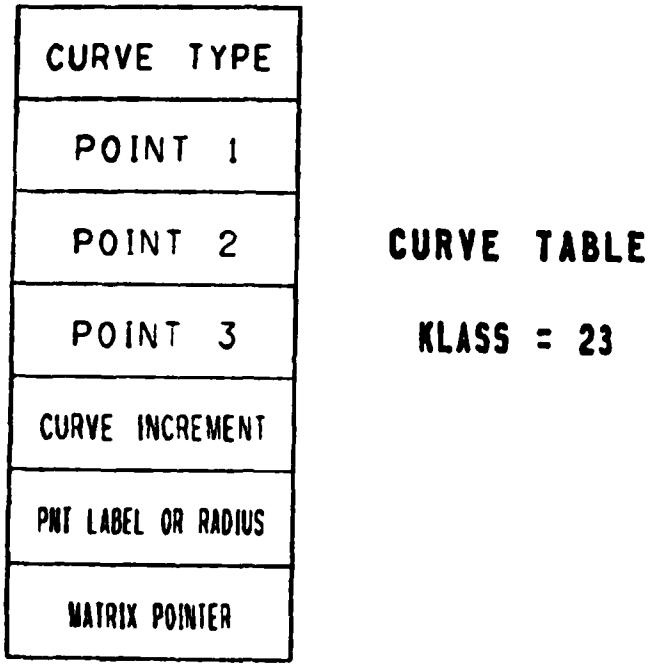


Figure 17. Curve table format

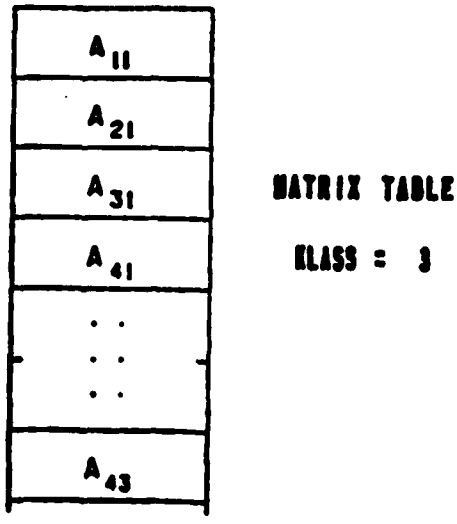


Figure 18. Matrix table layout

The dimension table constructs follow the form shown in Figure 19. The first word in the table is an indicator of the total number of dimension strings comprising the table. The record lengths are variable, and the first entry in each dimension string denotes the number of records that make up a given string. After that, the words alternately contain plane numbers and distance values (the dimensions specified), with both the initial and final words being plane numbers.

Two additional tables are created when dimensioning is complete. These

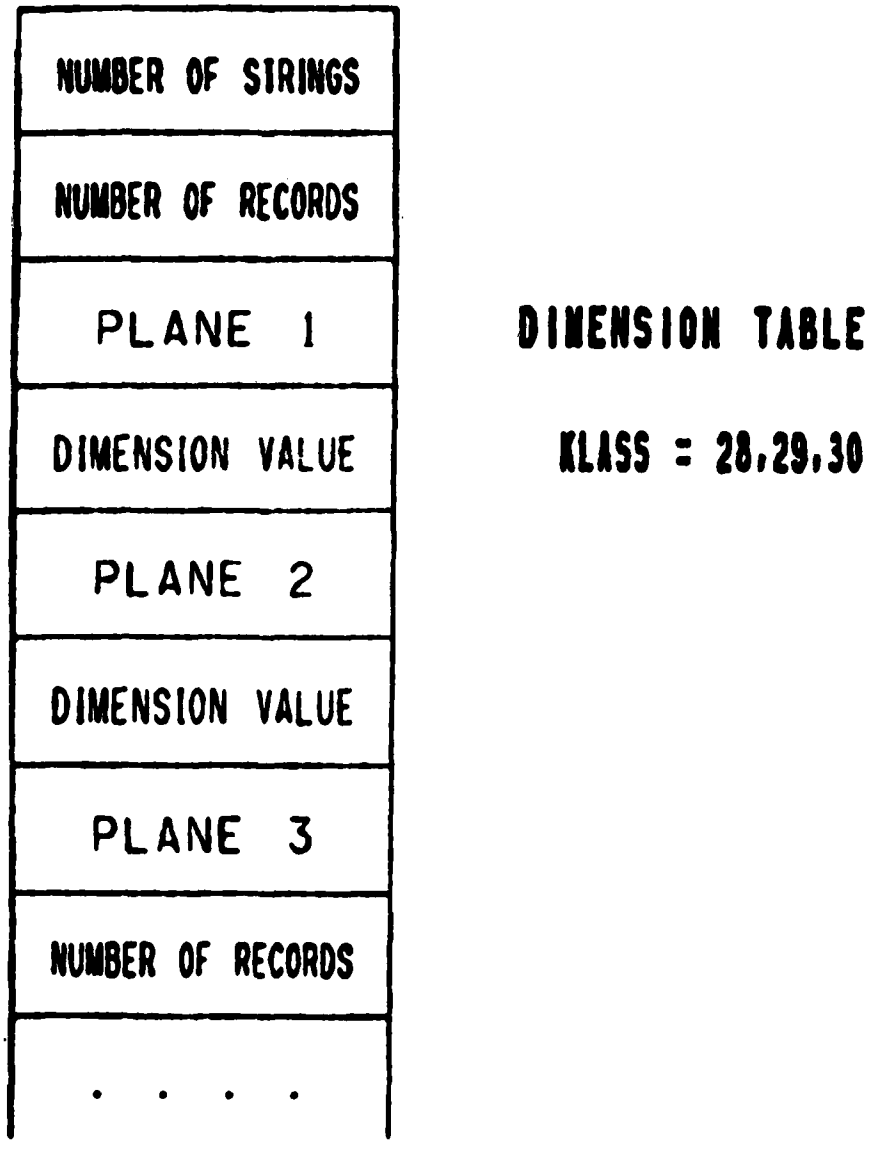


Figure 19. Organization of the dimension table elements

tables (xxxxxxVV and xxxxxxPP) are used for reorganizing the dimension information and plane definitions. This assists the redimensioning process if and when it is required.

Multiple Objects

Multiple-object capability is provided for in SKETCH by utilizing the power of the data management system. SKETCH allows the user to name each object with a six-character label. Internally, the data manager treats all names as eight characters in length (see the description of objects internal structure on page 34). The last two characters enable SKETCH routines to relate the various information tables to a single object. All tables that contain the same first six characters are part of the description of a single object.

There is only one active object in SKETCH at any given time. The "active" object is simply a utility provided for the user so that the object name does not have to be entered each time a command is given. When no object name is specified with a command, SKETCH uses the currently active object name which is contained in variable EDTNAM, the first element of the labeled common area EDSTAT. Whenever an object name is specified as part of a command, it is immediately loaded into EDTNAM and it becomes the active object.

The USING 'objnam' command is supplied as a convenience for the user so that at any time an object can be made the active object. This command is convenient to use when an object name has been misspelled and subsequent commands issued yield an ERROR 110. This error means that SKETCH could not locate any information under the misspelled name. The USING command enables the active object name to be corrected. It is also helpful when switching between various objects adding curve information because names are not accepted as part of the curve commands. The active object is used for all curve commands.

The handling of information by the SKETCH data manager also allows the user to operate on objects individually. Each one can be edited, erased, or modified without affecting the status of the others. The multiple-object capability allows a user to define a section of a complicated structure as a separate object. When all elements are completed, then the resulting assembly can be formed by merging components together.

Merge Concepts

For a sketching program to be truly effective as a design tool, the user must be provided the capability of combining several component geometries into a single, more complex entity. This enables the user to recall a common element to be utilized in several different designs without having to reconstruct it each time, and it allows the building of complex geometries in sections.

Within the SKETCH environment, the MERGE subroutine handles this processing. It compiles, under a single name, the geometric information contained in two or three separate objects. Access to MERGE is accomplished either directly through a user command (KILFLG = 0) or indirectly from the INNPUT subroutine (KILFLG = 1).

The MERGE logic consolidates all coordinate, line, curve, and matrix information for separate 3-D objects. MERGE does not accept 2-D sketched information. Also, face tables are not processed in merging. If that information is desired for the combined object, it must be regenerated using the GENERATE FACES command.

The command syntax utilized for a call to MERGE is as follows:

MERGE OBJECTA (OBJECTB) (INTO) OBJECTC

OBJECTB and INTO are optional components of the command; however, if specified, OBJECTB must exist and the word INTO must be correctly spelled. An OBJECTC, if it does not already exist, will be created when the MERGE command is given, but the same does not hold true for OBJECTA which must exist prior to the merge call.

In combining multiple objects under a single name, all redundant information is eliminated and, in the process, data management addresses are updated to reflect all changes. The procedure through which merging is accomplished is described in the following paragraphs.

The initial sections of code in MERGE handle variable initialization and processing of the user command. Checks are made for syntax errors, and internal flags are set to facilitate further processing.

Upon completing the command processing, MERGE goes through a sequence of locating the required objects and information. It also computes the memory requirements to complete the process. If any problems are determined, the merge attempt is aborted, and program execution is returned to the driving routines.

The object information associated with SKETCH consists of a series of tables containing geometry and topology. These tables constitute the data base required to generate a display on the user's terminal. Upon examination of the object structure, it is evident that a coordinate table establishes the basis for all other geometric information. That is, all line and curve information is generated using the coordinate table data and offsets into the coordinate table; therefore, because of its importance, the MERGE routine deals with the coordinate information first.

Initially, if OBJECTC exists, it is renamed to be handled internally as $#$#$. In the next step, copies of the coordinate information for $#$#$, OBJECTA, and OBJECTB are placed behind the newly created object definition for OBJECTC. After each table is copied, a search is initiated to locate repeated coordinates which are marked by replacing their coordinate or number label with the negative absolute value of the coordinate label of the point which it matched. When done, the new OBJECTC will have a continuous sequence for its coordinate labels.

Line table information is processed next and is handled very similar to the coordinates. In transferring the line information, however, each new element is checked with existing lines. When no match is found, the information for the single line is copied to the new OBJECTC. Curve information is processed in the same manner as the lines, with the additional requirement of transferring the matrices that correspond to the individual curves. In each case, the element labels are resequenced.

When all object geometry has been transferred to the new OBJECTC, a search is conducted through the coordinate table, and all elements with a nonpositive label are deleted from the object. In this manner, the newly completed OBJECTC occupies a minimal amount of the FREE array. The final step in the merging process is to eliminate all information referenced by $#$#$.

When MERGE is accessed through the INNPUT subroutine, slight changes occur in the procedure. The command interpreter section is bypassed, and the matrix information is not processed. Bypassing the interpreter results because there is no command to be decoded when MERGE is called from the INNPUT routine. Skipping the matrix processing section occurs because all matrix generation on input data takes place after INNPUT has finished collecting data. The last major difference is that all data referenced by the name $$$$$# are eliminated in addition to those referenced by $#$#$ ($$$$$# is a temporary

name assigned to objects from the INNPUT routine).

The internal arrangement of object information in the FREE array and a display of addresses internal to the MERGE routine are depicted in Figures 20 and 21 for convenience.

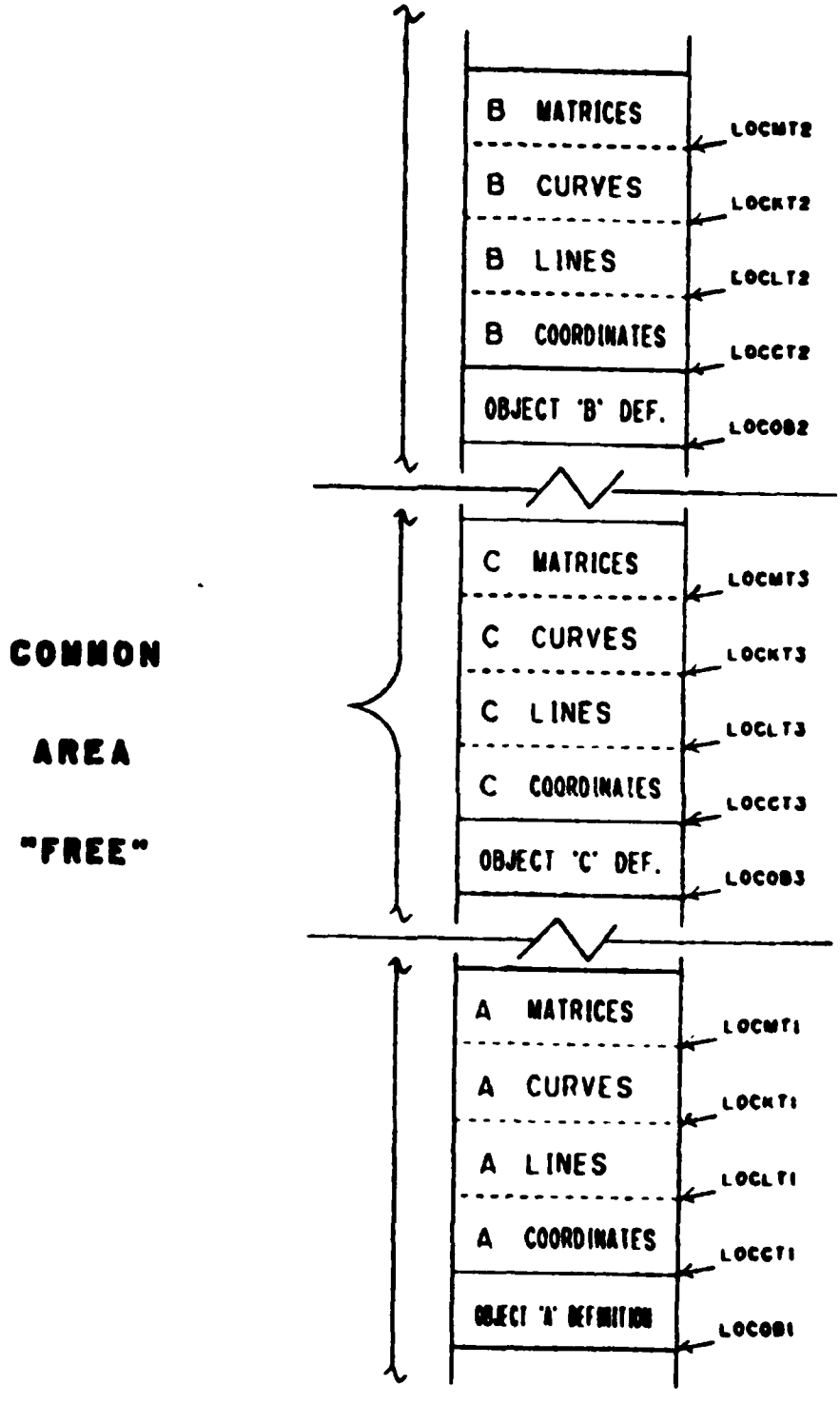


Figure 20. Merge objects; original object layout

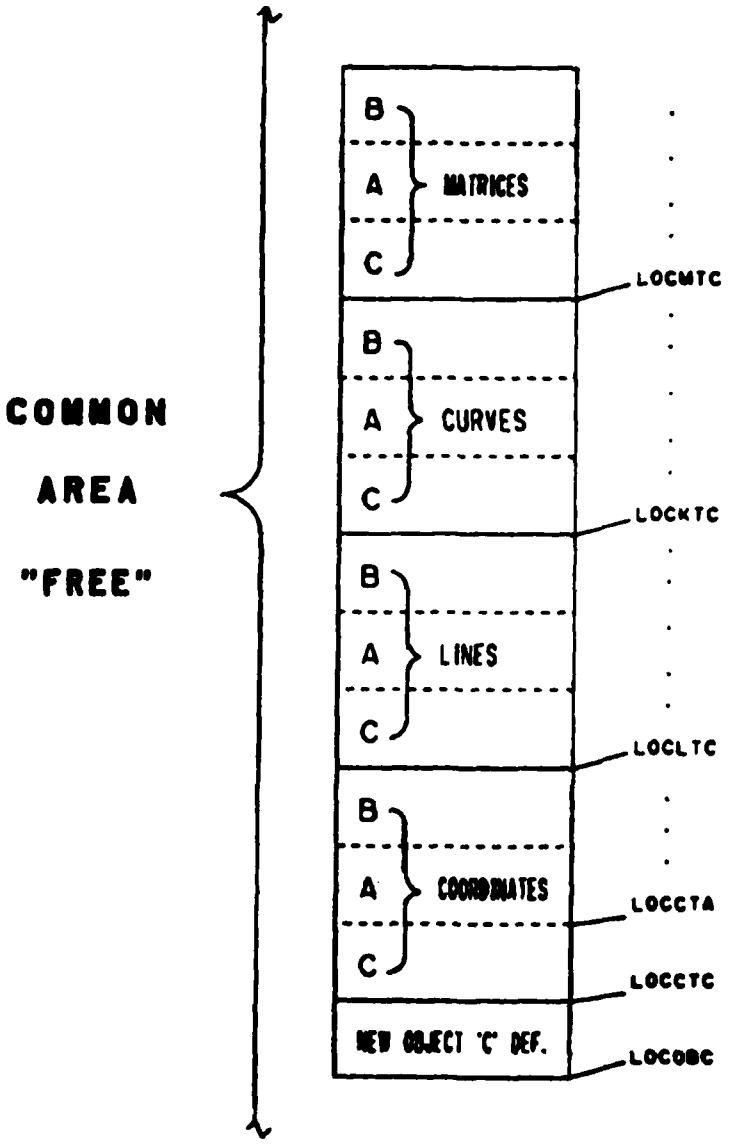


Figure 21. Final form of combined object information

USER INTERFACE

Command Interpretation

In order to provide a user-friendly interface for SKETCH, a powerful command interpreter is utilized. As commands are entered, the character and numerical strings are sorted by a preprocessor (see Appendix D). The results are passed through the labeled common area CMD to the routine DRVER2.

In DRVER2, the utility interpretation routine MASK is used extensively. It allows DRVER2 access to any number of characters in the words used in specifying a command. This capability allows the user the freedom of entering commands in an abbreviated fashion. Usually only the first two or three characters are required to recognize specific commands; therefore, misspellings of commands past the first two or three characters present no problems in SKETCH.

Within DRVER2, several flags may be set depending on the command specified. Most of these flags reside in the labeled common area SKLINK. The major flag of concern is the variable MODULE. It is used in the main driver to branch to the proper routines. The value of MODULE is set when the first word in a command sequence is recognized. Object names are loaded into the common area EDSTAT, and depending on the routines involved, several other flags may be set also.

Some degree of interpretation is handled by individual routines, but these occurrences are very specific and usually only one format is accepted.

Error Processing

In the course of running the SKETCH program, mistakes on entering data or commands are inevitable. Facilities are provided so that errors are flagged, and the user is notified as to the nature of the problem.

The utility routines provided with SKETCH handle nonfatal execution time errors in either of two ways. In very many cases, a direct explanation of the error is displayed in the scrolling area on the screen. In this manner, the user has an immediate description of the problem. Errors of this type usually result from some form of syntax mistake on entering a command string. The corresponding message may simply be a request for reentering the correct information, or it may indicate some other form of corrective action to be taken.

The second mode of notification is handled by the ERROR subroutine. It utilizes information loaded into the labeled common storage shown below:

```
COMMON / ERRORS / ROUTNE,IERMSG
```

This same common area is included in most subroutines. Whenever an error occurs, the individual routine names are loaded into the double-precision word ROUTNE, and an error code is stored in IERMSG.

The ERROR subroutine displays the name of the routine and the error code on the user's terminal, notifying him that an error has occurred. Execution is then returned to the command processor. To obtain a description of the problem, the user must refer to the listing of errors messages found in Appendix E.

BIBLIOGRAPHY

Brewer, J. A., III. 1983. "User's Guide: Computer Graphics Program for Generation of Engineering Geometry (SKETCH)," Instruction Report K-83-2, U. S. Army Engineer Waterways Experiment Station, Vicksburg, Miss.

Brewer, J. A. and Anderson, D. C. 1977. "Visual Interaction with Overhauser Curves and Surfaces," Computer Graphics, Vol 11, No. 2, pp 132-137.

Brewer, J. A. and Haag, A. S. 1980. "A New Approach to Observation Transformations for Three Dimensional Pictorials," Internal Document, Computer Graphics Research & Applications Laboratory, Mechanical Engineering Department, Louisiana State University, Baton Rouge, La.

Wilbanks, D. E. 1981. "The Development of Systematic Procedures for Interactive Design," unpublished Masters Thesis, Mechanical Engineering Department, Louisiana State University, Baton Rouge, La.

APPENDIX A: OVERLAY STRUCTURE

Table A1 is the command file to create the overlay structure for the SKETCH program on the Honeywell 6600 or DPS-1. A graphical representation of the overlay structure is given in Figure A1. Table A2 provides an index of the overlay element labels shown in Figure A1.

Table A1

Overlay Command File or Job Control File

```
10##N,J
20$:IDENT:404245,BREWER,11ROKAJAB
30$:LOWLOAD:36
40$:OPTION:NOGO
50$:USE:.GTLIT,.TSGF.,.FTSU.,.FXEMA.,FTLK
60$:SELECT:11ROKAJAB/CKDRVER1
70$:SELECT:GRAPHICS/OBJECT2D/TEKBLK
80$:LIBRARY:LO,L1,L2,L3
90$:LINK:LEVIB
100$:SELECT:11ROKAJAB/CKDMENU
110$:LINK:LEV1A,LEV1B
120$:SELECT:11ROKAJAB/CKDISOBJ
130$:LINK:LEV2A
140$:SELECT:11ROKAJAB/CKMODIFY
150$:LINK:LV21B
160$:SELECT:11ROKAJAB/CKDMENU
170$:LINK:LEV3B,LV21B
180$:SELECT:11ROKAJAB/CKINSRTL
190$:LINK:LEV3C,LEV3B
200$:SELECT:11ROKAJAB/CKLOCCL
210$:LINK:LEV3D,LEV3C
220$:SELECT:11ROKAJAB/CKLOCPT
230$:LINK:LEV3E,LEV3D
240$:SELECT:11ROKAJAB/CKUPDTL2
250$:LINK:LEV3F,LEV3E
260$:SELECT:11ROKAJAB/CKINSRTP
270$:LINK:LEV3G,LEV3F
280$:SELECT:11ROKAJAB/CKLOCLN
290$:LINK:LEV3H,LEV3G
300$:SELECT:11ROKAJAB/CRUPDTL1
310$:LINK:VEV2V,LEV2A
320$:SELECT:11ROKAJAB/CKDRVER2
330$:LINK:LEV3A
340$:SELECT:11ROKAJAB/CKLSUCMD
350$:LINK:LEV2C,LEV2B
360$:SELECT:11ROKAJAB/CKDFAXES
370$:LINK:LEV2D,LEV2C
380$:SELECT:11ROKAJAB/CKINIT
390$:LINK:LEV2E,LEV2D
400$:SELECT:11ROKAJAB/CKDETPNS
410$:LINK:LEV2F,LEV2E
420$:SELECT:11ROKAJAB/CKEXILIN
430$:LINK:LV23B
440$:SELECT:11ROKAJAB/CKINSRTL
450$:LINK:LV23F,LV23B
460$:SELECT:11ROKAJAB/CKINSRTP
```

(Continued)

Table A1 (Continued)

```
470$:LINK:LEV2G,LEV2F
480$:SELECT:11ROKAJAB/CKLIST
490$:LINK:LEV2H,LEV2G
500$:SELECT:11ROKAJAB/CKDIMENS
510$:LINK:LV23A
520$:SELECT:11ROKAJAB/CKLSUMCMD
530$:LINK:LV23C,LV23A
540$:SELECT:11ROKAJAB/CKLOCCL
550$:LINK:LV23D,LV23C
560$:SELECT:11ROKAJAB/CKLOCPT
570$:LINK:LEV3I,LV23D
580$:SELECT:11ROKAJAB/CKCKRTLF
590$:LINK:LEV3J,LEV3I
600$:SELECT:11ROKAJAB/CKCKBEFR
610$:LINK:LEV2I,LEV2H
620$:SELECT:11ROKAJAB/CKCVTDIM
630$:LINK:LEV2J,LEV2I
640$:SELECT:11ROKAJAB/CKSAVE
650$:SELECT:11ROKAJAB/CKASSIGN
660$:LINK:LV33A
670$:SELECT:11ROKAJAB/CKLSUCMD
680$:LINK:LEV2L,LEV2J
690$:SELECT:11ROKAJAB/CKFACGEN
700$:LINK:LEV2K,LEV2L
710$:SELECT:11ROKAJAB/CKINNPUT
720$:SELECT:11ROKAJAB/CKASSIGN
730$:SELECT:11ROKAJAB/CKIPUTPT
740$:SELECT:11ROKAJAB/CKIPUTLN
750$:LINK:LV43A
760$:SELECT:11ROKAJAB/CKLSUCMD
770$:LINK:LEV2M,LEV2K
780$:SELECT:11ROKAJAB/CKARC
790$:LINK:LV33C
800$:SELECT:11ROKAJAB/CKLOCCL
810$:LINK:LEV3K,LV33C
820$:SELECT:11ROKAJAB/CKIN3DPT
830$:LINK:LEV3L,LEV3K
840$:SELECT:11ROKAJAB/CKUTIL
850$:LINK:LEV3M,LEV3L
860$:SELECT:11ROKAJAB/CKDEL4CR
870$:SELECT:11ROKAJAB/CKUTIL
880$:LINK:LEV2N,LEV2M
890$:SELECT:11ROKAJAB/CKPARCUB
900$:LINK:LV33D
910$:SELECT:11ROKAJAB/CKLOCPT
920$:LINK:LV43C,LV33D
930$:SELECT:11ROKAJAB/CKLOCCL
940$:LINK:LV23L,LV43C
950$:SELECT:11ROKAJAB/CKUTIL
```

(Continued)

Table A1 (Concluded)

```
960$:LINK:LV23G,LV23L
990$:SELECT:11ROKAJAB/CKELLIP
1000$:LINK:LV53C
1010$:SELECT:11ROKAJAB/CKLOCCL
1020$:LINK:LV23K,LV53C
1030$:SELECT:11ROKAJAB/CKIN3DPT
1040$:LINK:LV33L,LV23K
1050$:SELECT:11ROKAJAB/CKUTIL
1060$:LINK:LV23M,LV33L
1070$:SELECT:11ROKAJAB/CKDEL4CR
1080$:SELECT:11ROKAJAB/CKUTIL
1090$:LINK:LEV3N,LV23M
1100$:SELECT:11ROKAJAB/CKDEL4EL
1110$:SELECT:11ROKAJAB/CKUTIL
1111$:LINK:LEV2P,LEV20
1112$:SELECT:11ROKAJAB/CKMERGE
1113$:LINK:LEV2Q,LEV2P
1114$:SELECT:11ROKAJAB/CKINARC
1115$:SELECT:11ROKAJAB/CKUTIL
1116$:LINK:LEV2R,LEV2Q
1117$:SELECT:11ROKAJAB/CKINQUC
1118$:SELECT:11ROKAJAB/CKUTIL
1119$:LINK:LEV2S,LEV2R
1120$:SELECT:11ROKAJAB/CKINELL
1121$:SELECT:11ROKAJAB/CKUTIL
1120$:EXECUTE
1130$:LIMITS:5,37K,-2K,2K
1140$:PRMFL:LO,R,R,GRAPHICS/OBJECT2D/GCSLIB
1150$:PRMFL:L1,R,R,GRAPHICS/OBJECT2D/TEKLIB
1160$:PRMFL:L2,R,R,GRAPHICS/OBJECT2D/HWLLIB
1170$:PRMFL:L3,R,R,WESLIB/APPLIB
1180$:PRMFL:H*,W,R,11ROKAJAB/SK
1190$:ENDJOB
```

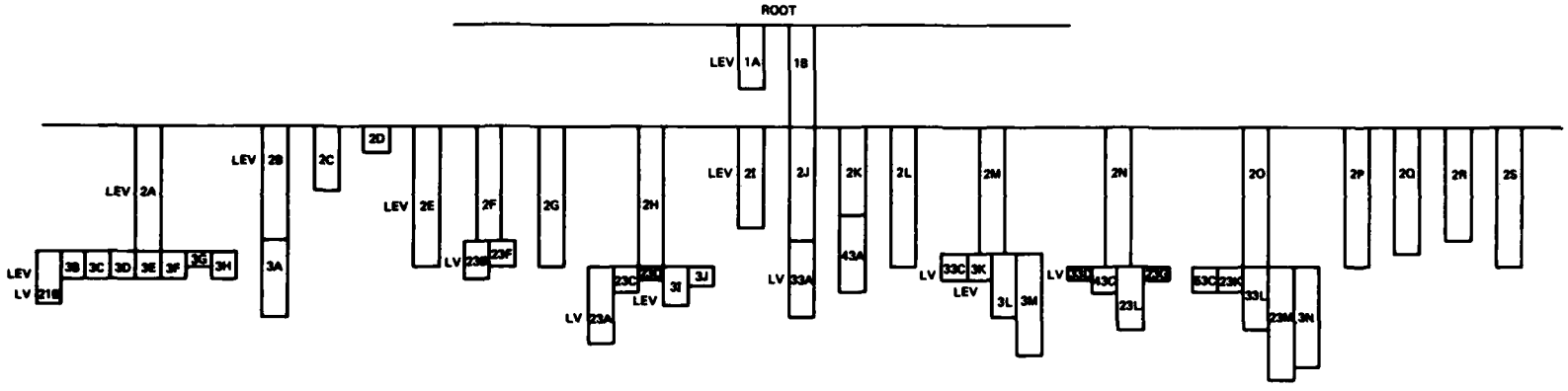


Figure A1. SKETCH overlay structure on the Honeywell 6600

Table A2

Overlay Label File Index

ROOT	CKDRVR1
LEV1A	CKDISOBJ
1B	CKDMENU
LEV2A	CKMODIFY
2B	CKDRVER2
2C	CKDFAXES
2D	CKINIT
2E	CKDETPNS
2F	CKEXTLIN
2G	CKLIST
2H	CKDIMENS
2I	CKCVTDIM
2J	CKSAVE,CKASSIGN
2K	CKINNPUT,CKASSIGN,CKIPUTPT,CKIPUTLN
2L	CKFACGEN
2M	CKARC
2N	CKPARCUB
2O	CKELLIP
2P	CKMERGE
2Q	CKINARC,CKUTIL
2R	CKINQUC,CKUTIL
2S	CKINELL,CKUTIL
LEV3A	CKLSUCMD
3B	CKINSRTL
3C	CKLOCCL
3D	CKLOCPT
3E	CKUPDTL2
3F	CKINSRTP
3G	CKLOCLN
3H	CKUPDTL1
3I	CKCKRTLF
3J	CKBEFR
3K	CKIN3DPT
3L	CKUTIL
3M	CKDEL4CR,CKUTIL
3N	CKDEL4EL,CKUTIL
LV21B	CKDMENU
LV23A	CKLSUCMD
23B	CKINSRTL
23C	CKLOCCL
23D	CKLOCPT
23F	CKINSRTP
23G	CKLOCLN
23K	CKIN3DPT
23L	CKUTIL
23M	CKDEL4CR,CKUTIL

(Continued)

Table A2 (Concluded)

LV33A	CKLSUCMD
33C	CKLOCCL
33D	CKLOCPT
33L	CKUTIL
LV43A	CKLSUCMD
43C	CKLOCCL
LV53C	CKLOCCL

APPENDIX B: SUBROUTINE DESCRIPTIONS

This appendix provides a brief summary of the purpose or function of the many subprograms that comprise SKETCH. It is not intended to be a complete documentation of the idiosyncrasies and concepts found in the subprograms.

The routine descriptions are in alphabetical order with the source file in which each can be located specified in parentheses. A condensed list of subprogram names and their corresponding source files is given in Table B1 at the end of this appendix.

ACOS (SKELLIP): ACOS is a function subprogram with one real argument, X. Input must be a number such that X < or = 1 and the inverse cosine value is returned in the word ACOS. A value of zero is returned in ACOS for illegal input values.

ARC (SKARC): This subroutine is used in defining circles and circular arcs. The required information is obtained interactively from the terminal keyboard. It handles the initial display of the curve and adds proper curve table and matrix table entries to retain its definition.

BCOS (SKINELL): This function subprogram performs the same function as ACOS; however, BCOS is used exclusively with the INELL subroutine.

CENUMB (SKDRVR1): CENUMB is a subroutine with two real and one integer arguments: X, Y, and IZ. The integer value contained in IZ is printed on the screen centered about the location X, Y.

CENSTR (SKDRVR1): This subroutine contains two real and three integer arguments: X, Y, NCHAR, ISTRNG, and IFLG. A string of characters stored in ISTRNG is displayed on the screen centered about the location X, Y. NCHAR is the number of characters stored in ISTRNG.

CMDINIT (SKINIT): Subroutine CMDINIT maintains no argument list. Its purpose is to initialize all of the information required by the command preprocessor. The labeled common areas CMD and QSTUFF are used to pass the results. For more details on the use of these common blocks, see Appendix D.

CMDSET (SKINIT): This subroutine has two integer arguments, I and J, which are used to establish the maximum numbers of words or numbers that will be acceptable to the command preprocessor LSUCMD.

CNTRPT (SKDRVR1): CNTRPT is a subroutine with six real arguments: X1, Y1, X2, Y2, XC, and YC. It computes the midpoint of a line segment with endpoints [X1,Y1] and [X2,Y2]. The centerpoint values are returned in XC and YC.

CRELIN (SKEXTLIN): CRELIN creates extension lines for the purpose of dimensioning a sketched object. CRELIN is called three successive times from EXTLIN. The first call creates XD extension lines parallel to the Z axis. The second call creates YD extension lines parallel to the X axis. The final call results in ZD extension lines parallel to the Y axis.

CROSS (SKUTIL): This subroutine has three real arrays as arguments. The cross product of the vectors A and B is computed and returned in the vector C.

CURFIN (SKDRVR1): A subroutine with no argument list, CURFIN is called when any curves were set up in subroutine INNPUT. It calls the correct subroutine to complete the corresponding curve and matrix table information.

DAXES (SKDMENU): DAXES has no argument list. The information required to position and draw the active coordinate system is contained in the AXES labeled common storage.

DBOX (SKDEMU): DBOX is a subroutine with no argument list. The information stored in the labeled common block BOUNDS is used to outline the workspace and menu areas.

DELETE (SKLIST): This subroutine does not hold any calling arguments. The EDSTAT labeled common storage holds the necessary information for DELETE to locate and remove all references to the specified object.

DELCOR (SKDEL4EL): This routine is used when the intersection of two lines is replaced by either a circular or elliptical arc. It replaces the common endpoint of each of the intersecting lines with the corresponding point of tangency between the curve and each line.

DEL4CR (SKDEL4EL): This routine deletes the two lines used in defining a full circle. If the two lines were sides of a square, then the remaining two sides of the square are also deleted.

DEL4EL (SKDEL4EL): This routine deletes the two lines used in defining a full ellipse. If the two lines were sides of a parallelogram, then the remaining two sides of the parallelogram are also deleted.

DETPNS (SKDETPNS): A major subroutine with no argument list, DETPNS is used to assign each vertex to a set of dimensioning planes. It also is set up to determine the maximum number of dimensioning planes required in each of the three principal directions which are then returned in the labeled common area DIMPLS.

DFAXES (SKDFAXES): DFAXES has no argument list but instead is used to interactively define a reference coordinate system. All of the inputs and required parameters are stored in the labeled common block AXES.

DIMENS (SKDIMENS): This is a major subroutine, and it has no calling arguments. The necessary information is contained in one of several common blocks: DIMPLS, SKLINK, EDSTAT, DISTAT, CMD, BOUNDS, and OFFSET. In DIMENS, the user interactively defines the dimensions needed to locate all the vertices in a 3-D coordinate system. For a more detailed description of the dimensioning process, refer to page 7 of this document or to the SKETCH User's Manual (Brewer 1983).

DISJNT (SKDRVR1): This subroutine is provided with four real arguments (X1, Y1, X2, Y2) that correspond to the endpoints of a line segment. The routine's function is to draw only the interior portion of the line segment, leaving the endpoints open so that the coordinate labels may be displayed clearly.

DISPLY (SKDRVR1): DISPLY is a major subroutine in the SKETCH package. It uses the labeled common areas EDSTAT and DISTAT to pass pertinent information. DISPLY is responsible for locating and ensuring that any visible objects are redrawn after the screen has been erased.

DISOBJ (SKDISOBJ): DISOBJ is a major subroutine and does not maintain an argument list. It processes the actual display of individual objects. All object information that is visible will be sent to the screen.

DIST (SKDRVR1): This is a function subprogram that holds two real arguments: DELX and DELY. It calculates the Euclidean norm or distance for the two deltas and returns the value in DIST.

DIST3D (SKUTIL): This function subprogram has one real and two integer calling arguments: IOFF1, IOFF2, and CTNAME. The integer arguments are used as offsets into the coordinate table whose name is in CTNAME. The 3-D coordinate values of the two points, located using IOFF1 and IOFF2, are used to compute the Euclidean norm or distance between the points: SQRT ((X2-X1)**2 + (Y2-Y1)**2 + (X2-Z1)**2)

DIMENU (SKDMENU): This subroutine has a single integer calling argument used to indicate which menu or portion thereof is to be displayed. It utilizes the information content of the labeled common area BOUNDS.

DOT (SKELLIP): DOT is a function subprogram with two real arrays, X and Y, as the calling arguments. The inner or "dot" product is computed and the value returned in DOT.

DOTPRO (SKEXTLIN): This function subprogram has four real arguments: X1, Y1, X2, Y2. It performs the same function as DOT but is used exclusively with the EXTLIN subroutine.

DRVER2 (SKDRVER2): Subroutine DRVER2 has no calling arguments but does make extensive use of the labeled common areas SKLINK, EDSTAT, CMD, and DISTAT. This routine functions as the command interpreter for SKETCH. The information from CMD is translated into a sequence of execution flags, and the results are transferred to the proper common areas for future processing.

ELLIP (SKELLIP): This is the subroutine used in defining elliptical arcs and full ellipses. The required information, three points and two radii, is interactively entered from the keyboard. ELLIP handles the initial display of the ellipse or elliptical arc and adds the proper curve and matrix definition into the object tables.

ERROR (SKLIST): This subroutine has no calling arguments but uses the information stored in the labeled common area ERRORS. It writes a message into the scrolling area on the screen. The message contains a number code referencing the problem and the routine name in which the error occurred.

EXTLIN (SKEXTLIN): EXTLIN generates two new tables in the dynamically managed data area. A coordinate table and a line table are temporarily required to store extension line information used in the dimensioning process.

FACGEN (SKFACGEN): FACGEN automatically generates polygonal face information for a sketched polyhedron. Faces are interactively displayed as they are created, and the user is given the opportunity to accept or reject each face. The order of vertices defining the face can also be reversed if desired.

IDLSH (SKLSUCMD): IDLSH is a function subprogram with a single integer argument, I. It is part of the command preprocessor package and is used to locate delimiters inside a command string.

IDLTES (SKLSUCMD): IDLTES is a function subprogram with a single integer argument, I. It is part of the command preprocessor package and is used to test for delimiters.

ILSQT (SKLSUCMD): ILSQT is a function subprogram with a single integer argument, ICNT. It is part of the command preprocessor package and is used to locate the second of a pair of quotation marks.

INARC (SKINARC): INARC is a subroutine with no calling arguments. It is used to complete the curve and matrix definitions for circular curve information that is being read from secondary storage.

INELL (SKINELL): INELL is a subroutine with no calling arguments. It is used to complete the curve and matrix definitions for elliptical curve information that is being read from secondary storage.

INITDM (SKINIT): INITDM is a subroutine with a single integer argument, LENGTH. Its function is to initialize all of the parameters necessary to use the data manager package of subroutines. The argument LENGTH is the size of the array space that the routines will dynamically manage. For a more detailed description of the data management package, see Appendix C.

INNPUT (SKINNPUT): Subroutine INNPUT has no calling arguments. It is utilized when reading object descriptions from secondary storage. This routine requires some level of user interaction, the amount of which depends on the number of objects being input and the complexity of the information read.

INQUC (SKINQUC): INQUC is a subroutine with no calling arguments. It is used to complete the curve and matrix definitions for quadratic curve information that is being read from secondary storage.

INSRTL (SKINSRTL): This routine inserts 2-D lines into appropriate line tables.

INSRTP (SKINSRTP): This routine inserts 2-D points into appropriate coordinate tables.

IN3DPT (SKIN3DPT): The routine inserts 3-D points into 3-D coordinate tables.

IOCHSZ (SKDRVR1): IOCHSZ is an input/output (I/O) routine that has a single integer calling argument, I. The value of I invokes a series of control characters to be sent out, resulting in a change in the size of the hardware-generated characters being displayed on the screen.

IOGET (SKDRVR1): IOGET is an I/O routine that has three integer-valued calling arguments: IST, NCASK, and NCREC. IST and NCASK are the number of characters actually read and the number of characters requested, while NCREC is the number of records actually input.

IOINIT (SKDRVR1): IOINIT is an I/O routine that has no calling arguments. It sets the limits on the scrolling area of SKETCH and initializes the required counters and parameters.

IONEXT (SKDRVR1): IONEXT is an I/O routine that has a single integer calling argument, I. This routine manages the scrolling area of the SKETCH display. The value of I is the number of lines requested to determine whether the command will fit in the scrolling area.

IOSET (SKDRVR1): IOSET is an I/O routine that has six integer-valued calling arguments: IRS, IRST, ICS, ICW, IIN, and IOT. Its purpose is to allow the scrolling area limits to be manipulated.

IOTOP (SKDRVR1): IOTOP is an I/O routine that has no arguments. It erases the screen and redisplays the active information whenever the routine is called. This occurs when the scrolling area becomes full, when a special set of commands requiring a change in the menu is issued, or when the erase command is given.

IOWRT (SKDRVR1): IOWRT is an I/O routine that contains two integer arguments, ISTR and NCHR. It manages the printing of messages, ensuring that the message is contained within the margins of the scrolling area. ISTR contains the message to be printed, and NCHR contains the number of characters in the message.

IPUTLN (SKIPUTLN): This routine is called by INNPUT to place line information into appropriate line tables.

IPUTPT (SKIPUTPT): This routine is called by INNPUT to place 2-D and 3-D coordinate information into appropriate coordinate tables.

IRENAM (SKINNPUT): IRENAM is a subroutine used when reading objects from secondary storage. It helps maintain the naming convention requirements by renaming an object from " " to "NONAME" (see SKETCH User's Manual). The single integer argument is used to pass an address to the INNPUT routine.

KVAL (SKDETPNS): KVAL is a function subprogram which returns an integer value from one to three to subroutine DETPNS. The integer value indicates that DETPNS is working with X, Y, or Z dimensioning planes, respectively.

LASCHR (SKLSUCMD): LASCHR is a function subprogram with a single integer argument, NEXT. It is part of the command preprocessor package and is used to locate the position of the last character in a command sequence.

LIST (SKLIST): The LIST subroutine has no calling arguments but utilizes the labeled common areas CMD and EDSTAT. LIST interprets the command string to determine what object information is to be listed (coordinates, lines, faces, etc.) and writes the information on the screen.

LISTOB (SKLIST): LISTOB has a single integer argument, IFACE, which is a flag noting whether or not an object's face information is to be printed. LISTOB is to LIST as DISOBJ is to DISPLY. It handles the actual listing of the object information.

LOCCL (SKLOCCL): LOCCL locates the closest line in the appropriate line table to the user-supplied position of input cross hairs.

LOCLN (SKLOCLN): LOCLN locates the line in the appropriate line table corresponding to two coordinate indices.

LOCPT (SKLOCPT): LOCPT locates the closest point within a tolerance in the appropriate coordinate table to the user-supplied position of input cross hairs.

LSUCMD (SKLSUCMD): The LSUCMD subroutine has no calling arguments but makes extensive use of the labeled common areas CMD and QSTUFF. It functions as the command preprocessor for the SKETCH routines. For more details on the command preprocessor package, see Appendix D.

MASK (SKDRVR1): The subroutine MASK is a part of the command preprocessor package. It has four integer arguments (NWORD, NCHR, TO, and MAXCHR) and is used to selectively extract words or portions of words from a command string.

MERGE (SKMERGE): MERGE is a utility subroutine with two integer arguments, KILFLG and ENPTNM. It enables the user to combine the coordinate, line, and curve information contained in two or three different objects. The arguments are used when MERGE is called from the INNPUT routine or in the event that the user wishes to delete the objects that combine to produce the single definition. The merge is accomplished by first making a copy of all of the coordinate tables since the remaining information is based on this construct. Next, the line and curve table elements are processed individually. All redundant information is eliminated.

MODIFY (SKMODIFY): The MODIFY subroutine has no calling arguments. It is a major subroutine that processes the graphical input of sketched information.

MOVE (SKUTIL): MOVE is a subroutine with two real arrays, A and B, as calling arguments. The routine moves the 3 × 3 array A into the 4 × 4 array B with the first three elements in the fourth row and column being zero. B(4,4) = 1.0.

MOVEDN (SKDRVR1): Subroutine MOVEDN is part of the data management package. It has three arguments: A, B, and L. MOVEDN is used to transfer blocks of the free array to a different location. For more details, refer to Appendix C.

MMULT (SKUTIL): MMULT is a utility routine for multiplying two N × N matrices in the argument list, A and B. The results are returned in the N × N matrix C. The fourth argument is N, the dimension of the matrices.

MXMUP (SKDRVR1): MXMUP is a utility routine for multiplying a 1 × 3 vector, A, with a 4 × 4 matrix, B. The resulting 1 × 3 vector is stored in C.

MXMUP4 (SKDRVR1): MXMUP4 is a utility routine for multiplying a 2 × 4 vector, A, with a 4 × 4 matrix, B. The required information is stored in a 1 × 3 vector, C.

NADD (SKDRVR1): The NADD function subprogram is part of the data management package. It is used to dynamically allocate a portion of the data array to a specific object or construct. For more details, refer to Appendix C.

NDELETE (SKDRVR1): Function subprogram NDELETE is part of the data management package. It is used to dynamically eliminate information from the data array when that information is no longer needed. For more details, refer to Appendix C.

NEXEXL (SKEXTLIN): NEXEXL determines the next extension line to be considered by EXTLIN.

NEXCHR (SKLSUCMD): NEXCHR is a function subprogram with one integer argument, I. It is part of the command preprocessor package and is used to locate nonblank characters in a command string.

NFIND (SKDRVR1): Function subprogram NFIND is part of the data management package. It is used to locate information and addresses stored in the data array. For more details, refer to Appendix C.

NFINDO (SKDRVR1): Function subprogram NFINDO is similar to NFIND but restricts its search to a specific OBJECT DEFINITION. For more details, refer to Appendix C.

NLIST (SKDRVR1): Function NLIST is part of the data management package.

It is used to find and return the names and other pertinent information concerning the constructs. For more details, refer to Appendix C.

NLOAD (SKDRVR1): NLOAD is a function subprogram with three arguments: AI, AJ, and AK. It is used to move the first six characters in AJ plus two characters stored in AK and place them in AI.

NMOVCR (SKDRVR1): NMOVCR is part of the data management package. It is used to move a range of columns or rows when reformatting or deleting parts of arrays. For more details, refer to Appendix C.

NORM (SKUTIL): This subroutine has four real arrays as arguments. The plane containing the points PT1, PT2, and PT3 is calculated. The vector normal to the plane is computed and returned in the fourth array in the argument list.

NORMAL (SKFACGEN): NORMAL calculates the normal to a plane formed by three points in three space.

NREMAP (SKDRVR1): Subroutine NREMAP is part of the data management package. It is used to reformat an array A into array B (A(M,L) = B(N,L)) where the value of N is greater than that of M. For more details, refer to Appendix C.

NRENAM (SKDRVR1): Subroutine NRENAM is part of the data management package. It is used to change a construct name from 'OLDNAM' to 'NEWNAM' in the data management header information. For more details, refer to Appendix C.

NSERCH (SKDRVR1): Subroutine NSERCH is part of the data management package. It sets the range of values searched by the NLIST function. The routine resets the use flags for those constructs within the search range. For more details, see Appendix C.

NUMTES (SKLSUCMD): NUMTES is a function subprogram with a single integer argument. It is part of the command preprocessor package and is used to locate numbers inside a command string.

PARCUB (SKPARCUB): This subroutine is used in defining parametric quadratic curves and modified Overhauser cubic curves. The required information is obtained interactively from the keyboard for both types of curves. It handles the initial display of each curve and adds proper curve and matrix table entries to retain the curve definitions.

PLNTES (SKDETPNS): PLNTES determines all of the points in the current dimensioning plane.

RENAME (SKLIST): This subroutine is used to replace all references of 'OLDNAM' with 'NEWNAM' in the SKETCH object definitions.

RESET (SKDETPNS): RESET interchanges high reference plane values with low values in the process of determining dimensioning planes.

REVERS (SKFACGEN): REVERS reverses the current ordering of points in the active polygon or face.

SAVE (SKSAVE): This subroutine is utilized to save to secondary storage all information necessary to define an object or objects that have been previously defined.

SAVOB (SSAVE): This subroutine is called by subroutine SAVE to record on secondary storage all necessary information to completely define an object.

SCLMUP (SKUTIL): This subroutine has a scalar (A) and two vectors (B,C) as arguments. Vector B is scaled by A, and the results are returned in C.

SDIST (SKEXTLIN): SDIST calculates the 2-D distance between two points.

SETPLS (SKDETPNS): SETPLNS determines the number of dimensioning planes in a given direction and establishes the points contained in each plane.

SET4RD (SKINNPUT): This subroutine has a construct name and type as arguments. It checks to see if the specified construct exists in the header information; if not it creates an entry.

SHOW (SKFACGEN): This subroutine is used with the face generation routines. It traces on the screen the line segments that define a face.

SKDUMP (SKLIST): SKDUMP allows internal data in labeled common blocks to be inspected for debugging purposes.

SKINIT (SKINIT): This subroutine initializes all system variables and constants when the program execution begins.

STRP22 (SKMERGE): This subroutine has two integer arguments, FROM and TO. It takes the last two characters of 'FROM' and concatenates them onto the end of 'TO'.

STRP23 (SKINNPUT): Same as STRP22.

UNIT (SKUTIL): This subroutine has a single argument, a real vector A. The vector A is normalized and returned.

UNMARK (SKFACGEN): This routine sets the line numbers in the appropriate line table to positive values.

UNPACK (SKDRVR1): This subroutine has one integer (NC) and two integer array (IIN and IOUT) arguments. NC elements of the 1-byte array INN are copied into the 2-byte array IOUT.

UPDTL1 (SKUPDTL1): UPDTL1 does garbage collection for the appropriate line table during the sketching process.

UPDTL2 (SKUPDTL2): UPDTL2 does works complementary to that done by UPDTL1 with line tables.

VCTADD (SKUTIL): This subroutine has three real vectors as arguments: A, B, and C. A and B are added, and the result is returned in C.

VCTSUB (SKUTIL): This subroutine has three real vectors as arguments: A, B, and C. B is subtracted from A, and the result is returned in C.

Table B1

Subprogram/File Index

ACOS	SKELLIP1
ADEIN	SKDRVR1
ADEOUT	SKDRVR1
ARC	SKARC
BCOS	SKINELL
CENUMB	SKDRVR1
CENSTR	SKDRVR1
CKRTLF	SKCKRTLF
CMDINIT	SKMODIFY
CMDSET	SKMODIFY
CNTRPT	SKDRVR1
CRELIN	SKEXTLIN
CROSS	SKUTIL
CURFIN	SKDRVR1
DAXES	SKDMENU
DROX	SKDMENU
DELETE	SKLIST
DELCOR	SKDEL4CR
DEL4CR	SKDEL4CR
DEL4EL	SKDEL4EL
DETPNS	SKDEPTNS
DFAXES	SKDFAXES
DIMENS	SKDIMENS
DISJNT	SKDRVR1
DISPLY	SKDRVR1
DISOBJ	SKDISOBJ
DIST	SKDRVR1
DIST3D	SKUTIL
DMENU	SKMENU
DOT	SKELLIP
DOTPRO	SKEXTLIN
DRVER2	SKDRVER2
ELLIP	SKELLIP
ERROR	SKLIST
EXTLIN	SKEXTLIN
FACGEN	SKFACGEN
GCSWHR	SKDRVR1
IDLSH	SKLSUCMD
IDLTES	SKLSUCMD
ILSQT	SKLSUCMD
INARC	SKINARC
INELL	SKINELL
INITDM	SKINIT
INNPUT	SKINNPUT
INQUC	SKINQUC
INSRTL	SKINSRTL
INSRTP	SKINSRTP
IN3DPT	SKIN3DPT
IOCHSZ	SKDRVR1

(Continued)

Table B1 (Continued)

IOGET	SKDRVR1
IOINIT	SKINIT
IONEXT	SKDRVR1
IOSET	SKDRVR1
IOTOP	SKDRVR1
IOWRT	SKDRVR1
IPUTLN	SKIPUTLN
IPUTPT	SKIPUTPT
IRENAM	SKINNPUT
KVAL	SKDETPNS
LASCHR	SKLSUCMD
LIST	SKLIST
LISTOBJ	SKLIST
LOCCL	SKLOCCL
LOCLN	SKLOCLN
LOCPT	SKLOCPT
LSUCMD	SKLSUCMD
MASK	SKDRVR1
MERGE	SKMERGE
MODIFY	SKMODIFY
MOVE	SKMODIFY
MOVE	SKUTIL
MOVEDN	SKDRVR1
MMULT	SKUTIL
MXMUP	SKDRVR1
MXMUP4	SKDR
NADD	SKDRVR1
NDELETE	SKDRVR1
NEXEL	SKEXTLIN
NEXCHR	SKLSUCMD
NFIND	SKDRVR1
NFINDO	SKDRVR1
NLOAD	SKDRVR1
NLIST	SKDRVR1
NMOVCR	SKDRVR1
NORM	SKUTIL
NORMAL	SKFACGEN
NREMAP	SKDRVR
NRENAM	SKDRVR1
NSERCH	SKDRVR1
NUMTES	SKLSUCMD
PARCUB	SKPARCUB
PLNTES	SKDETPNS
RENAME	SKLIST
RESET	SKDETPNS
REVERS	SKFACGEN
SAVE	SKSAVE
SAVEOB	SKSAVE
SCLMUP	SKUTIL

(Continued)

Table B1 (Concluded)

SDIST	SKEXTLIN
SETPLS	SKDETPNS
SET4RD	SKINNPUT
SHOW	SKFACGEN
SHOWTB	SKDEL4CR
SKDUMP	SKLIST
SKINIT	SKINIT
STRP22	SKMERGE
STRP23	SKINNPUT
UNIT	SKUTIL
UNMARK	SKFACGEN
UNPACK	SKDRVR1
UPDTL1	SKUPDTL1
UPDTL2	SKUPDTL2
VCTADD	SKUTIL
VCTSUB	SKUTIL

APPENDIX C: IN-CORE DATA MANAGEMENT SYSTEM

General Information

The data management system consists of a collection of subroutines used for the generation and handling of geometry related constructs in a dynamic environment. The construct is a collection of data which can be dynamically created, expanded, or deleted.

The application programmer provides space for the data in a common block named 'FREE' and initializes the system by specifying the length of this block (see subroutine INITDM). The programmer now has the capabilities to create new constructs or add space to existing ones (function NADD); find an existing construct and retrieve information about it (function NFIND); delete all or part of an existing construct (function NDELET); and receive a series of construct names and information about specific classes of constructs (function NLIST and subroutine NSERCH).

The concept of an address associated with each construct will be used throughout the description of the system. The address referred to is actually the dimension of the element in the common block 'FREE' where the data associated with the construct are located. Assume the following statement has been executed:

```
COMMON/FREE/ CORE(2000)
.
.
LOC = NFIND (ANAME , KLASS , ND1 , ND2)
.
.
```

The returned value of LOC is equal to 100. The data of the construct ANAME would start at CORE(100) and follow contiguously. See "Internal Construct Structure" (page C7) for more information on internal data format and storage.

Care must be taken when using the value returned as the location. Values of 0 or -1 are returned as error conditions from some routines and would be invalid as an index.

Constructs are defined by four variables: name (ANAME), class (KLASS), and two dimensions (ND1 and ND2). If ANAME is a variable, it must be at least 8 bytes (characters) in length. This can be accomplished in various ways: BYTE ANAME(8), LOGICAL*1 ANAME(8), INTEGER*2 ANAME(4), REAL*4 ANAME(2),

REAL*8 ANAME. KLASS, ND1, and ND2 are standard integer variables. The variable ANAME contains the programmer provided construct name of 1-8 ASCII characters. Care should be taken to ensure that all references to ANAME are consistent in justification and padding. Left-justified and blank padding are preferred. Below are examples of correct and incorrect methods of ANAME usage:

Correct:

```
1)    REAL * 8 ANAME/'OBJ01   '/
      .
      .
      .
      LOC = NFIND (ANAME , ......

2)    LOC = NFIND ('OBJ01   ' , ......

3)    REAL * 8 ANAME
      READ(5,1) ANAME
 1    FORMAT( A8 )
      LOC = NFIND (ANAME , ......
```

(NOTE: FORTRAN READ stores ASCII left-justified and blank-filled.)

Incorrect:

```
1)    LOC = NFIND ('OBJ01' , ......
```

Names are stored within the system headers exactly as provided and returned exactly as they are stored. If the same name is used later and is not consistent with the original, the original will not be located.

The variable KLASS allows the programmer to classify each construct or group of constructs according to their type or function. KLASS must be an integer greater than zero and need not be unique for each construct. For example, a programmer may define all vectors with values between 50 and 59. KLASS 51 may be reserved for vectors that are node points, KLASS 52 for coordinate vectors, etc.

In this way, the programmer subdivides his construct definitions numerically and later finds constructs by KLASS range (see function NLIST).

The dimension variables ND1 and ND2 are provided by the programmer to

define the amount of space and the format desired for a construct. These are analogous to FORTRAN pseudodimensioning; i.e.,

```
ARRAY(ND1 , ND2) for an array
CVECT(ND1 ,  1 ) for a column vector
RVECT( 1  , ND2) for a row vector
SCALR( 1  ,  1 ) for a scalar value
```

For more information about these varibles and restrictions for use, see the individual subroutine explanations.

The application programmer should remain aware of two conditions that exist within the operation of this system. First, allocated space is not guaranteed to be zero-filled. The programmer must take care to fill all data locations with a desired value. Second, the starting address of the data associated with a construct will not remain constant. Addition and deletion of space in a construct may result in the movement of data related to other constructs. An NFIND should be executed to find the new starting address of any construct before it is used. See "Internal Construct Structure" (page C7) for more information about data movement resulting from addition and deletion.

Description of Construct Handling Routines

INITDM

Subroutine INITDM(LENGTH)

This subroutine is called to initialize the data management system. INITDM initializes internal variables which the other routines require and must be the first data management system routine called. The input argument LENGTH is used to specify the length in words of the common block 'FREE' which should be defined in the user's calling program.

```
COMMON/FREE/ CORE(2000)
.
.
CALL INITDM(2000)
```

The entire system can be restarted at any time by calling INITDM again. Note that when restarting there are no longer any constructs in core and all space is available for use.

NADD

Function NADD(ANAME, KLASS, ND1, ND2)

The function NADD is used to create a new construct or add space to an existing construct.

The internal headers are searched to see if the construct ANAME exists. If it does not already exist, a new one is created using the information specified in the arguments KLASS, ND1, and ND2. These may be integer constants or variables. The function value returns the starting address of the new construct.

If the construct already exist, its dimensions are modified by the values ND1 and ND2. In this case, ND1 and ND2 become the change in dimension and may take on positive, zero, or negative values. (NOTE: The programmer should NOT use negative values to remove space; use function NDELET instead. NADD will not check to see if too much space is removed.)

Zero is valid when the programmer wishes to change one dimension only. To make the column vector larger or matrix column longer, ND1 ND2 typically would be NCHANG, 0. This is also true for width changes 0, NCHANG. Values of 0, 0 can be used when changing KLASS values since it is updated each time NADD is called.

Automatic remapping of data occurs when the column length of an array changes. This ensures that the data appear in the same format within the new space. See "Internal Construct Structure" (page C7) for more information.

If no space in 'FREE' is available to create or expand a construct, the function returns the value of -1. If the request is successful, the starting address of the construct is returned.

NFIND

Function NFIND(ANAME, KLASS, ND1, ND2)

The function NFIND searches the internal headers for a construct ANAME. If found, the values of KLASS, ND1, and ND2 are returned, and the function value returns the starting address of the construct. If the construct is not found, zero is returned for variables KLASS, ND1, and ND2 and the function value.

NDELET

Function NDELET(ANAME, NIF, NIL, N2F, N2L)

The function NDELET is used to delete all or part of the data in a construct. Deleting all of the data in a construct in effect causes the deletion of the construct itself.

The variables NIF, NIL, N2F, and N2L are used to specify the range of deletion. NIF to NIL specifies the first to last column vector elements or array row numbers. N2F to N2L specifies the first to last row vector elements or array column numbers.

The following restrictions must be observed when specifying deletion ranges:

N1F	N1L
N2F	N2L
N1L	ND1
N2L	ND2

If these restrictions are not observed, the function will return the error value of -1. A range value or 0 has multiple functions depending on its use. If NIF = NIL = 0, then no deletion will occur in that range. The same is true for N2F = N2L = 0. If N1F is assigned a value and N1L = 0, then only the N1F element or row is deleted; this is equivalent to specifying N1F = N1L. If N1F = N1L = N2F = N2L = 0, the whole construct is deleted; this is equivalent to specifying N1F = N2F = 1, N1L = ND1, N2L = ND2. The preferred method of deleting an entire construct is to specify all zeros for the range. This will result in the function returning 0 as its value. If the specification of a range results in the deletion of the entire construct, the function returns +1 as its value.

Just as the addition of space causes internal reformatting of data, deletion also does. See "Internal Construct Structure" (page C7) for more information.

NRENAM

Function NRENAM(OLDNAM, NEWNAM, KLASS, ND1, ND2)

The function subprogram NRENAM allows the user to change the label portion of the data in a header from 'OLDNAM' to 'NEWNAM'.

The internal headers are searched to see if 'NEWNAM' is being used. If it does already exist, then NRENAM returns with a value of -1. If not located, a search of the headers for 'OLDNAM' is initiated.

When 'OLDNAM' is located, it is replaced with 'NEWNAM'; the variables KLASS, ND1, and ND2 are set to the corresponding values found with the 'OLDNAM' construct; and NRENAM is set equal to the starting address of the construct.

NSERCH and NLIST

Subroutine NSERCH(IKLS1, IKLS2)

Function NLIST(ANAME, KLASS, ND1, ND2, IUSE)

NSERCH and NLIST allow the programmer to determine the name and other information about all constructs whose KLASS values lie within a specified range. The purpose of subroutine NSERCH is to define the range of KLASS values: IKLS1 to IKLS2. The default range at system initialization is 999 to 999. Calling NSERCH also clears the internal access count in the header of all constructs that are within the range IKLS1 to IKLS2. More information about this count is described below with variable IUSE. The value of IKLS1 need not be less than IKLS2; the program will check for any construct KLASS between IKLS1 and IKLS2.

The purpose of function NLIST is to return the starting address, name, class, dimensions, and use count for each construct within the search range. The function value returns the starting address, and other information is returned through the arguments. After calling NSERCH to specify a search range, NLIST is called repeatedly until a function value of 0 is returned. Zero signifies that the entire contents of the common block 'FREE' have been searched

and no more constructs with KLASS must be called again to reinitialize the search pointers and range before NLIST can find any constructs.

The following example illustrates the procedure:

```
      .
      .
C..   GET INFO ABOUT CONSTRUCTS KLASS = (20,40)
      CALL NSERCH( 20, 40 )
100   LOC = NLIST( ANAME, KLASS, ND1, ND2, IUSE)
      IF( LOC .EQ .0 ) GO TO 200
C..   PROCESS INFORMATION
      .
      .
      GO TO 100
C..   DONE, SET NEW RANGE
200   CALL NSERCH( 10, 5 )
      .
      .
```

The internal access count flag for a construct is incremented each time NLIST references that construct. The value returned through IUSE is before the increment is done; thus, a nonzero value signals the programmer that this information has already been received.

Utility Routines NMOVCR, NMVBLK, and NREMAP

These three subroutines are utility subroutines used by the system to move rows and columns within an array, to move blocks of memory in 'FREE', and to reformat arrays.

Internal Construct Structure

Each construct is made up of two parts: headers and data space. The data space is the section of memory where the programmer's data actually exist. The headers are blocks of memory used to keep tract of the information related to each construct and its data. The data and headers for each construct are maintained by this system within the user-provided common block 'FREE'. Figure C1 shows a diagram of the typical common block structure. Location 1 is considered to be the "bottom," and location LENGTH is the "top" of the common block. The data spaces are allocated from the bottom in variable lengths

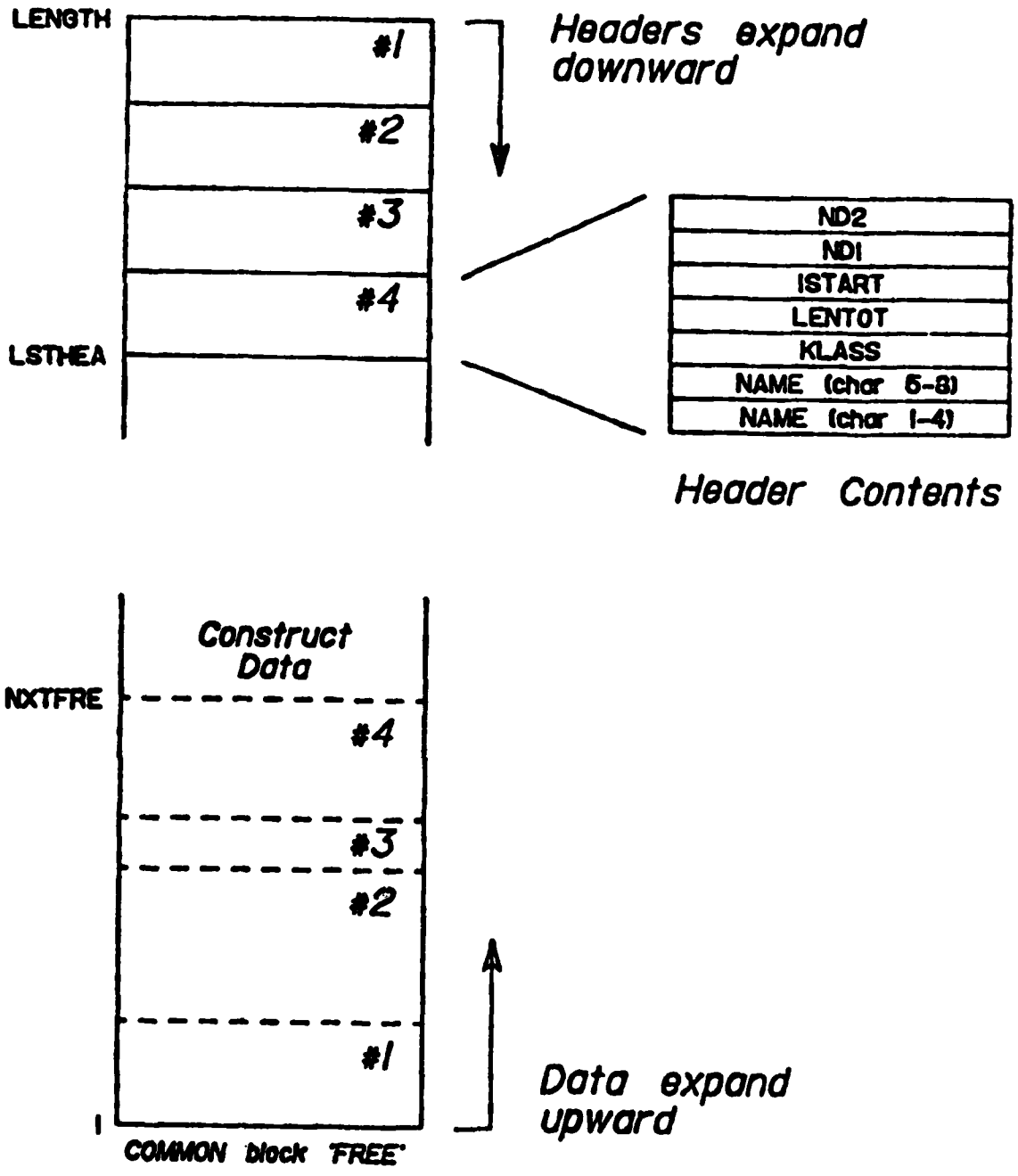


Figure C1. Example of DATMNG common block structure showing headers and data of four constructs

expanding upward. The headers are allocated from the top in fixed lengths expanding downward.

The insert in Figure C1 shows the contents of each header. Seven words of memory are used to store the following information:

2 words for the constructs name - ANAME

1 word for the class value - KLASS

1 word for the starting address of the data - ISTART

2 words for the two dimensions - ND1 and ND2

Addition of Space to a Construct

The addition of space to a construct results in the upward movement of the data above it. This provides a "hole" in memory for the new data to be placed. Figure C2 illustrates this concept. The ISTART variables in the headers for the constructs that are moved are updated to account for the

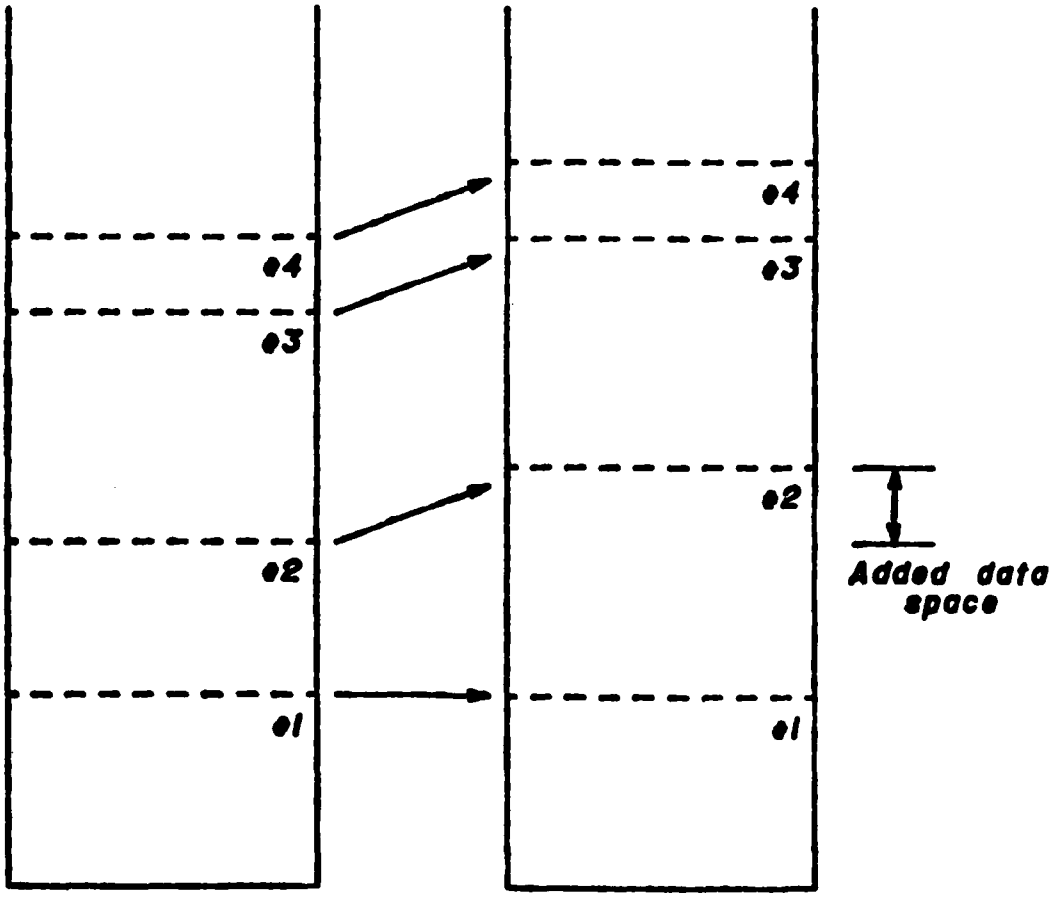


Figure C2. Example showing addition of data space to a construct

movement of these data. The addition of space to a construct does not result in the movement of that particular construct because space is always allocated at the end of the existing data. Note that the creation of a new construct causes space to be allocated at the end and causes no movement of data.

Deletion of Space from a Construct

The deletion of space has the opposite effect from that described above. The data of the constructs above the space to be deleted are moved down over that space (see Figure C3). Again, the ISTART variable in the headers of the

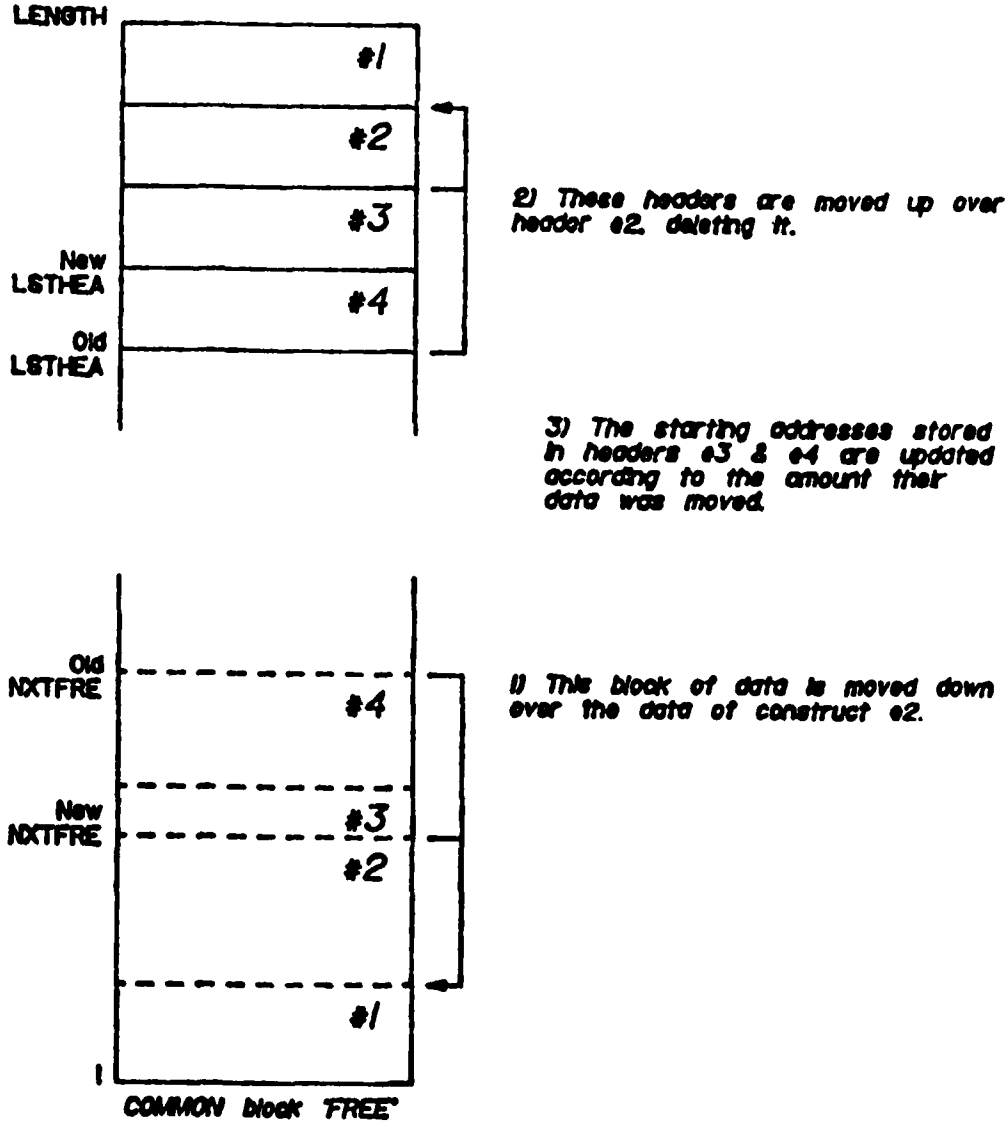


Figure C3. Example of DATMNG common block during construct deletion

moved constructs is updated to contain the new addresses of their data. If an entire construct is deleted, its header is deleted by moving the headers below it upward (Figure C3). As with addition, deletion of space does not result in the movement of the construct being acted upon.

Reformatting of Arrays

When space is added to an array by changing the column length, the contents of the array must be reformatted. This guarantees that any element of an array, A_{ij}, will remain in that position when accessed under the new dimensions.

Figure C4 shows the operation of changing a 3×3 array to a 4×3 array as its contents appear to a typical programmer. After the addition of space, the elements from A_{12} downward are no longer in the correct location within a 4×3 array format. The contents of locations A_{12} through A_{33} must be moved to their new locations as shown by the arrays in Figure C4.

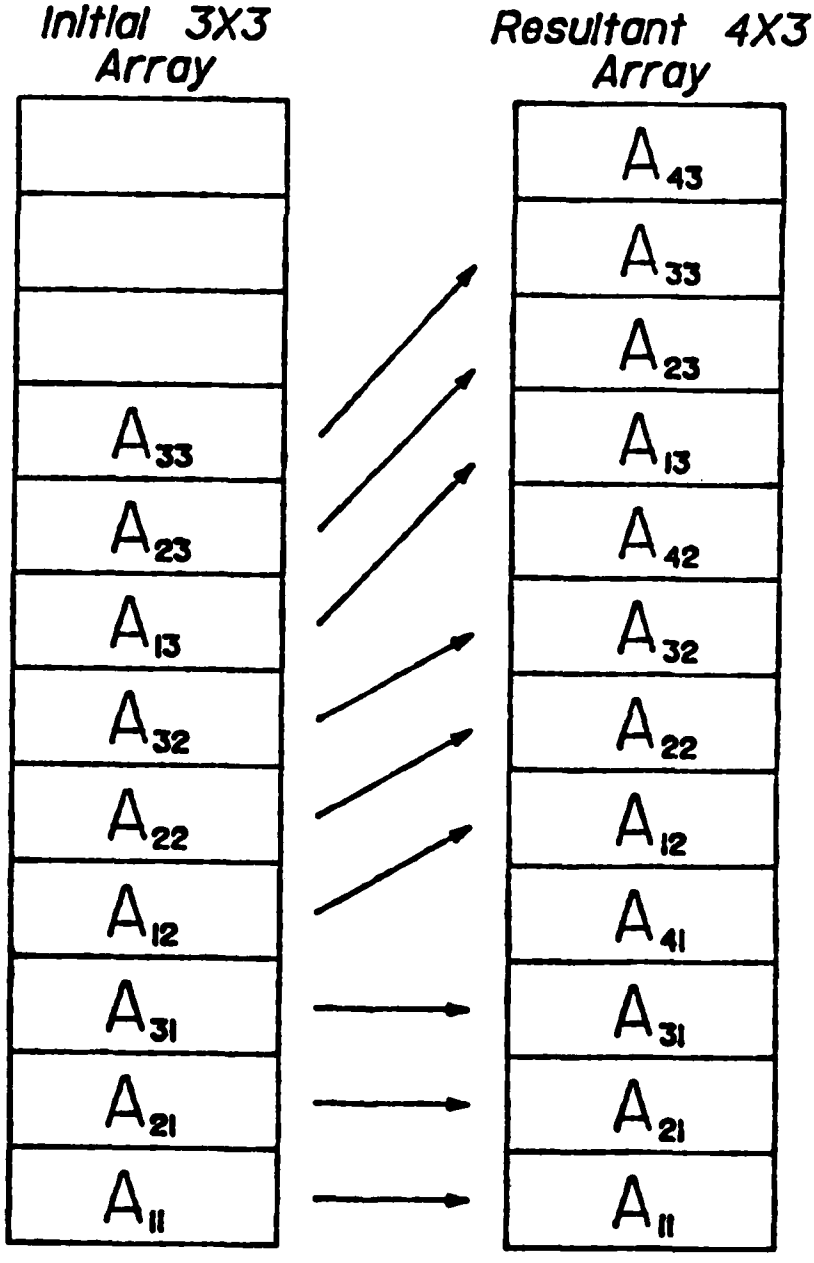


Figure C4. Reformatting of DATMNG arrays

APPENDIX D: COMMAND PREPROCESSOR LSUCMD
APPLICATION PROGRAMMER'S GUIDE

LSUCMD is a FORTRAN subroutine which accepts a command string input by the user and changes it to a compact usable form. Words and numbers of a user command are identified and reformatted such that flexible command interpreters can be quickly constructed by an application programmer. Information is passed back to the calling program through the use of a common block called CMD.

LSUCMD checks the command string for syntax errors such as multiple decimal points, an imbedded character, or a misplaced hyphen in a number. These features allow user errors without producing catastrophic results.

The command preprocessor will return to the calling program after every word (up to 32 characters) of the command string in a packed format. All integer and real numbers are returned in floating point format along with the words in a single array (ALPNU). The data are identified through the use of the ICODE array and positioned in accordance with the IPOS array (see the detailed description of variables).

The command preprocessor is called as follows:

```
CALL LSUCMD
```

There are no arguments as all information is passed back through the common block CMD. The common block CMD used to pass information to the user must be included at the top of the user's program. The common statement should read:

```
COMMON/CMD/ALPUT(40),ICODE(40),NWDS,NOS
CALL CMDSET (MAXWDS,MAXNOS)
```

The maximum number of words and the maximum number of numbers are redefined by positive values, including zero, of MAXWDS and MAXNOS, respectively. Negative values of MAXWDS or MAXNOS cause the previous value to be retained.

As mentioned previously, subroutine LSUCMD accepts up to 32 characters for each command word typed by the program user. However, the application programmer may wish to recognize only the first few characters of a given

word, thus providing a more flexible and less restrictive environment for the user. Subroutine MASK allows the application programmer to extract the specific number of characters he requires for a particular word. The call to MASK is as follows:

SUBROUTINE MASK (NWORD, NC, TO, NB)

The variables in the argument list are as follows:

NWORD - position of the word in the command string

NC - number of characters to be extracted from the indicated word

TO - name of the variable or array in which the extracted characters are placed, 1 character per byte. The user should be certain that this word is large enough to handle NC characters

NB - the maximum number of characters the variable or array TO can hold. Any unused bytes in TO are filled with blank characters

These variables in CMD are utilized as follows:

ALPNU(40) - alphanumeric array that contains up to 32 characters of every word in the command string packed 4 characters per 4-byte word. It also contains, in real number format, all numbers in the command string. If an integer number is required, the real number must be converted to an integer in the application program

ICODE(40) - array that denotes the type of information contained in ALPNU(40). If ICODE(n) is positive, it indicates that a word was found at the n^{th} position in the command string. The value of ICODE(n) denotes the number of characters found in that word. The maximum number of characters per word is 32; any more will be ignored. If ICODE(n) is a negative one (-1), a number was found at the n^{th} position in the command string. Allowable delimiters between words or numbers are a blank, a comma, or an equal sign

IPOS(40) - array that denotes the position in the ALPNU(40) array of the beginning of a word or a number from the command string. The IPOS(40) and ICODE(40) arrays correspond to each other in a one-to-one relationship. For instance, if ICODE(2) is a positive five and IPOS(2) is three, this indicates that the second word or number in the command

string is a word with 5 characters and the first 4 characters are located in ALPNU(3) and the last character followed by 3 blanks is located in ALPNU(4)

NWDS - integer *2 variable - denotes the number of words found in the command string

NOS - integer *2 variable - denotes the number of numbers found in the command string

An initialization routine called CMDINT must be called by the user prior to calling LSUCMD. This routine contains no arguments. It defines the maximum number of words and the maximum number of numbers per command string to be 20 each. If the user wishes to change these limits, he may do so by calling CMDSET.

APPENDIX E: ERROR MESSAGES

Error No.	Description of Error
0	Initialized value; no errors in execution
101	Coordinate table for the specified object does not exist
102	Line table for the specified object does not exist
103	Internal conflict with KLASS of the coordinate table; i.e., KLASS does not equal 21
104	Internal conflict with KLASS of the line table; i.e., KLASS does not equal 22
105	Internal conflict with KLASS of dimensioning information; i.e., extension line type does not match the dimension table type. Also, in curve routines, cannot find curve table for specified object
106	Trouble with deleting data contained in face information
110	Object specified does not exist
111	Invalid object KLASS; not equal to 50
112	No active objects in display list
113	In CVTDIM, could not change object KLASS to 52; i.e., could not change object from two- or three-dimensional
114	Invalid object KLASS; not equal to 50 or 52
121	Coordinate table for object specified does not exist in the object definition
122	Line table for the object specified does not exist in the object definition
131	Extension line coordinate table for the object specified does not exist in the object definition
132	Extension line table for the object specified does not exist in the object definition
191	Syntax error in the use of the MERGE command
192	Improper use of the word INTO in the MERGE command
200	Not enough memory in the FREE array to allow full processing of the command
201	Data have been shifted after an NADD and pointer have not been reset
202	Location of face table problems
203	No room to add face information
204	No room to add segment to current face definition

(Continued)

Error No.	Description of Error
300	Calculation error; division by zero avoided and 0 value returned, or the distance between two points is zero
395	Could not find plane No. 1 in first portion of a string in dimensioning information
396	Trouble matching values
397	Trouble searching dimensioning information
398	Trouble searching dimensioning information; the number of dimensioning planes is 0
399	Coordinate table does not have 5 or 8 units per coordinate
400	Trouble setting points in particular planes in subroutine DETPNS; check line types
401	Trouble setting points in particular planes in subroutine DETPNS; at least one point has not been set, possibly a free-standing point; check line types
402	Dimensioning planes have not been established; cannot go on with dimensioning procedure
403	Trouble establishing extension lines in CRELIN; also, could not locate dimensioning tables in routine DIMENS
404	User aborted dimensioning procedure before complete specification of information
405	Both coordinate pairs of an extension line definition are on a boundary
406	A line type for an extension line is not properly set
407	Could not find dimensions array construct
408	Internal problem; data have not been moved correctly
409	Trouble deleting extra records in dimensioning array when connecting dimension strings together
410	Number of strings value in dimensioning array is bad
411	Trouble trying to rearrange the dimensioning planes according to coordinate values
500	Could not find reference plane in any dimensioning string
501	Could not find a string with at least one point set
502	Counter LO is less than LOLIMIT
600	Improper file specification in SAVE command
700	Polygon order cannot be reversed in FACGEN logic
999	Coordinate table does not have 5 or 8 units per coordinate

WATERWAYS EXPERIMENT STATION REPORTS PUBLISHED UNDER THE COMPUTER-AIDED STRUCTURAL ENGINEERING (CASE) PROJECT

	Title	Date
Technical Report K-78-1	List of Computer Programs for Computer-Aided Structural Engineering	Feb 1978
Instruction Report O-79-2	User's Guide: Computer Program with Interactive Graphics for Analysis of Plane Frame Structures (CFRAME)	Mar 1979
Technical Report K-80-1	Survey of Bridge-Oriented Design Software	Jan 1980
Technical Report K-80-2	Evaluation of Computer Programs for the Design/Analysis of Highway and Railway Bridges	Jan 1980
Instruction Report K-80-1	User's Guide: Computer Program for Design/Review of Curvilinear Conduits/Culverts (CURCON)	Feb 1980
Instruction Report K-80-3	A Three-Dimensional Finite Element Data Edit Program	Mar 1980
Instruction Report K-80-4	A Three-Dimensional Stability Analysis/Design Program (3DSAD)	
	Report 1: General Geometry Module	Jun 1980
	Report 3: General Analysis Module (CGAM)	Jun 1982
Instruction Report K-80-6	Basic User's Guide: Computer Program for Design and Analysis of Inverted-T Retaining Walls and Floodwalls (TWDA)	Dec 1980
Instruction Report K-80-7	User's Reference Manual: Computer Program for Design and Analysis of Inverted-T Retaining Walls and Floodwalls (TWDA)	Dec 1980
Technical Report K-80-4	Documentation of Finite Element Analyses	
	Report 1: Longview Outlet Works Conduit	Dec 1980
	Report 2: Anchored Wall Monolith, Bay Springs Lock	Dec 1980
Technical Report K-80-5	Basic Pile Group Behavior	Dec 1980
Instruction Report K-81-2	User's Guide: Computer Program for Design and Analysis of Sheet Pile Walls by Classical Methods(CSHTWAL)	
	Report 1: Computational Processes	Feb 1981
	Report 2: Interactive Graphics Options	Mar 1981
Instruction Report K-81-3	Validation Report: Computer Program for Design and Analysis of Inverted-T Retaining Walls and Floodwalls (TWDA)	Feb 1981
Instruction Report K-81-4	User's Guide: Computer Program for Design and Analysis of Cast-in-Place Tunnel Linings (NEWTUN)	Mar 1981
Instruction Report K-81-6	User's Guide: Computer Program for Optimum Nonlinear Dynamic Design of Reinforced Concrete Slabs Under Blast Loading (CBARCS)	Mar 1981
Instruction Report K-81-7	User's Guide: Computer Program for Design or Investigation of Orthogonal Culverts (CORTCUL)	Mar 1981
Instruction Report K-81-9	User's Guide: Computer Program for Three-Dimensional Analysis of Building Systems (CTABS80)	Aug 1981
Technical Report K-81-2	Theoretical Basis for CTABS80: A Computer Program for Three-Dimensional Analysis of Building Systems	Sep 1981
Instruction Report K-82-6	User's Guide: Computer Program for Analysis of Beam-Column Structures with Nonlinear Supports (CBEAMC)	Jun 1982
Instruction Report K-82-7	User's Guide: Computer Program for Bearing Capacity Analysis of Shallow Foundations (CBEAR)	Jun 1982
Instruction Report K-83-1	User's Guide: Computer Program With Interactive Graphics for Analysis of Plane Frame Structures (CFRAME)	Jan 1983
Instruction Report K-83-2	User's Guide: Computer Program for Generation of Engineering Geometry (SKETCH)	Jun 1983
Technical Report K-83-1	Basic Pile Group Behavior	Jun 1983