1-1-1982

A comparative survey of computer graphics applications in mechanical design.

Nell Carroll Cates

A COMPARATIVE SURVEY OF COMPUTER GRAPHICS APPLICATIONS IN MECHANICAL DESIGN

by

Neil Carroll Cates, III

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science
in
Mechanical Engineering

Lehigh University
1982

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

14 May 1982
date

Professor in Charge

Chairman of Department

ACKNOWLEDGEMENTS

The author wishes to acknowledge partial support of this work by the Control Data Corporation under PLATO award entitled CAD/CAM in Industry, SRA File No. 79L01.

TABLE OF CONTENTS

CHAPTER			PAGE
1.0	NUMERICAL WIREFRAME GEOMETRIC MODELING		3
	1.1	Numerical Techniques	3
	1.2	Geometric Entity Description	5
	1.3	Entity Data Base Structure	19
2.0	WIREFRAME GRAPHICS SYSTEM STRUCTURE		28
	2.1	Principal Hardware/Software Components	28
	2.2	Display Devices	30
	2.3	Device-Independent Systems	34
3.0	WIREFRAME IMAGE GENERATION		41
	3.1	Clipping	42
	3.2	Projection	53
		3.21 Orthographic Projection	53
		3.22 Perspective Projection	65
		3.23 Multiple Views	69
	3.3	Summary	73
4.0	INTERACTION BETWEEN OPERATOR AND SYSTEM		75
	4.1	Graphical Input	76
	4.2	Entity Selection	84
	4.3	Function Selection	85
		4.31 Menu-Oriented Systems	85
		4.32 Command Language Systems	86
	4.4	Graphical Interaction	87
5.0	MODEL CONSTRUCTION ENHANCEMENTS		89
	5.1	Geometric Relational Expressions	89
	5.2	Entity Duplication	92
	5.3	Special Purpose Procedures	100
	5.4	Data Base Accessibility	101

LIST OF TABLES

		PAGE
1-1	Wireframe Model Vertices	7
1-2	Wireframe Model Edges	9
1-3	Hidden-line Calculation	15
1-4	Wireframe Model with Arcs	21
1-5	Sample Data Base	23
1-6	Minimum Entity Data Formats	25
1-7	Sample Model File	27
2-1	Sample Display File	40
5-1	Line Entity Definition Options	90

LIST OF FIGURES

		PAGE
1-1	Two-level Wireframe Model	6
1-2	Two-level Topological Graph	10
1-3	Three-level Wireframe Model	12
1-4	Three-level Topological Graph	13
1-5	Wireframe Model Rotated View	14
1-6	Face Definition Vertex Ordering	16
1-7	Wireframe Model with Arcs	20
1-8	Sample Wireframe Model	24
2-1	System Structure	29
2-2	Wireframe Model Raster Image	35
2-3	Wireframe Model Calligraphic Image	36
2-4	Sample Calligraphic Image	39
3-1	2-D Clipping	43
3-2	Clipped Screen Image	46
3-3	Clipping in 3-D, Method 1	48
3-4	Clipping in 3-D, Method 2	49
3-5	Enlarged View Cube	51
3-6	Clipping Side-effect	52
3-7	Orthographic Projection	54
3-8	Trimetric Projection Process	58

3-9	Trimetric Screen Image	59
3-10	Dimetric Projection Process	60
3-11	Dimetric Screen Image	61
3-12	Isometric Projection Process	62
3-13	Isometric Screen Image	63
3-14	Perspective Projection	67
3-15	Perspective Screen Image	68
3-16	Four-View Display, Front X-Y	70
3-17	Four-View Display, Top X-Y	72
3-18	Image Generation Sequence	74
4-1	Auxiliary Coordinate System	77
4-2	Digitizing Plane Orientation	79
4-3	Skewed-Line Entry	81
4-4	Digitizing Grid	83
5-1	Model Construction, Planar Techniques (1)	94
5-2	Model Construction, Planar Techniques (2)	95
5-3	Mirroring Procedure	97
5-4	Rectangular Array Procedure	98
5-5	Circular Array Procedure	99

ABSTRACT

Current applications of computer graphics in mechanical engineering design are reviewed. The organization of numerical 3-D wireframe geometric models is related to both data requirements and model completeness. Typical data base formats are described for both hierarchical and list-oriented forms. A basic turnkey system structure is then introduced, principal hardware and software components and display device-independence. The characteristics of current display technologies are compared for engineering applications in design, drafting and analysis. The basic wireframe graphics procedures for image generation then are developed from this foundation, including clipping and projection. A generalized image generation process is described for the production of arbitrarily scaled and rotated single and multiple related sets of views from 3-D model data.

The way these basic principles combine in the interaction between operator and machine is then examined. Implications of both menu-oriented and command language-based systems are described in terms of engineering requirements and operator convenience. Uses of graphical interaction in the specification of system commands and model geometric data are described in their relation to model construction techniques. Common economics in model construction afforded by referencing and transforming existing

data are discussed. Finally, the implications of integrated engineering design, analysis and manufacturing are examined regarding computing requirements and the accessibility of the geometric model data base.

1.0 Numerical Wireframe Geometric Modeling

General-purpose 3-D computer graphics systems depend on the existence of numerical models which describe the geometric configuration of arbitrary 3-D objects. In nearly all current systems so-called wire-form or wireframe models are used. Here the locations of points and lines are specified relative to a 3-dimensional cartesian coordinate system, alternately referred to by the names *world*, *absolute* or *model* coordinates. The origin of this coordinate system, the direction of its axes and frequently the units of measurement can be freely chosen in most current systems to present conditions most convenient for a particular object.

1.1 Numerical Techniques

The size of the 3-D modeling space is limited by the forms of internal data representation that are supported by the host computer. Systems running on smaller machines often use integer data types, while some more recent versions using 32-bit processors have implemented floating-point data types. In both cases, the maximum and minimum coordinate values are restricted by the number of bits used to represent them. For example, in one system which uses 24-bit integers to represent coordinate values, the corresponding range of representable values in two's

MICRODEX CORRECTION GUIDE (M-8)

CORRECTION

The preceding document has been re-photographed to assure legibility and its image appears immediately hereafter.

[illegible]

REMINGTON RAND
OFFICE SYSTEMS DIVISION

1.0 Numerical Wireframe Geometric Modeling

General-purpose 3-D computer graphics systems depend on the existence of numerical models which describe the geometric configuration of arbitrary 3-D objects. In nearly all current systems so-called wire-form or wireframe models are used. Here the locations of points and lines are specified relative to a 3-dimensional cartesian coordinate system, alternately referred to by the names *world*, *absolute* or *model* coordinates. The origin of this coordinate system, the direction of its axes and frequently the units of measurement can be freely chosen in most current systems to present conditions most convenient for a particular object.

1.1 Numerical Techniques

The size of the 3-D modeling space is limited by the forms of internal data representation that are supported by the host computer. Systems running on smaller machines often use integer data types, while some more recent versions using 32-bit processors have implemented floating-point data types. In both cases, the maximum and minimum coordinate values are restricted by the number of bits used to represent them. For example, in one system which uses 24-bit integers to represent coordinate values, the corresponding range of representable values in two's complement notation is:

$$-2^{(24-1)} \text{ to } 2^{(24-1)} -1 = -8388608_{10} \text{ to } +8388607_{10}$$

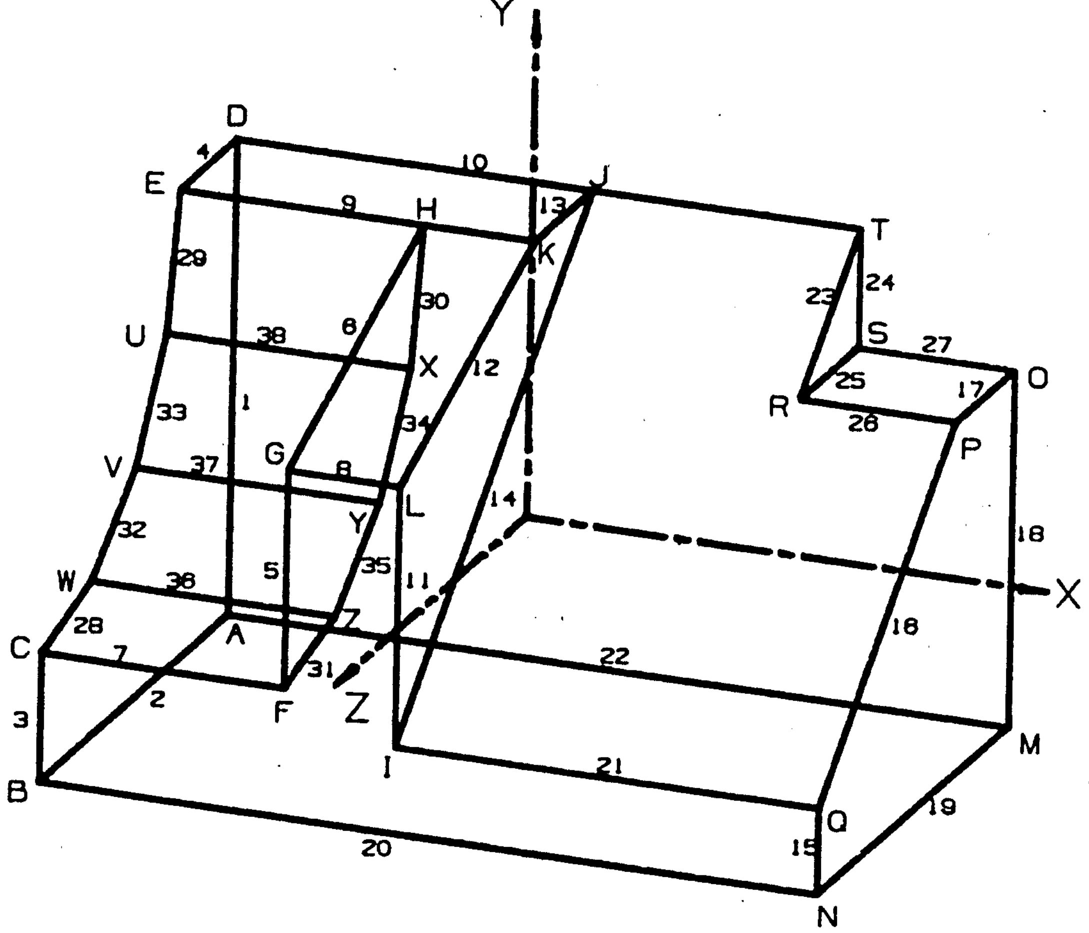
If a point-to-point resolution of 0.0001 inches is required, this range expressed in standard units is -838.8608 inches to +838.8607 inches. No other values are representable, and precision is maintained in all integer arithmetic operations excepting division, which can introduce truncation [8]. Coordinates are normally entered using standard units in floating-point format, and are subsequently converted to integer form in model space using the given resolution factor.

In contrast, one 32-bit floating-point representation scheme uses an 8-bit excess-128 coded exponent and 24-bit normalized-fraction mantissa having approximately 7 decimal digits precision; the range of possible values is $.29(10)^{-38}$ through $1.7(10)^{38}$. Although this allowable range is far more extensive than the integer example, the loss of significance which can result from all floating-point arithmetic operations implies that precision is assured with the above 0.0001 inch resolution only when coordinate values remain within the range -999.9999 inches to 999.9999 inches. In these systems, values entered in floating-point form are often simply stored directly; any values less than 0.0001 thus may be lost in subsequent arithmetic.

1.2 Geometric Entity Description

The requirements of conventional engineering drafting normally can be satisfied by wireframe geometric modeling. In one simple scheme, objects are represented as polyhedral solids; arcs are replaced by a sequence of straight lines. This latter form of approximation is routinely applied in the rendering of an image on a CRT screen, and, hence, is not a restriction for drafting purposes.

A wireframe model comprises a collection of vertices and edges, which correspond graphically to points and lines, respectively. In the most primitive form of wireframe modeling, moreover, there is no explicit use of the point entity; points are referred to implicitly only as line endpoints, and the resulting image consists entirely of a collection of straight lines. Figure 1-1 depicts such a simple model. The object vertices have been labeled and floating-point numerical values have been assigned for each vertex (see Table 1-1) with the object positioned as shown in a 3-dimensional cartesian coordinate system. Orientation of the object with respect to the coordinate system origin is completely arbitrary; for reasons to be shown later, it is often most convenient for the user to align one face of the object parallel to one of the principal axes. Many current wireframe systems permit the user to make reference to auxiliary


Y
D
4
10
E
J
9
H
13
T
K
29
30
24
23
U
38
6
S
27
X
12
25
17
O
R
33
1
34
26
P
V
37
G
8
L
14
32
Y
18
W
36
5
35
11
X
28
A
Z
C
7
31
22
16
2
F
Z
3
M
I
21
B
19
Q
15
20
N

VERTEX	COORDINATES			VERTEX	COORDINATES		
	X	Y	Z		X	Y	Z
A	-150.	-94.	-99.	N	150.	-94.	99.
B	-150.	-94.	99.	O	150.	47.	-99.
C	-150.	-44.	99.	P	150.	47.	-38.
D	-150.	94.	-99.	Q	150.	-60.	99.
E	-150.	94.	-39.	R	90.	47.	-38.
F	-56.	-44.	99.	S	90.	47.	-99.
G	-56.	43.	99.	T	90.	94.	-99.
H	-56.	94.	-39.	U	-150.	41.	-28.
I	-13.	-60.	99.	V	-150.	-2.	2.
J	-13.	94.	-99.	W	-150.	-32.	47.
K	-13.	94.	-39.	X	-56.	41.	-28.
L	-13.	43.	99.	Y	-56.	-2.	2.
M	150.	-94.	-99.	Z	-56.	-32.	47.

TABLE 1-1. Wireframe Model Vertices

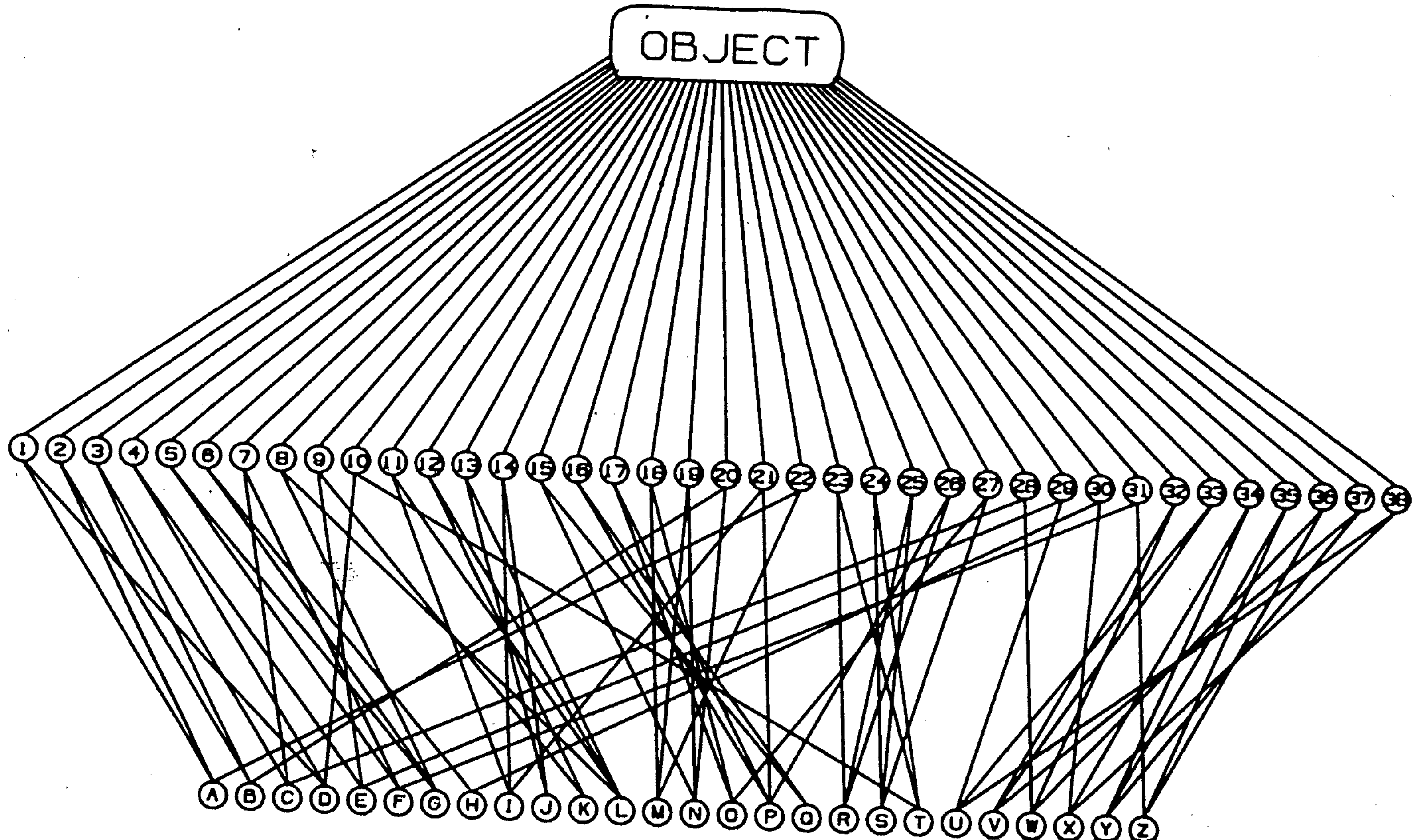
coordinate systems to facilitate entity specification and the various graphics display devices each necessarily involve their own particular hardware-fixed coordinate systems. All must ultimately be mapped by software into the model coordinate system when defining geometry; the inverse mapping must likewise be performed when generating graphical output.

After establishing the 3 coordinates for each vertex, the existence of edges can be simply indicated by including a data structure which defines the topology or "connectivity" of the object. Table 1-2 lists an integer array specifying the edges for the object shown in figure 1-1. The two letters in each row are the beginning and end vertices for each line. Order is clearly unimportant for modeling purposes, but often can be optimized for drawing speed.

This form of hierarchical model structure is conveniently illustrated by a topological graph; such a graph has been drawn for the object of figure 1-1, and is shown in figure 1-2. For the simple wireframe modeling technique described above, an object is first defined in terms of a discrete number of bounding edges, which are then further subdivided into beginning and end vertices. The utilization of separate data structures for each topological level takes advantage of the fact that vertices are necessarily shared by a number of edges, and, hence, avoids

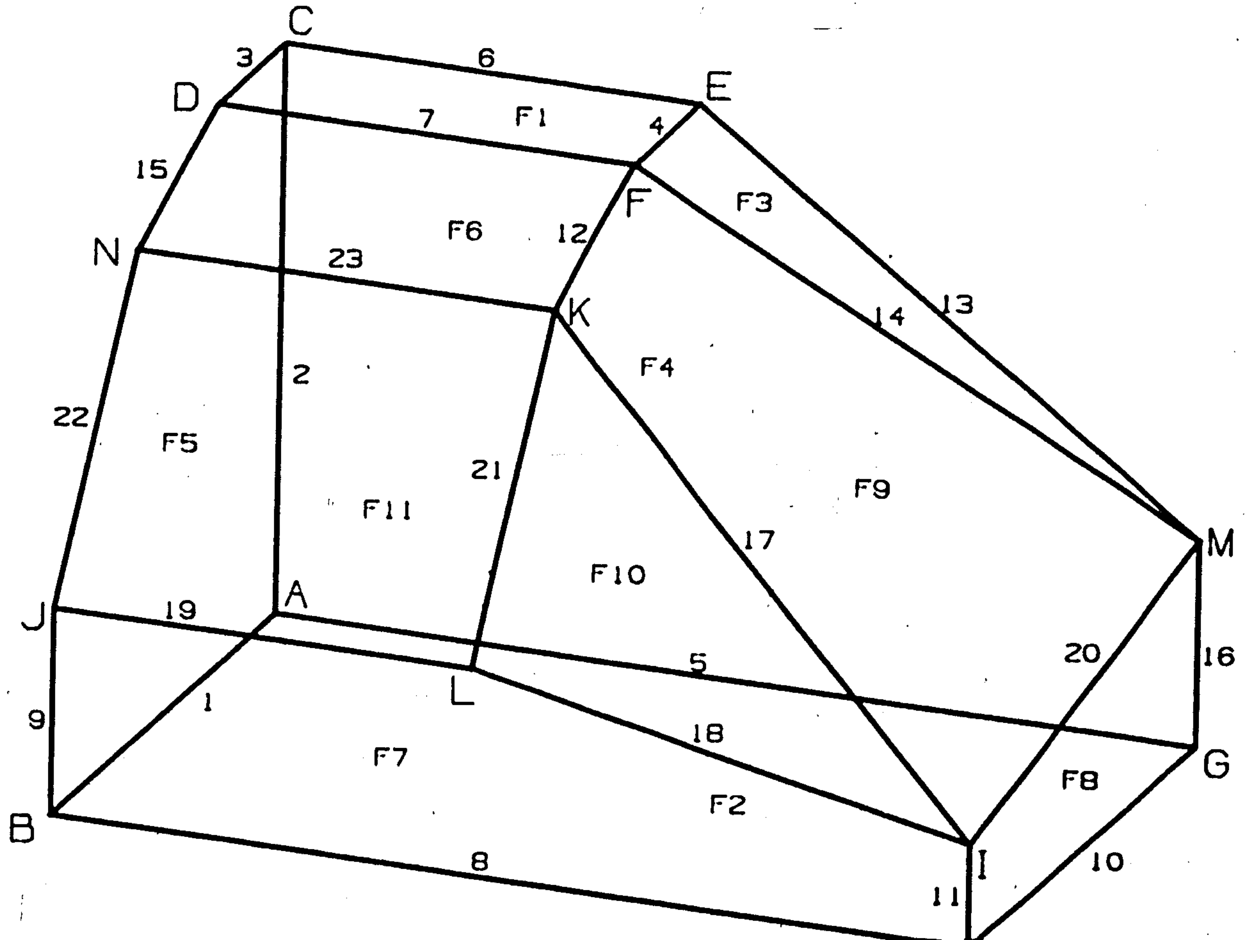
LINE	VERTICES		LINE	VERTICES	
	START	END		START	END
1	D	A	20	B	N
2	A	B	21	I	Q
3	B	C	22	A	M
4	E	D	23	R	T
5	F	G	24	T	S
6	G	H	25	S	R
7	C	F	26	R	P
8	G	L	27	S	O
9	E	K	28	C	W
10	D	T	29	U	E
11	I	L	30	H	X
12	L	K	31	Z	F
13	K	J	32	W	V
14	I	J	33	V	U
15	N	Q	34	X	Y
16	Q	P	35	Y	Z
17	P	O	36	W	Z
18	O	M	37	V	Y
19	M	N	38	U	X

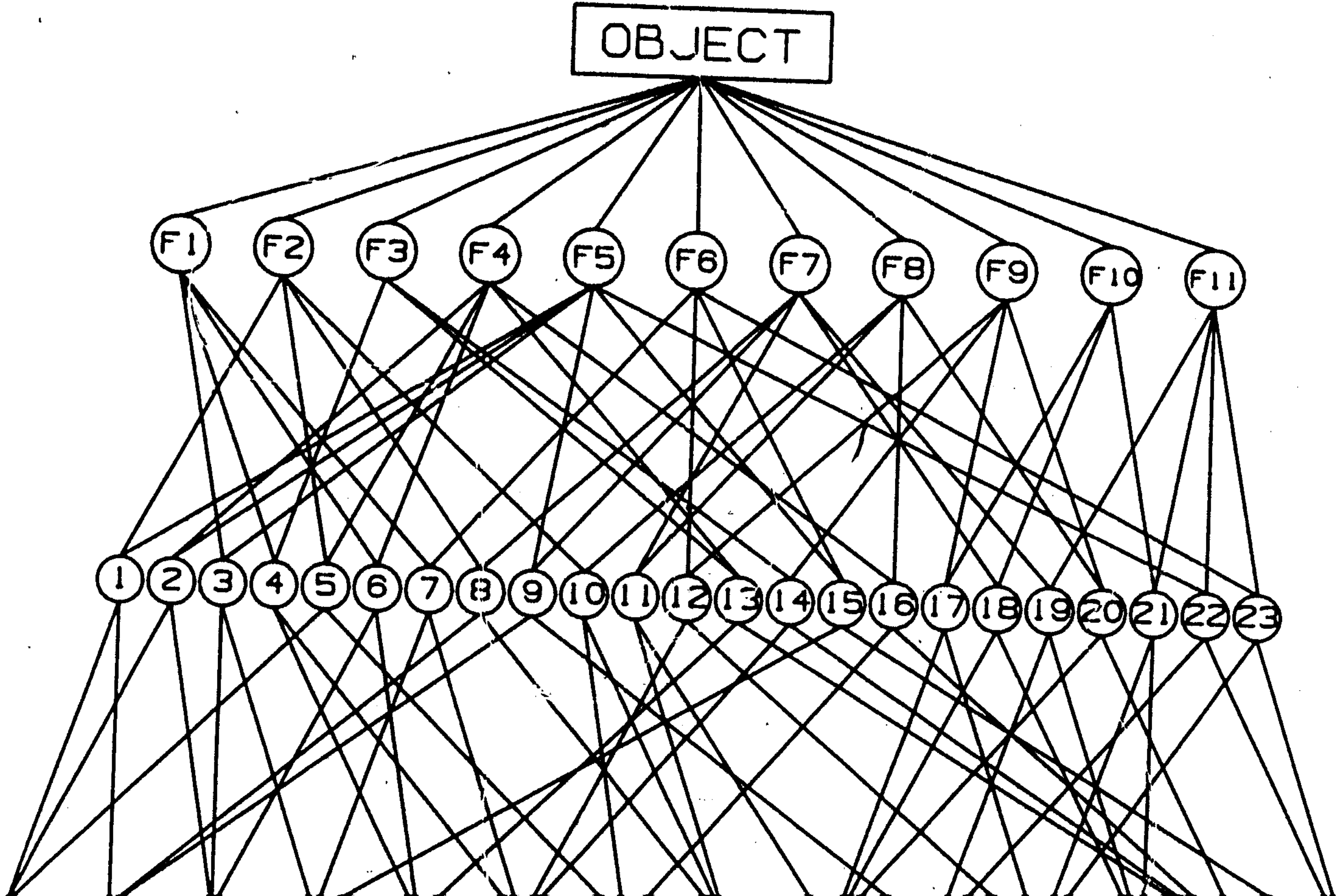
TABLE 1-2. Wireframe Model Edges.

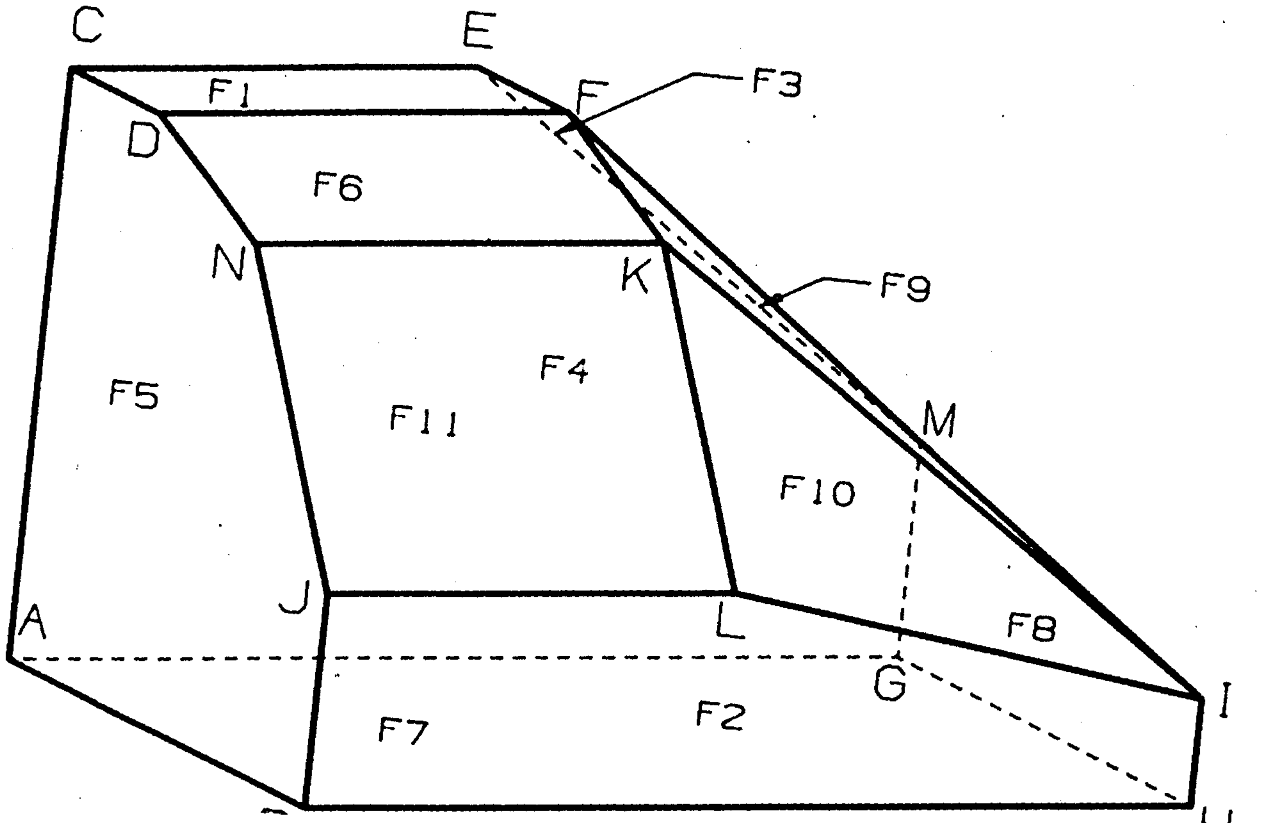

OBJECT
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

duplication [1]. This simplified wireframe model form is generally not used in wireframe drafting systems, but is sufficient for object definition and is sometimes more efficient for applications involving high-speed dynamic graphics.

More elaborate wireframe models include an additional level which defines the several faces of the object. Figures 1-3 and 1-4 show such an object and its topological graph. Information concerning the faces of a wireframe model is normally required only when certain forms of *hidden-line* processing are used, and usually is not maintained in wireframe CAD systems. The progress of one such hidden-line algorithm from this face information is shown in Table 1-3 operating on the image data for figure 1-5. Here all faces are defined directly in terms of the bounding vertices, arranged in a particular order; the vertices are listed in a sequence which traces a counter-clockwise motion when that face's outer surface faces the observer, and thus is visible. This ordering process is shown in figure 1-6a for the object in figure 1-3. Figure 1-6b illustrates the result of calculating a rotated projection of the same object. A normal vector extending from the outer surface of the sample face now subtends an angle greater than 90^{o} with the line of sight, and the original order now traces a clockwise motion according to the observer.


C
D
E
F
N
K
J
A
L
B
I
M
G
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11


OBJECT
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23


C
E
F1
F
F3
D
F6
N
K
F9
F5
F4
F11
M
F10
J
A
L
F8
G
I
F7
F2
H

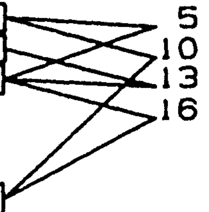
PROJECTED POINT COORDINATES

	X	Y
A	-179.	-88.
B	-86.	-125.
C	-179.	95.
D	-151.	84.
E	-58.	109.
F	-30.	98.
G	86.	-59.
H	179.	-95.
I	179.	-62.
J	-86.	-58.
K	3.	61.
L	35.	-45.
M	86.	7.
N	-118.	47.

FACE DATA

	VERTICES, CCW	EDGES	AREA
F1	CDFEC	3-4-6-7	3446.
F2	AGHBA	1-5-8-10	$\boxed{-24832.}$
F3	EFME	4-13-14	$\boxed{-1272.}$
F4	ACEMGA	2-5-6-13-16	$\boxed{-80343.}$
F5	ABJNDCA	1-2-3-9-15-22	28792.
F6	DNKFD	7-12-15-23	9878.
F7	BHILJ	8-9-11-18-19	24667.
F8	GMIHG	10-11-16-20	$\boxed{-9207.}$
F9	FKIMF	12-14-17-20	1994.
F10	IKLI	17-18-21	14720.
F11	LKNJL	19-21-22-23	26395.

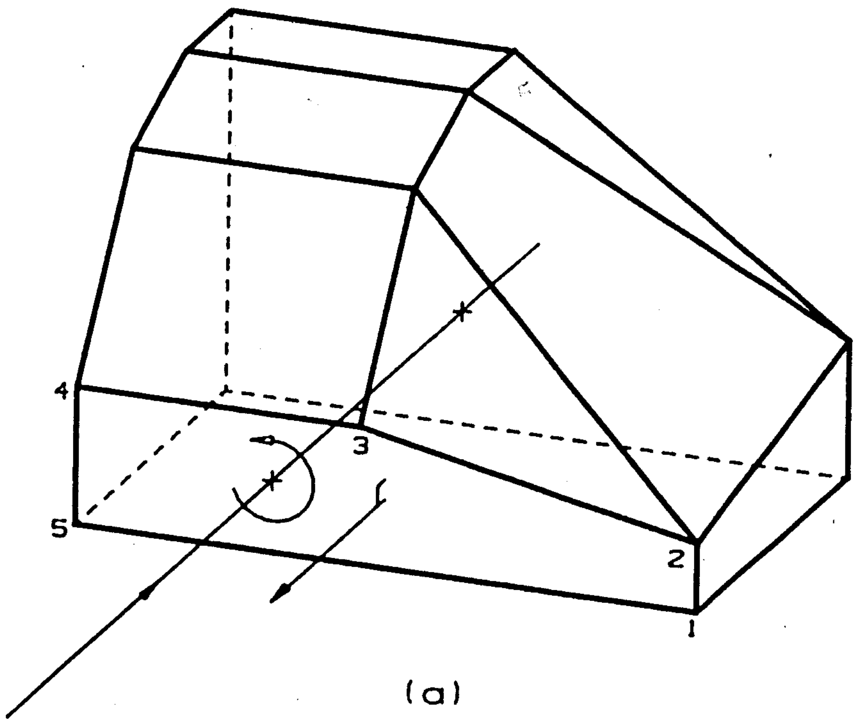
LINES SHARED (HIDDEN)



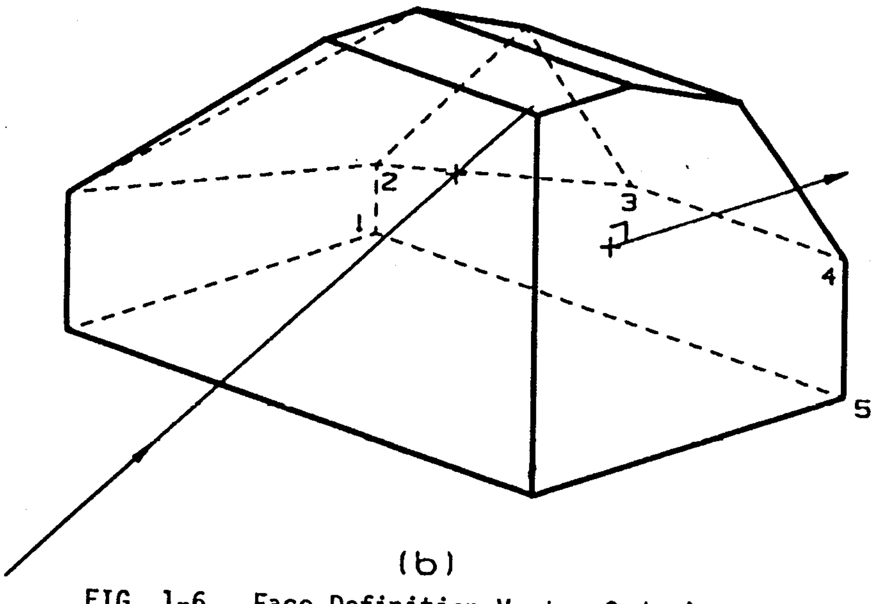
AREA CALCULATION, F10:

$$\begin{vmatrix} 179 & 3 & 35 & 179 \\ -62 & 61 & -45 & -62 \end{vmatrix} = \begin{aligned} &179(61) - 3(-62) \\ &+3(-45) - 35(61) \\ &+35(-62) - 179(-45) \end{aligned}$$

$$= 10919 + 186 - 135 - 2135 - 2170 + 8055$$



(a)



(b)

FIG. 1-6. Face Definition Vertex Ordering

All such rearward-pointing faces in any subsequent rotations can be found by computing the area enclosed by the bounding edges using the so-called determinant method [2]. This method results in a negative area for each face in which the pre-defined counter-clockwise ordering is reversed. Hidden lines are then removed by scanning the list of rearward-pointing faces and eliminating all lines which are shared by two such faces. When applied to convex polyhedral wireframe models this algorithm is sufficient to eliminate all hidden lines; for concave solids it becomes a preliminary step in a more complicated process which must further compute face/edge intersections [7].

The abovementioned geometric modeling techniques contain all information necessary for generating wireframe images on a CRT screen or incremental plotter, and are sufficient for production of pictorial engineering drawings. As a consequence of the trend toward more highly integrated design, analysis and manufacturing systems, it has become commonplace for the graphical data base itself to contain far more extensive information.

Virtually all systems provide automated generation of part dimensions, usually including several forms of radial and horizontal or vertical linear dimensions. The numerical values are extracted automatically from the model data base; the user often simply indicates the dimension required and the desired location. Extension lines and arrowed dimension lines are generated automatically. The numerical dimension is expressed in standard units with a specified precision; some systems even append explicit tolerance information. Dimension instances normally are stored with the rest of the model data as geometric entities. Most systems also include a facility for adding text strings to the object model for manufacturing notes, parts lists, etc. Like dimensions, many systems associate text with a particular view of the model. In these systems characters and dimensions are not defined in 3 dimensions, and, therefore, cannot be transformed along with graphical data to obtain rotated views.

Another natural development was the inclusion of arcs as separate entities in the model data base. Although this capability requires further software procedures to generate line segments for the display of arc entities, the almost universal occurrence of axisymmetric geometry makes the subsequent use of numerical control machine tools impossible without such model information. Clearly this single addition eliminates much of the simplicity of the previously described modeling techniques. Specification of a general arc entity in the geometric model requires at least the following pieces of information: 3-D center coordinates, radius, beginning and end angles, and finally information concerning the plane of the arc (commonly the elements of a transformation matrix). Figure 1-7 shows an adaptation of the object of figure 1-1, this time with arcs defined as part of the object model; the corresponding form of the data base is listed in table 1-4.

1.3 Entity Data Base Structure

Further elaboration of the model data base, arising primarily from the desire to produce numerical control data for automated manufacture, reflects modeling techniques quite different from those which satisfy the relatively simple requirements of wire-frame image generation. The capability of maintaining complex 3-D surface data, for example, in forms suitable both for display and for manufacturing purposes, relies primarily on the use of larger, list-oriented data bases instead of the earlier hierarchical forms. These improvements are introduced at the expense of both storage capacity and processing time; resources which are in short supply in minicomputer-based CAD systems.

Early versions, relying primarily on 16-bit processor architecture, were often designed around integer arithmetic operations because of the limited scope of the CPU instruction set. Because of the high level of interaction necessary with graphics systems, reasonable speed could be achieved only by maintaining and operating on an integer data base. Such systems were entirely

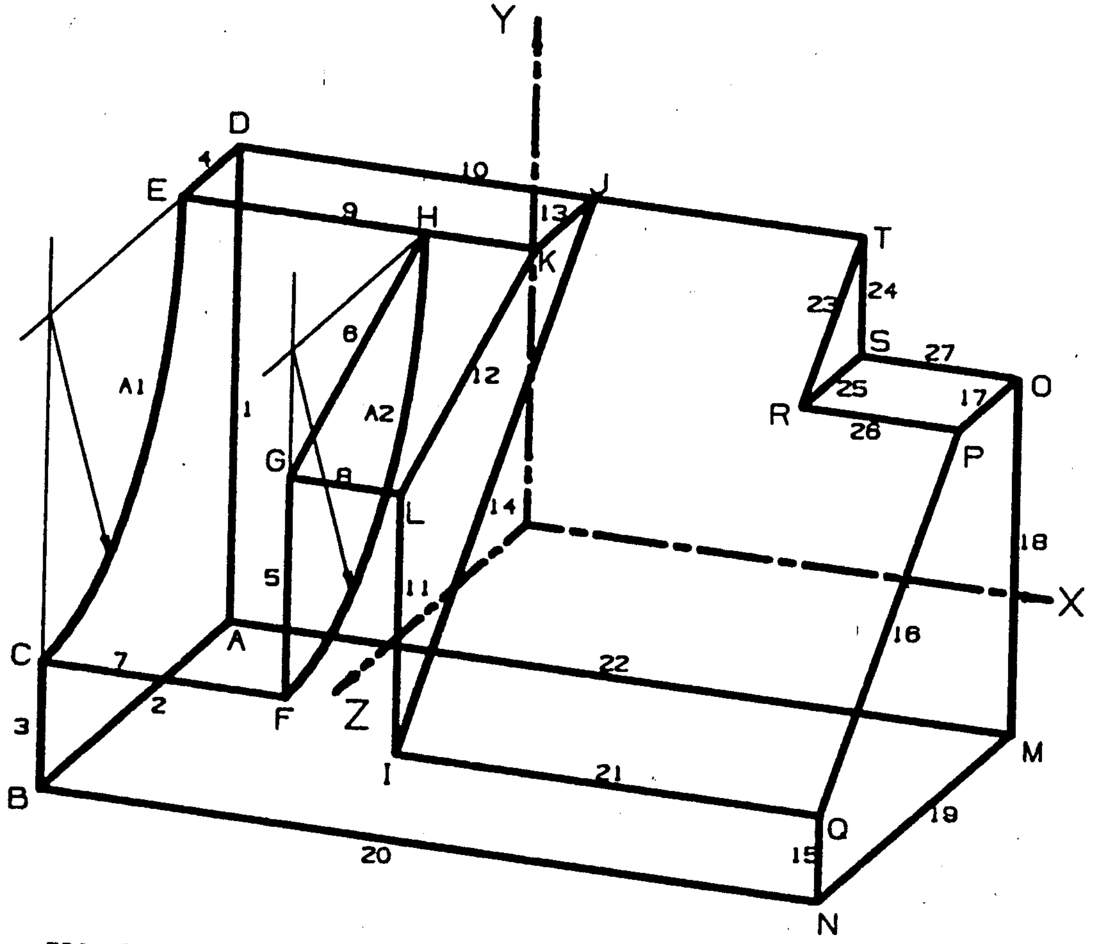


FIG. 1-7. Wireframe Model with Arcs

VERTEX	COORDINATES X	Y	Z
A	-150.	-94.	-99.
B	-150.	-94.	99.
C	-150.	-44.	99.
D	-150.	94.	-99.
E	-150.	94.	-39.
F	-56.	-44.	99.
G	-56.	43.	99.
H	-56.	94.	-39.
I	-13.	-60.	99.
J	-13.	94.	-99.
K	-13.	94.	-39.
L	-13.	43.	99.
M	150.	-94.	-99.
N	150.	-94.	99.
O	150.	47.	-99.
P	150.	47.	-38.
Q	150.	-60.	99.
R	90.	47.	-38.
S	90.	47.	-99.
T	90.	94.	-99.

LINE	VERTICES START	END	LINE	VERTICES START	END
1	D	A	15	N	Q
2	A	B	16	Q	P
3	B	C	17	P	O
4	E	D	18	O	M
5	F	G	19	M	N
6	G	H	20	B	N
7	C	F	21	I	Q
8	G	L	22	A	M
9	E	K	23	R	T
10	D	T	24	T	S
11	I	L	25	S	R
12	L	K	26	R	P
13	K	J	27	S	O
14	I	J			

	CENTER X	Y	Z	RADIUS	ANGLES		PLANE
A1	-150.	94.	99.	138.	180.	270.	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$
A2	-56.	94.	99.	138.	180.	270.	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

TABLE 1-4. Wireframe Model with Arcs

adequate for the purposes of wireframe drafting, which can be accomplished using relatively simple data structures. It is expensive and intrinsically inefficient, however, to interface these data structures, through intermediate files, to other machines running independent analysis and manufacturing software. This has led to the increasing use of 32-bit "super-minicomputers" with floating-point hardware, capable of handling a larger proportion of all 3 applications.

Accordingly, most systems now record floating-point data, some using as many as 64 bits for exponent and mantissa. In perhaps the most flexible data base scheme, each separate object model is written into a single large *library file* on mass storage. Each separate *part* likewise consists of an end-to-end sequence of variable-length records, one record being entered for each geometric entity as it is created. The general appearance of such a data file is shown in table 1-5 for the part or collection of entities in figure 1-8. Each entity type normally requires a different amount of information for its description.

One possible set of minimum formats for point, line and arc entities is shown in table 1-6. Here a point entity consists of 3 successive 64-bit entries representing its x, y and z coordinates in model space. Line entities require six 64-bit entries, giving the model space coordinates of the beginning and end

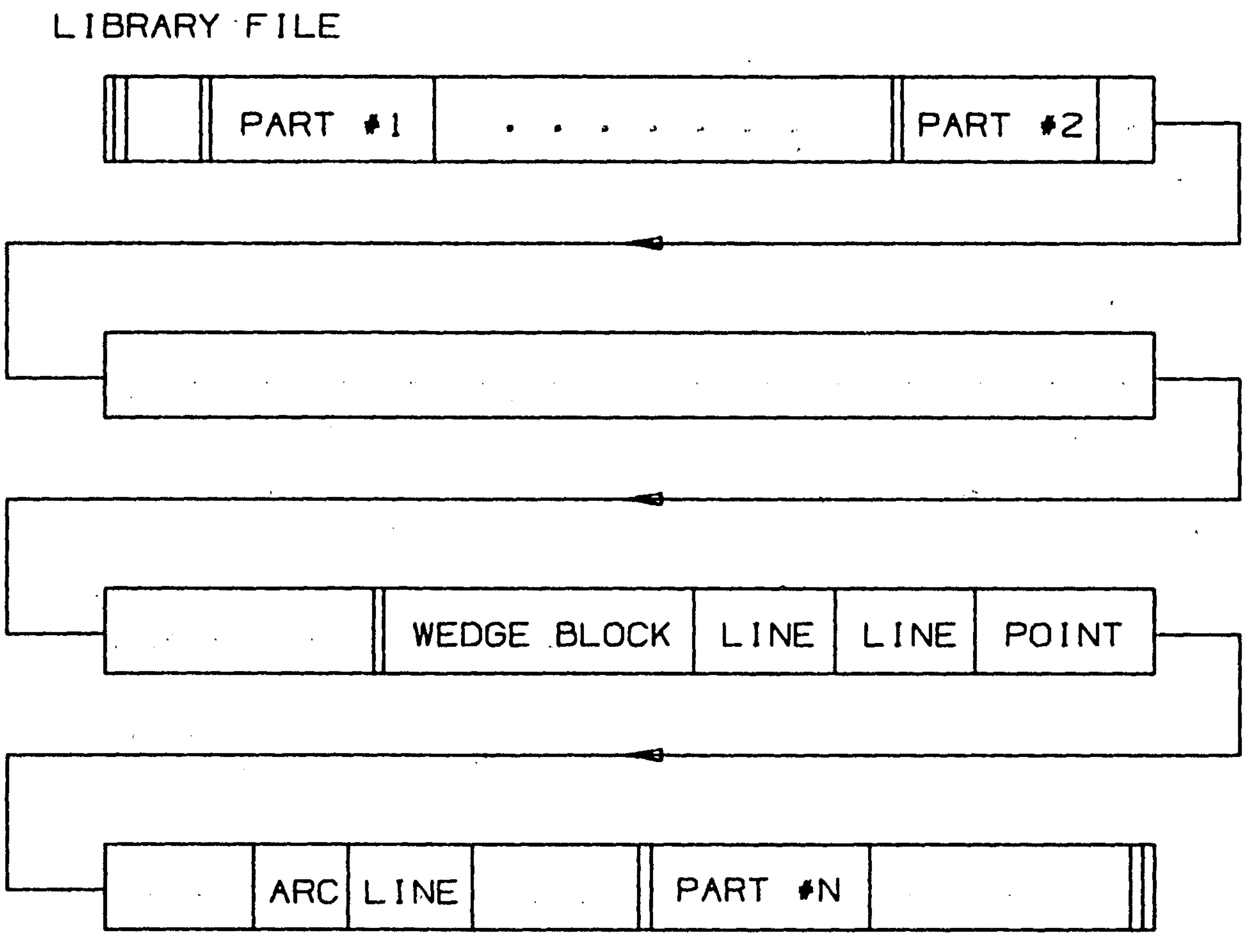
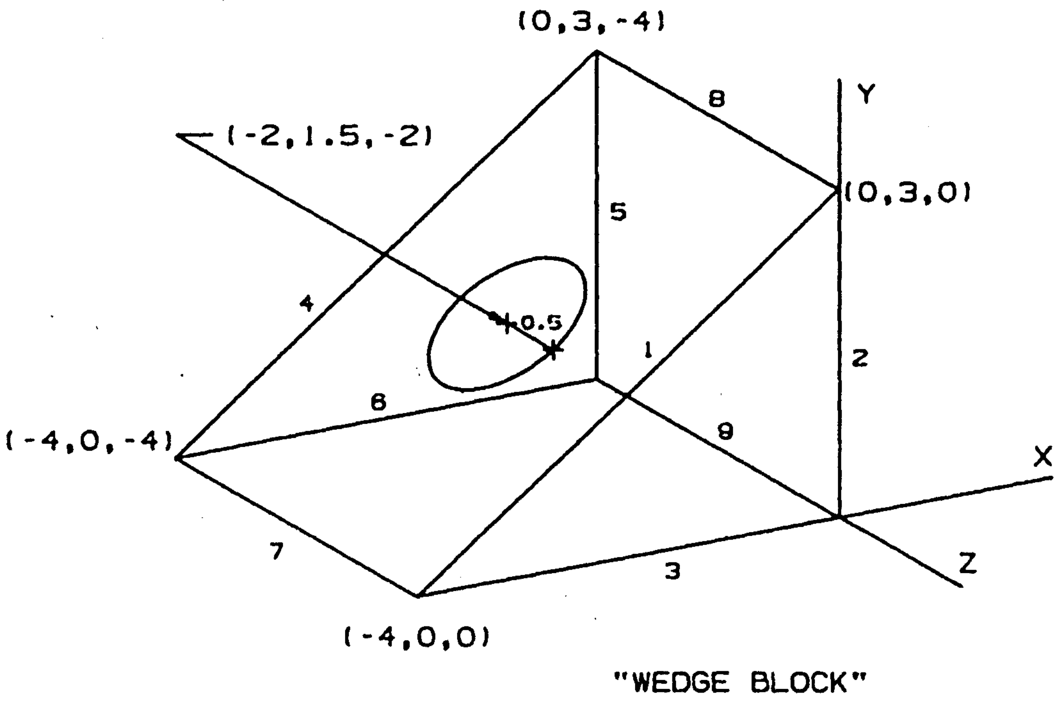


TABLE 1-5. Sample Data Base



LINES:

1	-4,0, 0;	0,3, 0	7	-4,0,-4;	-4,0, 0
2	0,0, 0;	0,3, 0	8	0,3,-4;	0,3, 0
3	0,0, 0;	-4,0, 0	9	0,0,-4;	0,0, 0
4	0,0,-4;	-4,0,-4			
5	0,3,-4;	0,0,-4			
6	-4,0,-4;	0,0,-4			

FIG. 1-8. Sample Wireframe Model

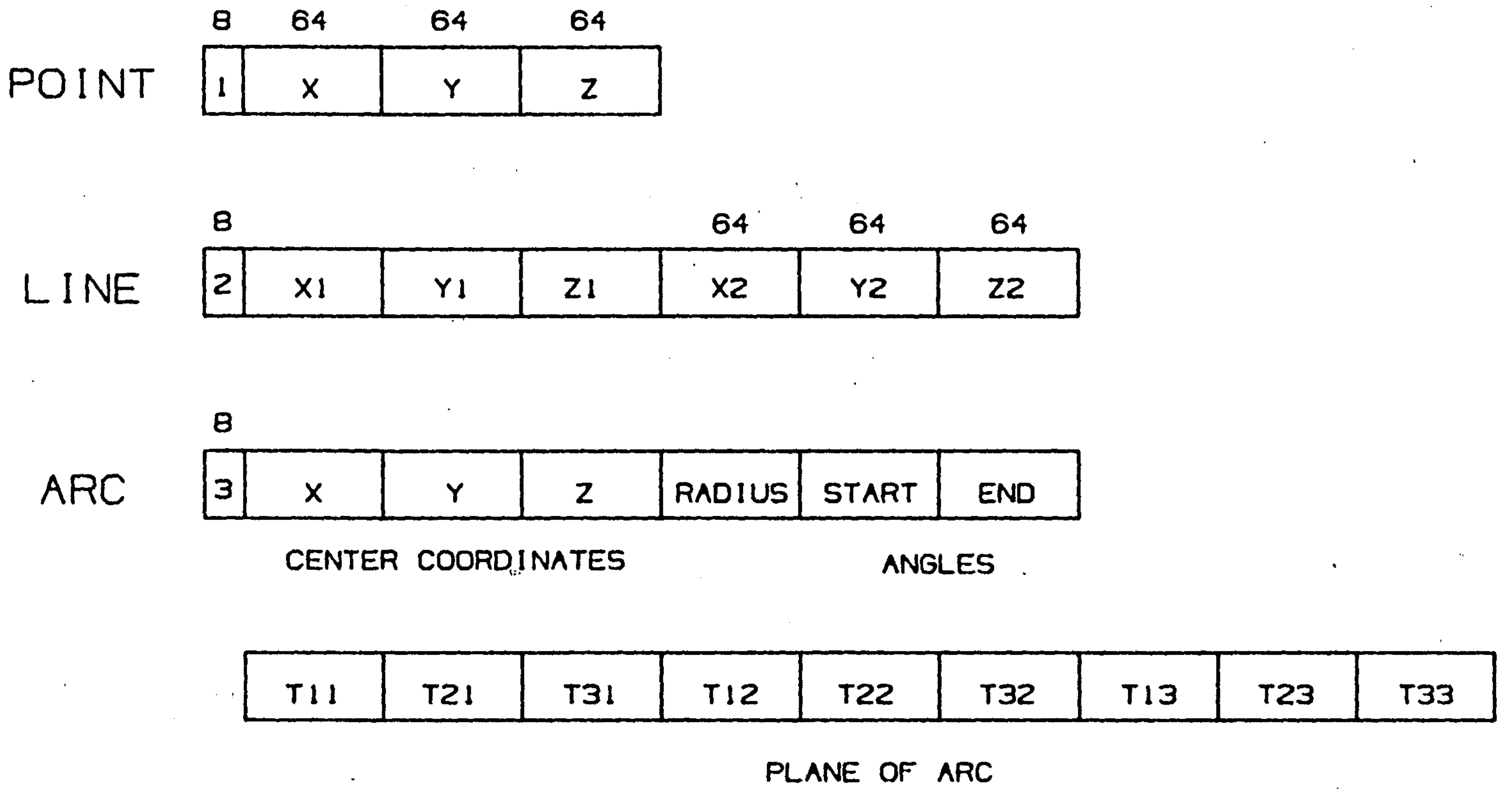
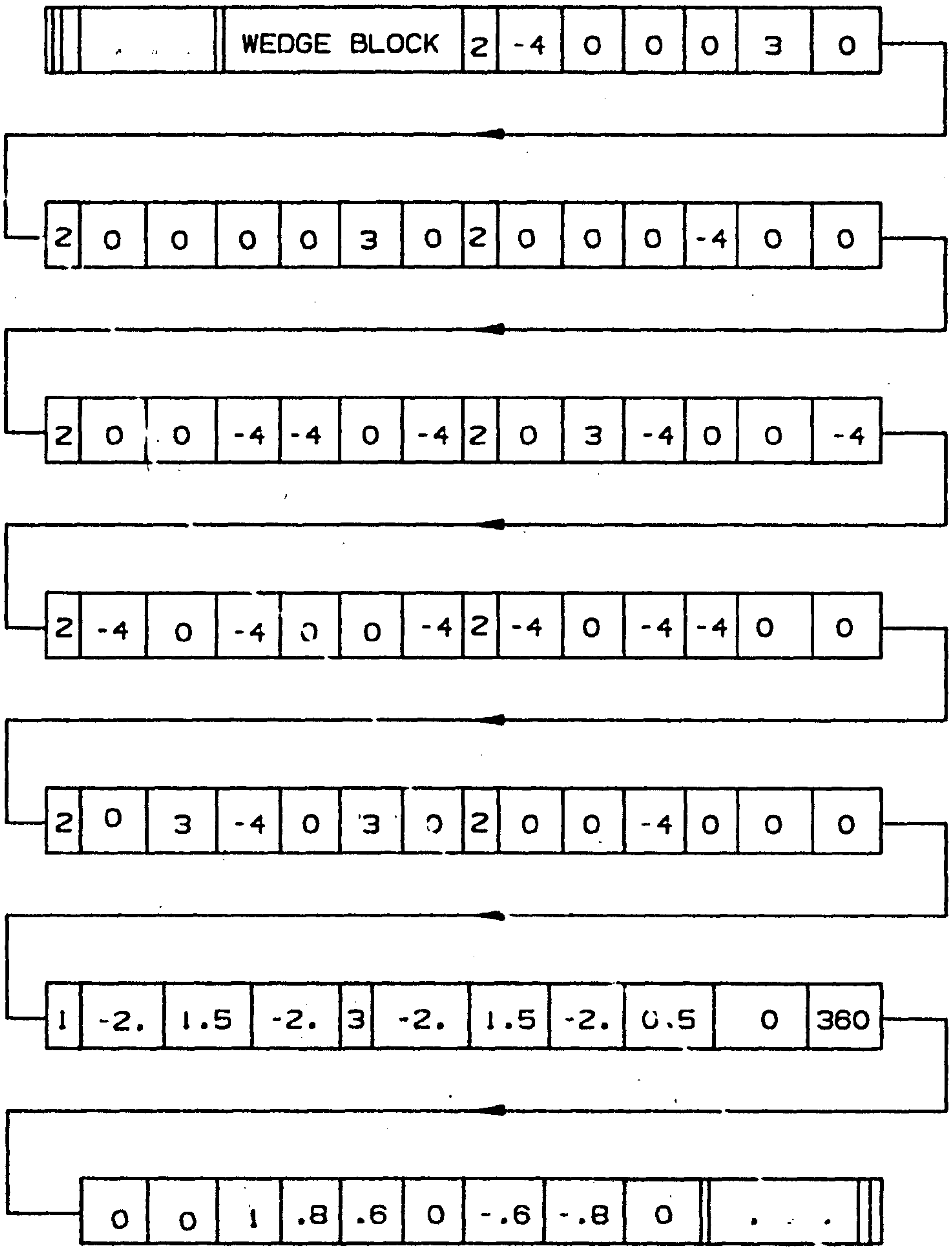


TABLE 1-6. Minimum Entity Data Formats

points of the line. Arc entities require fifteen 64-bit entries: 3 for center coordinates, 1 for radius, 2 for beginning and end angles, and the remaining 9 for the elements of a 3x3 transformation matrix defining the plane of the arc. Table 1-7 shows the actual values recorded in the data base for the object of figure 1-8. The single-byte record header gives the entity type, and consequently the number and manner in which the following bytes are to be interpreted. Most systems also include a block of integer data within each entity record specifying various status information, such as an entity number, display mode (solid, dashed), pointers to other entities, etc.

The most significant differences between current systems involve the methods by which complex entities, e.g., splines, analytic curves, planar and curved surfaces, etc., are described. It is from these entries that much of the information for numerical analysis and manufacturing is derived; hence, it can be expected that modeling techniques in this area will be continually refined as design, analysis and manufacturing become more widely integrated.



. . .	WEDGE BLOCK	2	-4	0	0	0	3	0					
2	0	0	0	0	3	0	2	0	0	0	-4	0	0
2	0	0	-4	-4	0	-4	2	0	3	-4	0	0	-4
2	-4	0	-4	0	0	-4	2	-4	0	-4	-4	0	0
2	0	3	-4	0	3	0	2	0	0	-4	0	0	0
1	-2.	1.5	-2.	3	-2.	1.5	-2.	0.5	0	360			
0	0	1	.8	.6	0	-.6	-.8	0	. . .				

TABLE 1-7. Sample Model File

2.0 Wireframe Graphics System Structure

Current wireframe graphics-systems are alike in that nearly all are supervised by dedicated minicomputer systems. The basic operation of these systems consequently is fixed by the common constraints imposed by minicomputer system architecture and the intrinsic computational demands of wireframe graphics.

2.1 Principal Hardware/Software Components

The diagram of figure 2-1 shows the typical structure of a multi-user wireframe graphics system. The two blocks at the top of the diagram, the graphics procedures library, and the model data base, can be shared by all users of the system; the blocks below are duplicated for each individual user or workstation. In this scheme the graphics procedures library is the set of all software procedures (or subroutines) in executable form responsible for carrying out each of the graphics functions implemented by the system. The data base resides on mass storage devices, commonly high-capacity hard-disk drives, often in both permanent and temporary forms; i.e., each workstation normally writes to its own scratch area, editing not the original but a copy of the permanently filed part.

The remaining software, which is duplicated for each user, consists primarily of a real-time I/O monitor loop which

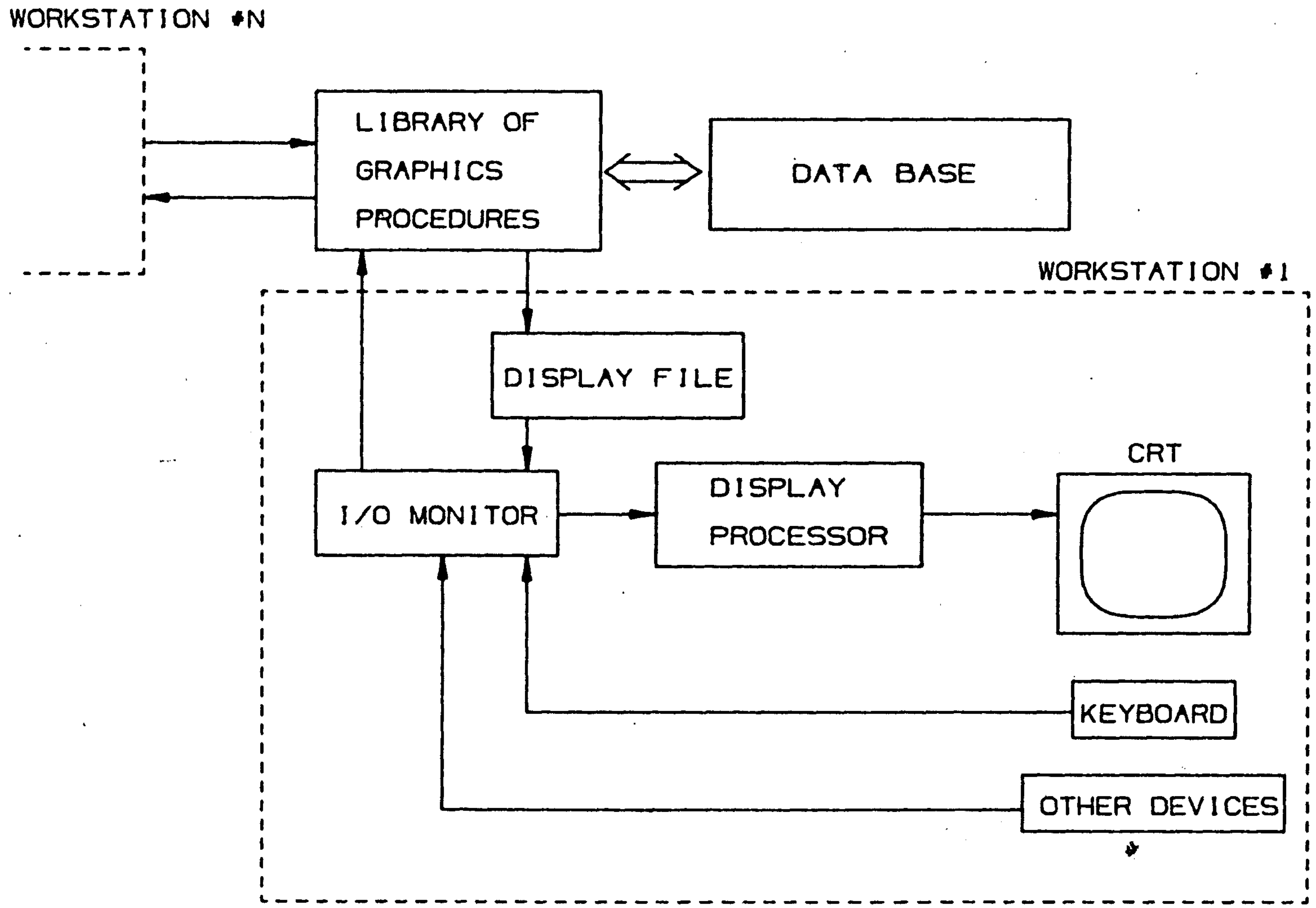


FIG. 2-1. System Structure

periodically polls or queues user I/O events. Other user-specific components are mostly various pieces of display or interaction hardware, such as keyboards, CRT's, function buttons, etc. In these systems user commands typically are issued either as alphanumeric strings entered using a standard keyboard, or, more effectively, using pre-defined (and sometimes user-defined) function buttons. The monitor program accepts and evaluates the various forms of input data, and matches user requests to executable procedures maintained in the system library. Control is then transferred to the selected procedure, which, using parameters passed to it by the monitor program, operates on or extends the current data base as necessary. Results are communicated to the user either as messages indicating current system status or as changes in the displayed image.

2.2 Display Devices

Details of the conversion from model data to displayed image are highly machine-dependent, particularly with respect to the type of CRT being driven. There are three significantly different technologies in current use: storage tube, vector refresh and raster scan. All three find application in engineering graphics, and each excels in certain specific areas.

Storage tube displays are generally the most economical, both in hardware cost and in software requirements. They typically communicate with the host computer via standard serial interfaces, and have traditionally afforded the highest screen resolution (4096x4096 is not uncommon). Lines are produced on the screen by issuing sequences of character codes representing the endpoint screen coordinates; hardware local to the display (referred to as the display processor) interprets the data and activates the deflection circuitry of the CRT. Since all lines are retained on the screen indefinitely, an image may be "painted" line-by-line at any arbitrary speed. Storage tube CRT's are normally monochrome, although beam-penetration models with up to four different-colored phosphor layers are available.

Vector refresh CRT's are similar to storage tube models except for the obvious absence of the storage feature. Stable images are obtained through rapid retracing of the entire display (approximately 30 times per second is common). High persistence phosphors are often used to reduce the apparent flicker of complex images.

Raster scan CRT's used in computer graphics are enhanced versions of commercial television monitors. In these displays, deflection circuitry does not trace the image directly on the screen face, but rather generates a fixed pattern of horizontal scan lines. Dots are produced at precise locations along each scan line by pulsing the acceleration potential of the electron beam. As in vector refresh displays, the data for any particular image must be stored by the display hardware as long as the image is to be displayed.

The three types of displays above fall into two classes so far as engineering graphics applications are concerned: static displays, and dynamic displays. The storage tube terminal is most appropriate for line drawing tasks requiring a relatively low interaction level, such as in defining model geometry or in plotting graphs. Here higher screen resolution produces a smoother, more accurate rendering of curves. These terminals also can be located remotely from the host computer since they use standard serial communication methods. The vector refresh display, in contrast, is inherently dynamic, requiring a constant stream of image data in order to produce any coherent image at all. As a natural consequence, a smooth motion of the object can be simulated by recalculating this image data in real time. Motion of an image helps the designer visualize complex 3-D objects, and helps the engineering analyst visualize deflections due to applied loads or free or forced vibration, for example.

Until recently vector refresh terminals had been used almost exclusively for dynamic display applications. Sharply decreasing hardware costs, however, particularly in random-access memory packages are making raster scan displays economically competitive. Further, they afford the advantage of color. While vector refresh displays require that the complete set of graphics instructions and data, called the *display file*, be stored for reference, raster scan displays store numerically the state of each addressable screen dot, *or pixel*[5]. For monochrome tubes the value stored corresponds to a gray scale intensity level; for color tubes the value is a number corresponding to intensity levels for each of the three primaries red, green and blue. Implementing a 512x512 matrix of pixels in a 16-color or 16-intensity system requires a vast amount of image memory: 512·512·4 bits of data per pixel = 128K (131,072) 8-bit bytes. The display file for vector refresh displays, in contrast, is typically no more than several thousand bytes in length.

The relatively low screen resolution of raster scan displays is typified by the "staircase effect" in the representation of an angled line. While storage tube and vector refresh displays, referred to as calligraphic or stroke displays, generate a straightline between any two discretely addressable screen coordinates using analog methods, raster displays illuminate only a string of dots, each being one of the addressable screen coordinates. Figure 2-2 shows a raster image of a fairly complex object, produced on a dot matrix printer with a dot spacing of 0.033 inch, comparable to a 512x512 12-inch diagonal CRT. Figure 2-3 shows the same object as reproduced on a high-resolution drum plotter having a step size of 0.0004 inch. The staircase effect is of little practical consequence in systems using such a plotter, since designers working on raster CRT's need only interact with images good enough to verify the integrity of the model data base; high quality engineering drawings can be generated independently if needed at a later time.

2.3 Device-Independent Systems

The wide variations in display capability and mode of operation have resulted inevitably in graphics systems suitable only for certain kinds of applications. This high level of particular device dependence also constitutes one of the major obstacles to the user's understanding of the underlying principles of computer graphics. Both problems have served to hinder the progress of integration of design, analysis and manufacture.

The diagram of figure 2-1 incorporates steps to ensure device-independent system structure. A two-step translation process is assumed between the procedures library and the display screen,

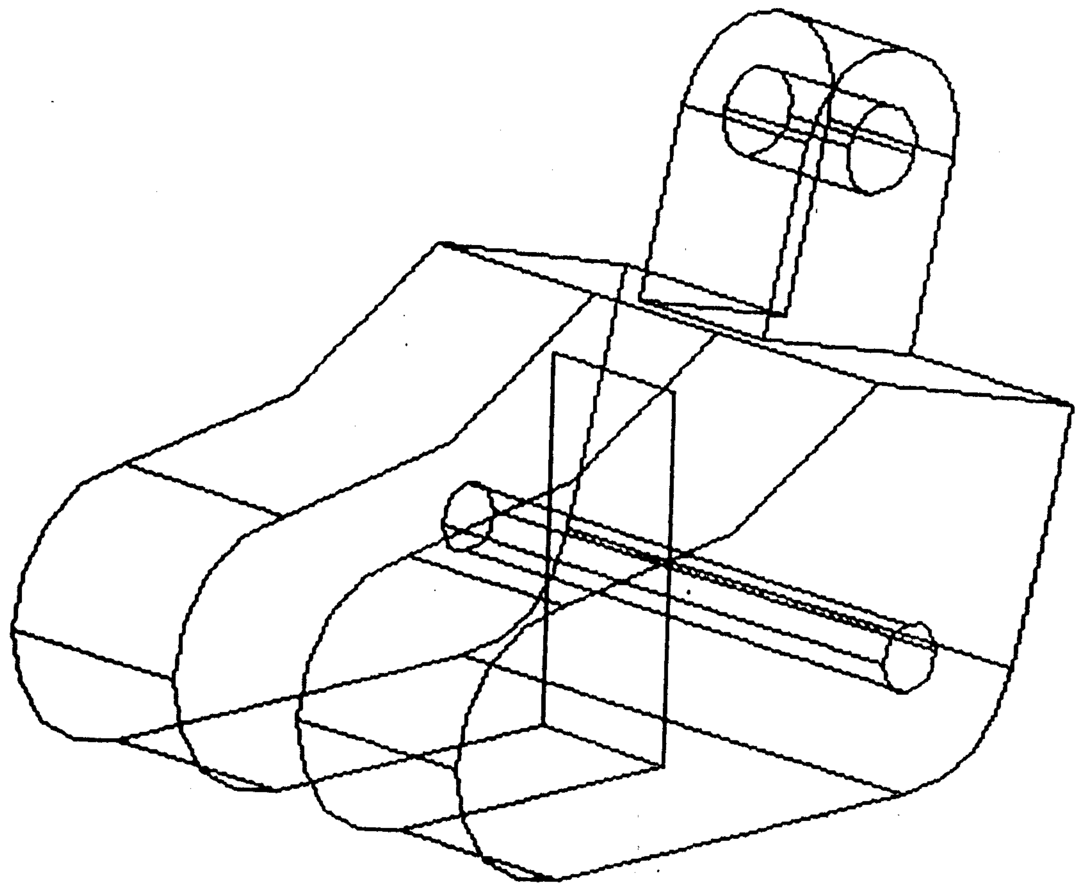


FIG. 2-2. Wireframe Model Raster Image

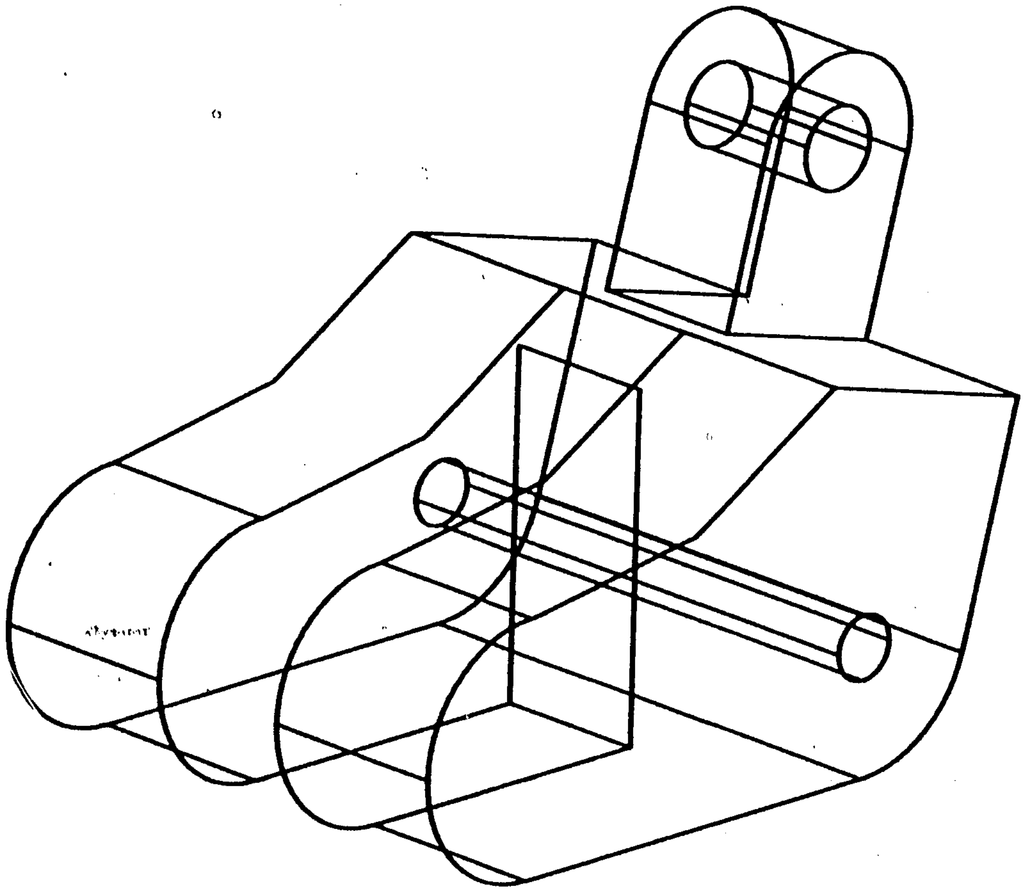


FIG. 2-3. Wireframe Model Calligraphic Image

namely the display file and the display processor. This extended structure is sufficient to implement any and all of the three previously mentioned types of CRT technologies, though it may require redundant processing, especially with storage tube displays. A separate display file procedure can be included in the library for each different type of display, all sharing the same model data base. The proper procedure is then invoked automatically by the system, transparent to the user. The resulting display file is either transmitted serially to the display device, or may simply be made available to the display processor in central memory for independent, high-speed access.

The actual translation process which occurs between model data base and displayed image thus consists of the assembly, in the host computer, of a sequence of commands and data representing the desired image in a form recognizable by the particular device's display processor. This implies a dual representation of the object to be displayed; the data base contains a generalized 3-D geometric model of the object, while the display file is an instruction code sequence representing a particular 2-D projection of the object.

To illustrate, assume a calligraphic CRT having a display processor which implements in hardware the following limited set of graphics primitive operations:

MOVA	xx,yy	move absolute
DRWA	xx,yy	draw absolute
MOVR	xx,yy	move relative
DRWR	xx,yy	draw relative

Here the "move" operation causes an invisible change in the position of the electron beam on the CRT face. The "draw" operation causes the beam position to move in a straight line from the current position to the specified one, leaving a visible trace on the screen. In these instructions, for the absolute form, xx and yy (positive integers) give the screen coordinates of the new position. In the relative form, xx and yy (positive or negative integers) commonly indicate changes from the current position. The display file for this processor giving the image shown in figure 2-4a would thus appear as listed in table 2-1. The path of the electron beam in tracing this image is shown in figure 2-4b.

The graphics system structure as presented in this chapter is typical of the more powerful 3-D wireframe systems currently available, and is most readily adapted to different displays and expanded to wider applications. In the next chapter the content of the procedures library will be examined in greater detail, particularly those graphical functions most frequently applied in engineering design.

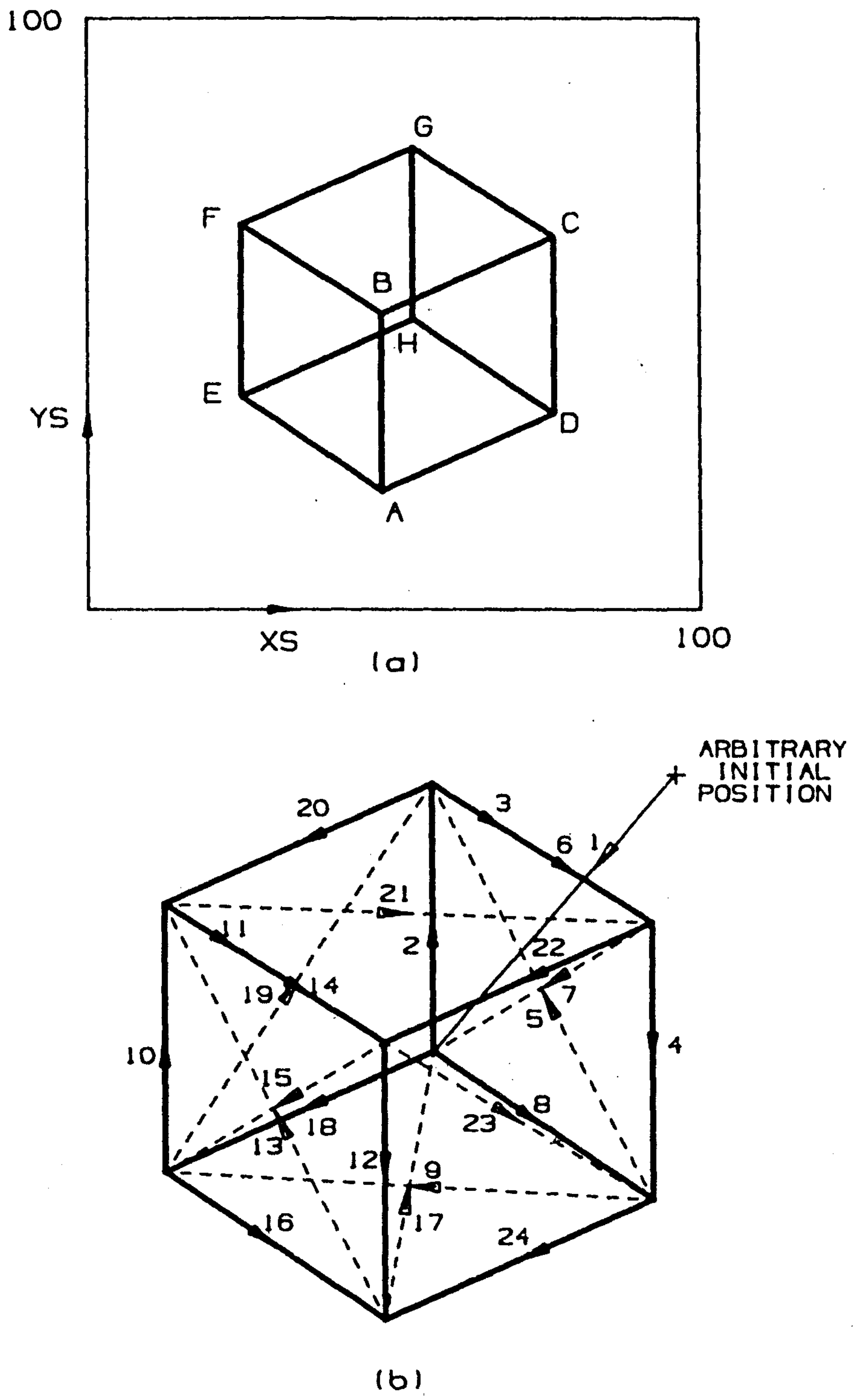


FIG. 2-4. Sample Calligraphic Image

ENDPOINTS		LINES
A	48,20	A,B
B	48,50	C,D
C	76,63	B,C
D	76,33	A,D
E	25,36	E,F
F	25,65	G,H
G	53,78	F,G
H	53,49	E,H
		A,E
		B,F
		C,G
		D,H

DISPLAY FILE INSTRUCTIONS					
1	MOVA	48,20	13	MOVA	25,65
2	DRWA	48,50	14	DRWA	53,78
3	MOVA	76,63	15	MOVA	25,36
4	DRWA	76,33	16	DRWA	53,49
5	MOVA	48,50	17	MOVA	48,20
6	DRWA	76,63	18	DRWA	25,36
7	MOVA	48,20	19	MOVA	48,50
8	DRWA	76,33	20	DRWA	25,65
9	MOVA	25,36	21	MOVA	76,63
10	DRWA	25,65	22	DRWA	53,78
11	MOVA	53,78	23	MOVA	76,33
12	DRWA	53,49	24	DRWA	53,49

TABLE 2-1. Sample Display File

3.0 Wireframe Image Generation

Early engineering graphics systems were introduced primarily to supersede manual drafting. As a consequence many began as 2-D systems, the model data base tailored specifically to the efficient generation of single view drawings. The desire to produce sets of related orthographic views, and later the integration of drafting with design, analysis and manufacturing led to more elaborate 3-D modeling techniques, similar to those developed in Chapter 1. This trend has necessarily generalized the procedures required for assembling a particular display file from the model data base.

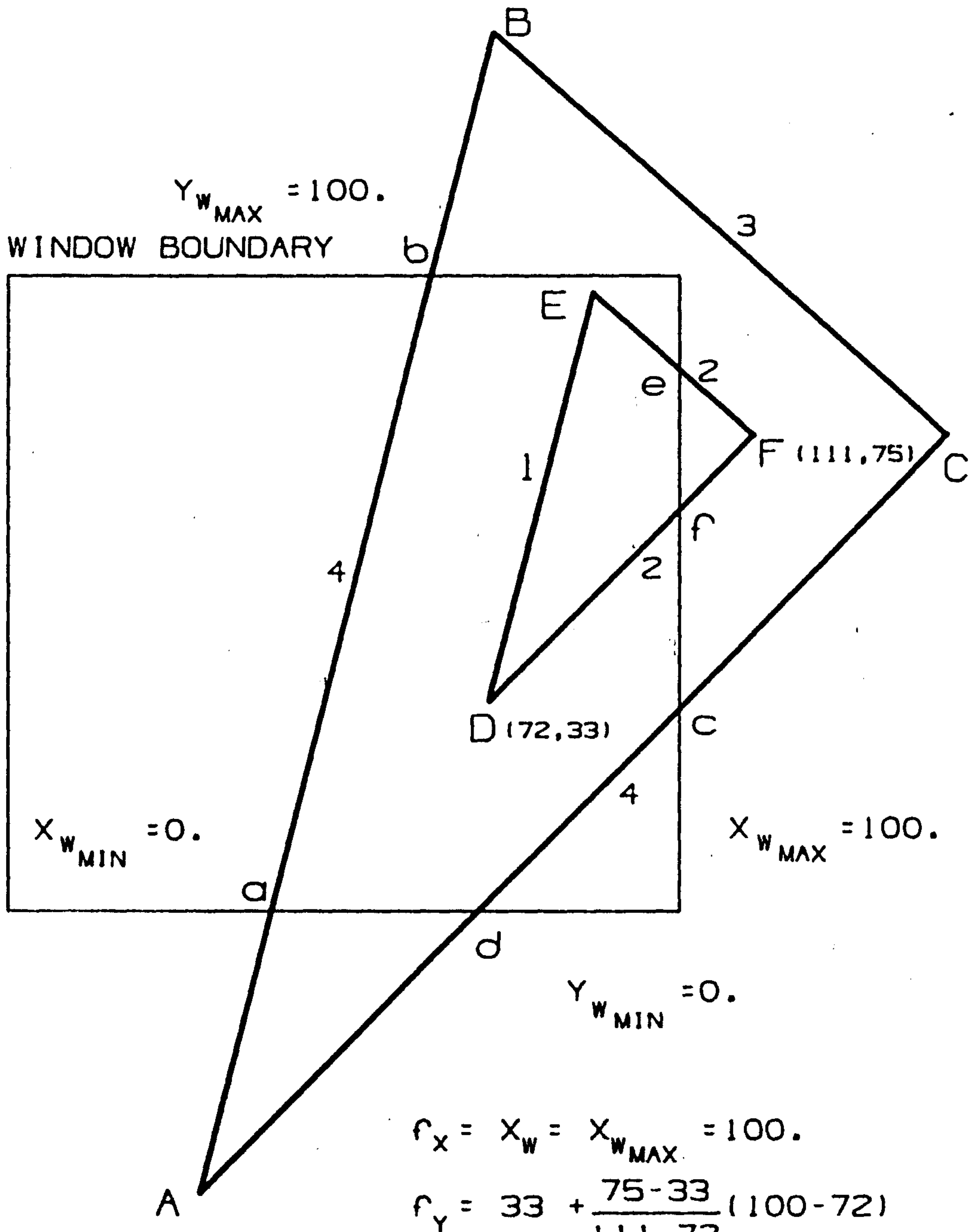
The function of these basic graphics procedures is essentially that of mapping a 3-D model coordinate system into the 2-D screen coordinate system of the display device. Two of these procedures, referred to as clipping and projection, determine how much of the object appears in the screen image, and in what orientation, respectively. These functions have been handled in a variety of ways in currently available wireframe graphics systems, and are possibly the greatest source of confusion to new users. The methods used in any particular system also dictate to a large degree the form of man/machine interaction best suited to that system, and are often a significant factor in the choice between competing vendors.

3.1 Clipping

The clipping procedure is one of the most basic graphics operations. In order to generate a close-up or "zoomed" image of an object, all geometric entities which would consequently lie off the screen must be actively suppressed from the corresponding display file for that image. Figure 3-1 illustrates for the 2-dimensional case a typical situation to be resolved by the clipping procedure. As shown in the figure the boundary of the "visible" portion of the data base, referred to as a *window*, is almost universally rectangular, often square. This is not an essential restriction, but greatly simplifies the testing procedure. Point entities afford the least difficulty; their coordinates are compared with those of the four lines which define the window, and the point is declared either in or out according to the results of the test. Line entities, however, are much more complex[4]. There are four possible cases encountered in the clipping of lines, classified by number in figure 3-1:

CASE 1. IN-IN - line is completely within window; entire line displayed

2. IN-OUT - line is partially within window; segments Ee and Df displayed, segments eF and fF not displayed



$$f_X = X_W = X_{W_{MAX}} = 100.$$

$$f_Y = 33 + \frac{75-33}{111-72}(100-72)$$

$$= 33 + 1.077(28) = 63.2$$

$$f = (100, 63.2)$$

FIG. 3-1. 2-D Clipping

3. OUT-OUT - line is entirely outside window, not displayed

4. OUT-IN-OUT - line is partially within window; segments ab and dc displayed, segments Aa, bB, Ad and cC not displayed.

Like many other higher-level graphics procedures, clipping is essentially a sorting operation. It not only reduces the whole collection of lines in the object to a particular subset, but moreover it modifies existing lines by creating new endpoints (those labeled with lowercase letters in the figure). The new endpoints can be determined by solving the original line equations, derived from the endpoint coordinates (x_i,y_i) and (x_j,y_j) stored in the model data base, for an intersection (x_k,y_k) with the given window boundary.

At left or right side: $$x_k = x_w \qquad 3.1$$

$$y_k = y_i + \frac{y_j-y_i}{x_j-x_j}(x_w-x_i) \qquad 3.2$$

At top or bottom: $$x_k = x_i + \frac{x_j-x_i}{y_j-y_i}(y_w-y_i) \qquad 3.3$$

$$y_k = y_w \qquad 3.4$$

The calculation using equations 3.1 and 3.2 above for the new endpoint f of line DF is given in figure 3-1; the clipped screen image described in the resulting display file is graphically represented in figure 3-2. Window specification normally is made by giving center coordinates and either a single scale factor for square boundaries, or two linear dimensions for rectangular boundaries.

Computational requirements incurred by a general floating-point clipping procedure alone can clearly become quite substantial for complex objects, particularly since the intersection equations involve both multiplication and division. Lines must also be classified among the four possible cases stated above using the only available data, the coordinates of the two end-points in model space. In many cases, it is not possible to determine immediately from this information alone which window border is violated. Algorithms for this procedure are commonly iterative or recursive, either by explicitly solving for boundary intersections in x and y repeatedly until both coordinates are on or inside the window, or by searching logarithmically for a point on the line which falls just inside the window according to some fixed tolerance. Processing time for clipping purposes alone often is sufficient to preclude dynamic image generation on minicomputer-based systems.

The 2-dimensional example presented above is extended by analogy to the general 3-D case. The visible region of the model

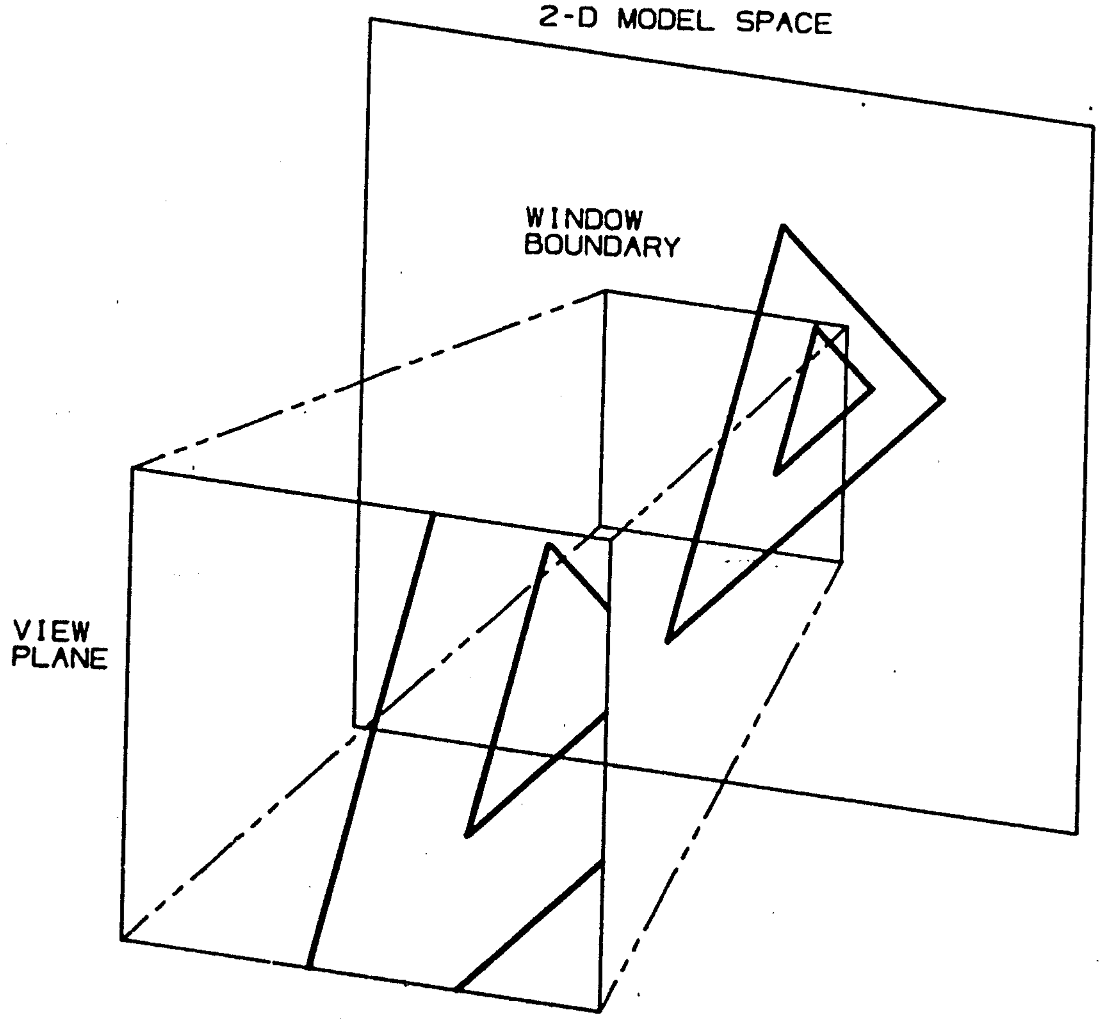


FIG. 3-2. Clipped Screen Image

data base subsequently becomes a 3-D volume, usually cubic, referred to as the view volume or view cube. The clipping procedure must further be concerned with intersections at the "front" and "back" of this region. Two different approaches to the 3-D clipping problem have been taken in current wireframe systems, producing images significantly different in content. Those current 3-D systems which originally evolved from more primitive 2-D versions often are betrayed by their particular use of clipping in the view direction.

The first method simply applies a 2-D clipping procedure identical to that detailed above to all entities in the view direction. The front and back of the view volume are thus fixed at the bounds of model space. Figure 3-3 is a graphical representation of such a system. Here the plane triangular object of figure 3-1 has been extended non-uniformly in the depth direction, completing a 3-D model. The 2-D screen image generated from this model is shown in the figure in front of and along the direction of view of the 3-D space containing the object model. The orientation of the rectangular view volume is indicated in dashed lines, enclosing as described the full extent of model space in the view direction.

An image produced from the same model using an explicit 3-D clipping algorithm is depicted for comparison in figure 3-4.

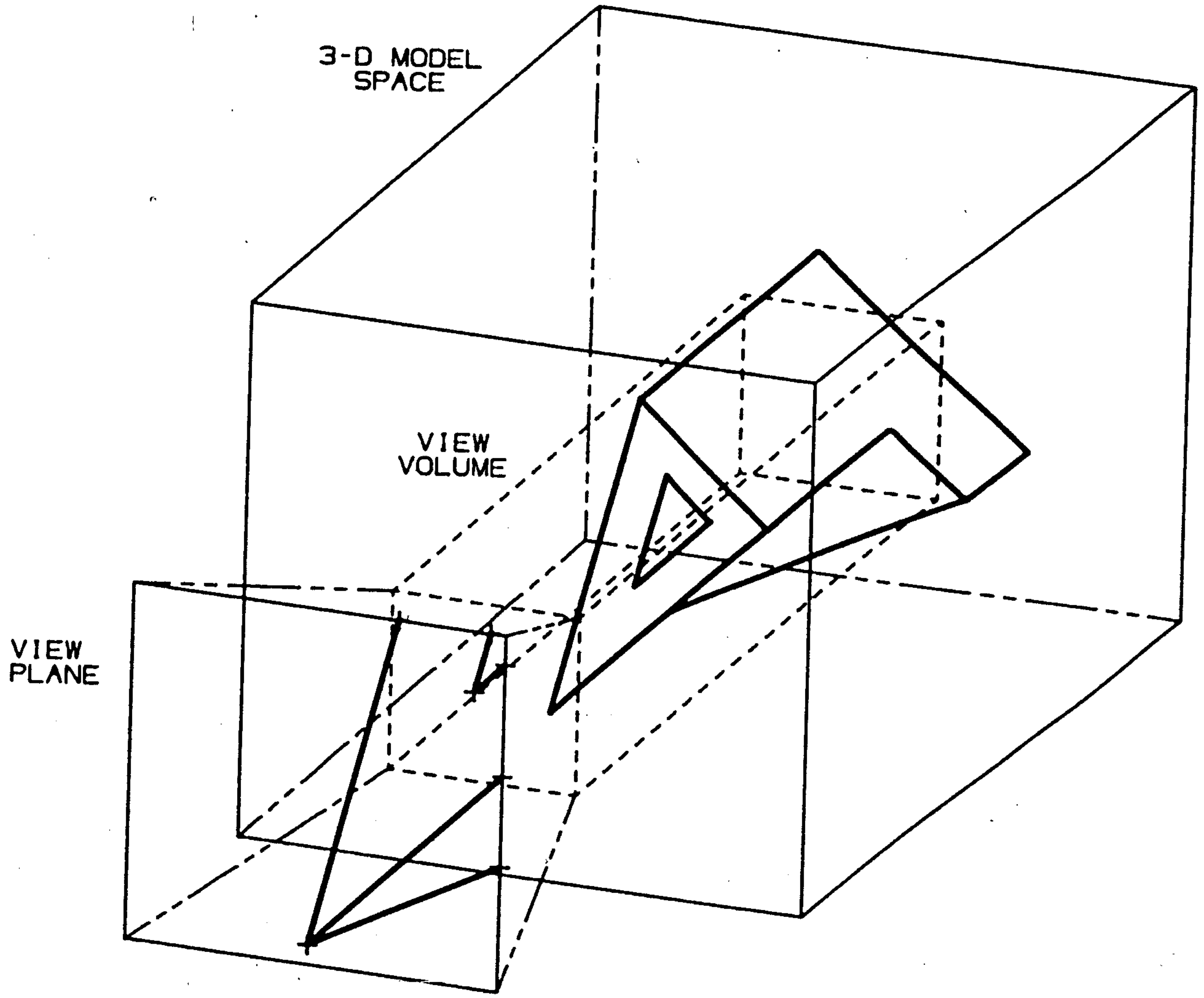


FIG. 3-3. Clipping in 3-D, Method 1

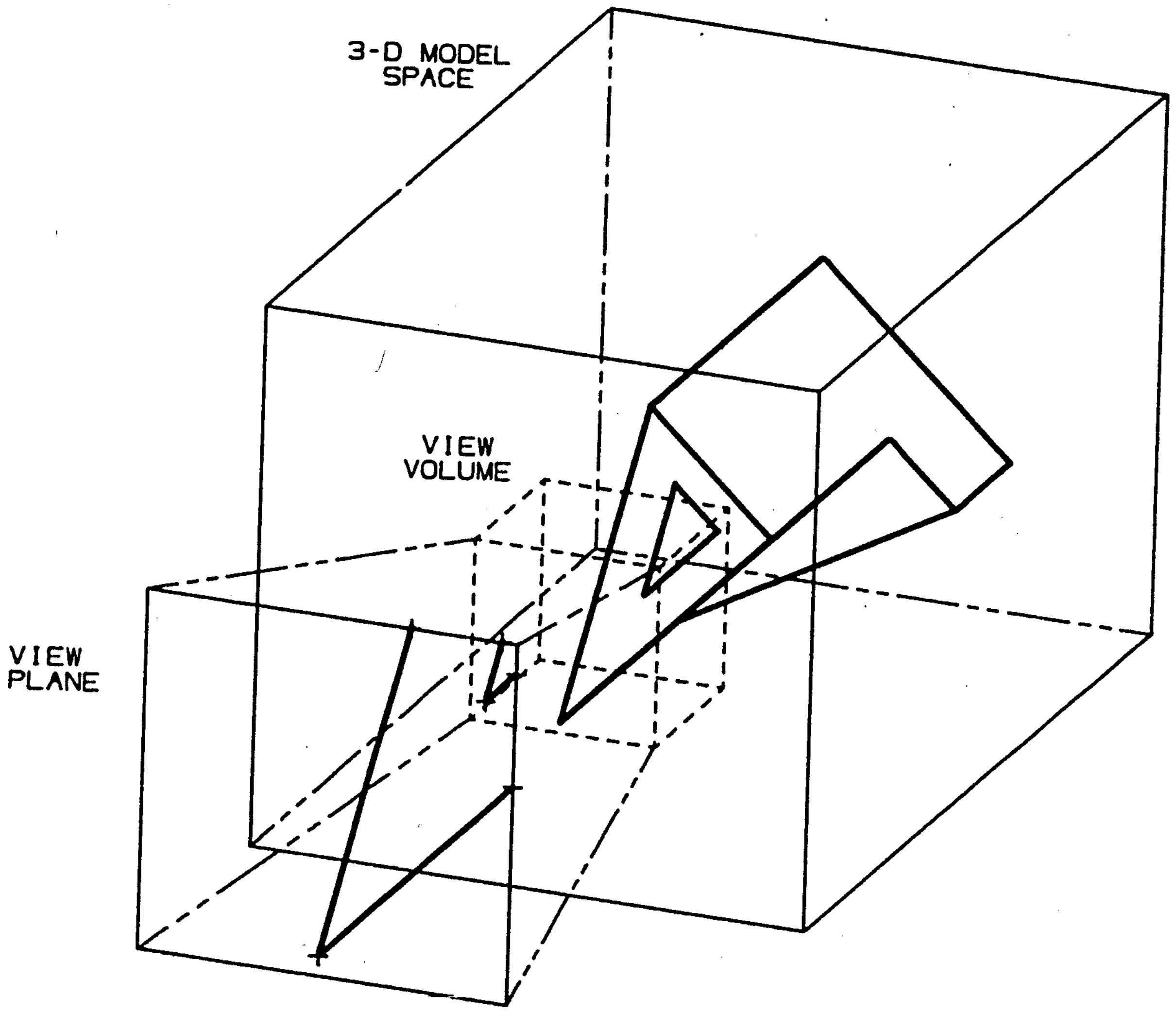


FIG. 3-4. Clipping in 3-D, Method 2

In this case the view volume is a cube, commonly specified in current systems by its width and center coordinates in model space. The screen image showing the contents of this view volume differs from that of figure 3-3 in the omission of a line representing an edge near the back of the object. This detail is suppressed by the clipping procedure because it lies outside the view cube. The scales of the two screen images are identical. Details of the back of the object are revealed in the latter screen image only if the location or width of the view cube is modified to enclose these entities. Figure 3-5 shows the results of relocating and enlarging the view cube; the change in shape near the back of the part is now included in the screen image, and the entire object appears in a smaller scale.

One side-effect of the implementation of clipping is the ability to become "lost" in model space; i.e., the possibility of specifying a window in 2-D, or view volume in 3-D, which encloses no part of the modeled object. The probability of such an occurrence is aggravated by 3-D clipping procedures; figure 3-6 shows such a situation. More effective systems guard against such a frustration. The distinction between graphics systems utilizing 2-D or 3-D clipping is important in engineering applications, particularly regarding the generation of consistent sets of orthographic views, and will be examined more fully in the next section.

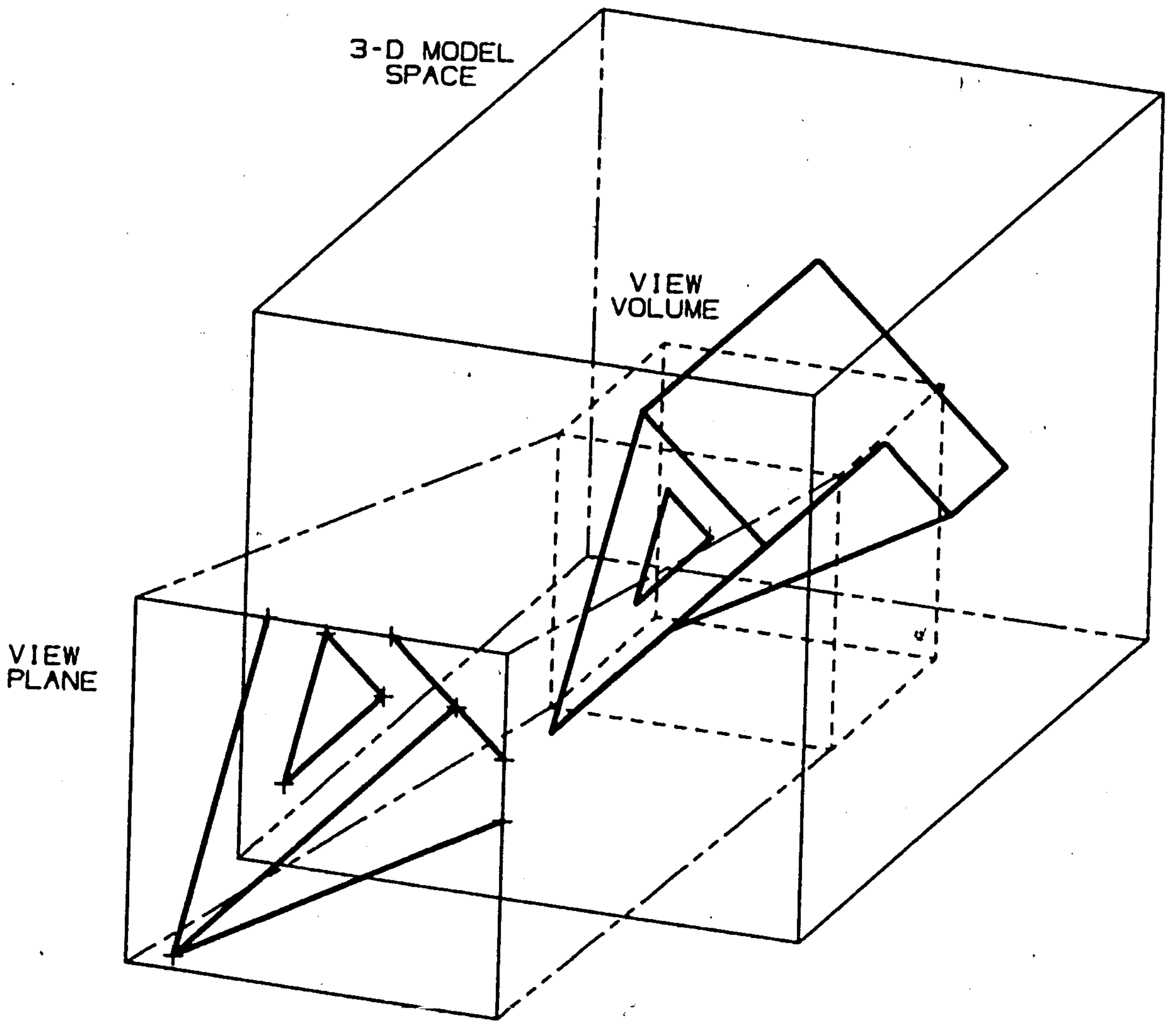


FIG. 3-5. Enlarged View Cube

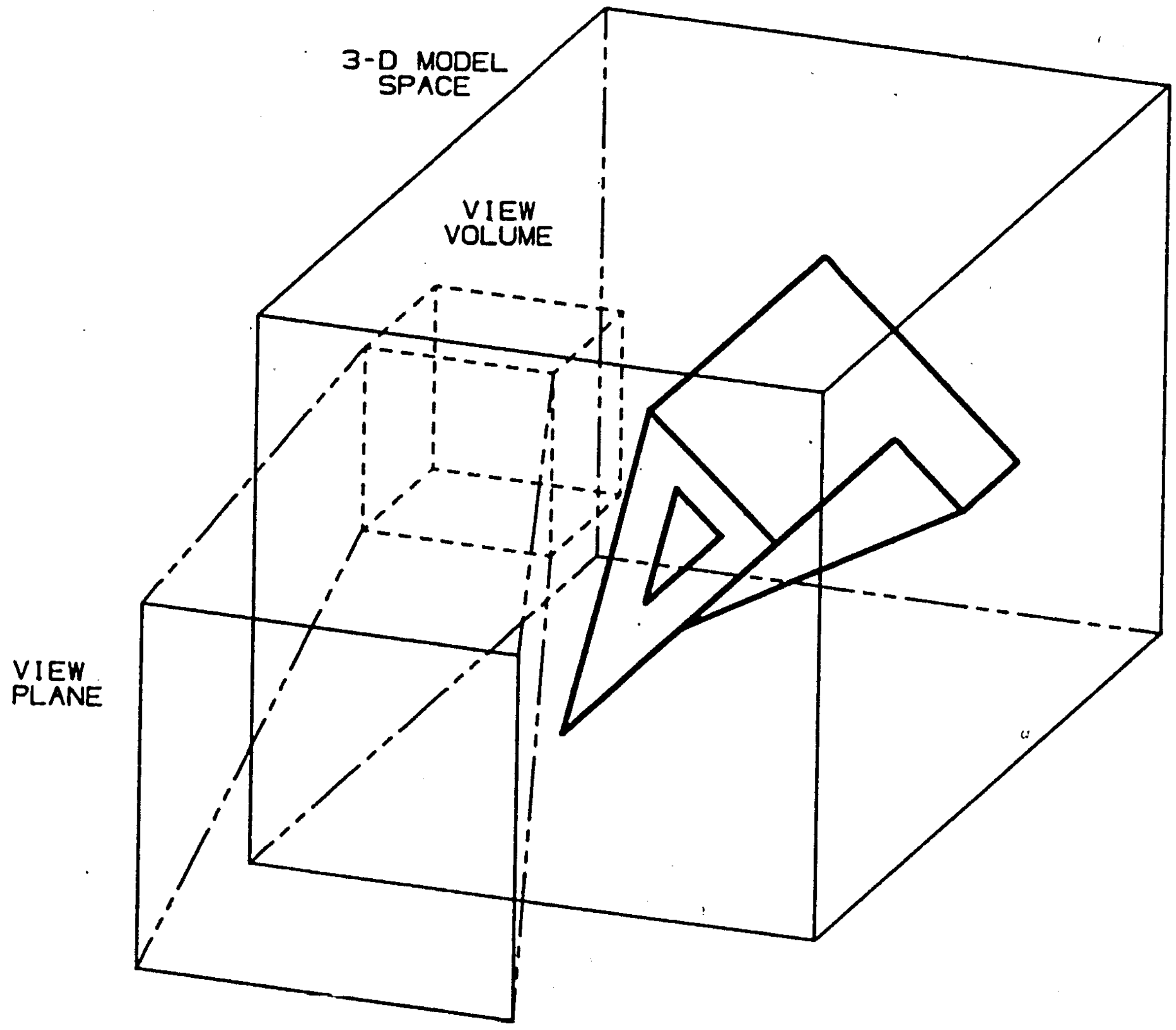


FIG. 3-6. Clipping Side-effect

3.2 Projection

The other basic wireframe graphics procedure, responsible for collapsing the 3-D model data onto the 2-D screen or view plane, is referred to as projection. Much of the power of graphics systems lies in their ability to generate an image from a 3-D model showing the object in any possible physical orientation. This ability is founded upon the methods of matrix coordinate transformations. There are two principal forms in use: orthographic or parallel projection, and perspective or central projection.

3.21 Orthographic Projection

The process of orthographic projection has been tacitly employed in all of the preceding figures and examples. Up to this point the x and y model space axes have been presumed parallel to the view plane x and y axes, with the viewer located infinitely far from the view plane. This situation is represented graphically in figure 3-7. In this case, device x and y coordinates for all entities to be displayed are obtained, after appropriate scaling and type conversion, directly from the existing model x and y coordinates. Graphically, this implies that projection lines extend from the object vertices to the view plane parallel to the line of sight. For other arbitrary

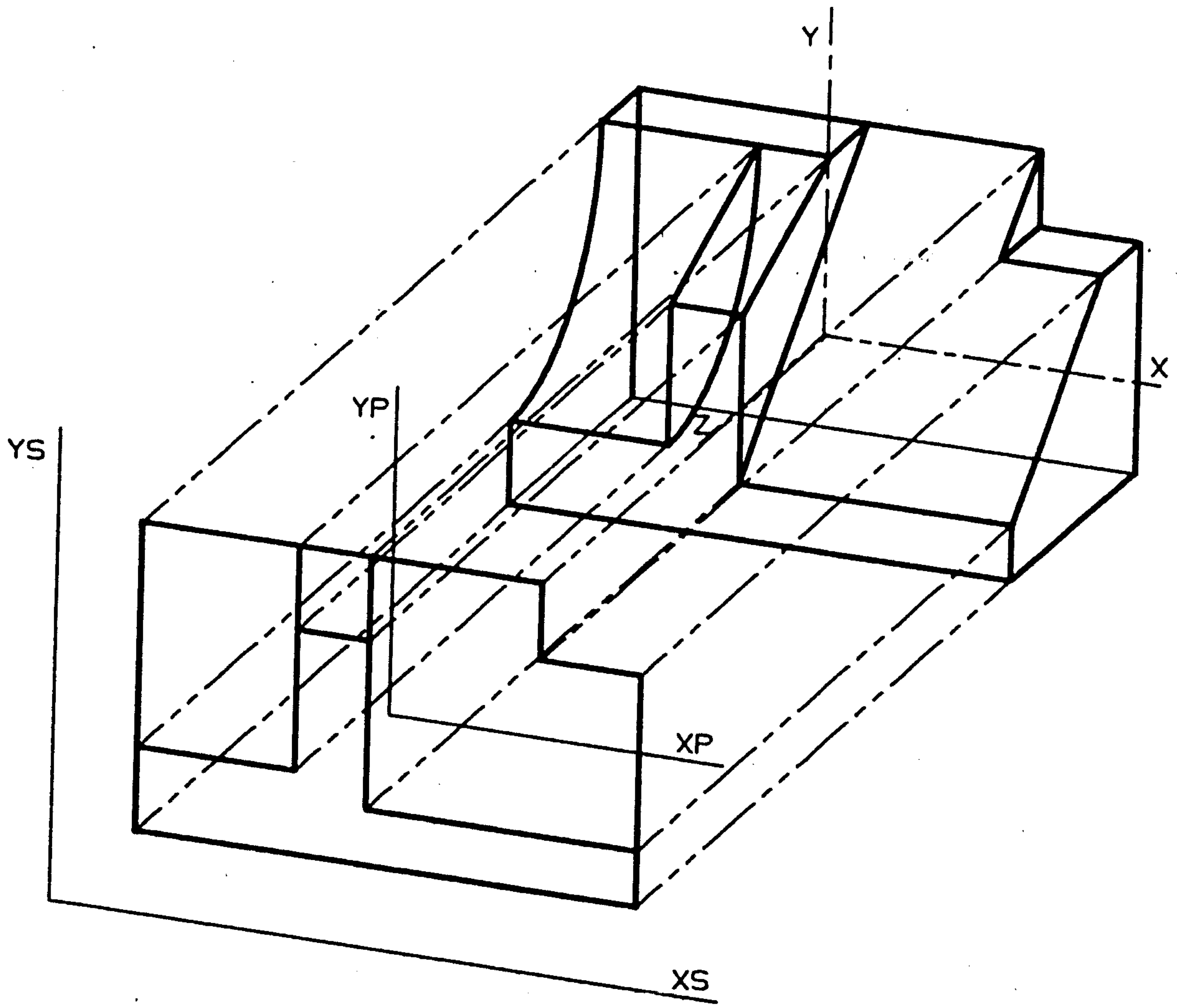


FIG. 3-7. Orthographic Projection

orientations the scaling and conversion steps mentioned above are preceded by a transformation of all model vertices into the desired rotated coordinate system; this can be represented by the equation

$$[x^* \; y^* \; z^*] = [x \; y \; z][T]$$

where x^*, y^* and z^* are vertex coordinates of the rotated object, x, y and z are original model vertex coordinates, and T is the particular transformation matrix.

In this expression the transformation matrix has nine members (three rows, three columns); this form is derivable from the more general 4x4 homogeneous coordinate form and is commonly used in current graphics systems to compute single or compound rotations[4].Further, since the three columns represent mutually orthogonal vectors, the third column of $[T]$ can be obtained by taking the cross product of the first and second columns. In some graphics systems, especially those which provide user-defined auxiliary coordinate systems, such transformation matrices are stored more efficiently by retaining only the first two columns. They are used profusely to map from one coordinate system to another.

View rotations are most simply performed about the model x, y or z axes through some specified angle θ [6]. For single axis rotations the members of the transformation matrix are given as follows:

rotation about x: $[T] = \left\{\begin{matrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{matrix}\right\}$ or expanding $x^* = x$, $y^* = y\cos\theta - z\sin\theta$, $z^* = y\sin\theta + z\cos\theta$

rotation about y: $[T] = \left\{\begin{matrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{matrix}\right\}$ or $x^* = x\cos\theta + z\sin\theta$, $y^* = y$, $z^* = -x\sin\theta + z\cos\theta$

rotation about z: $[T] = \left\{\begin{matrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{matrix}\right\}$ or $x^* = x\cos\theta - y\sin\theta$, $y^* = x\sin\theta + y\cos\theta$, $z^* = z$

Compound rotations can be performed in one step by first calculating the corresponding transformation matrix product; for example, rotation about the y-axis by ϕ degrees followed by rotation about the x-axis θ degrees gives:

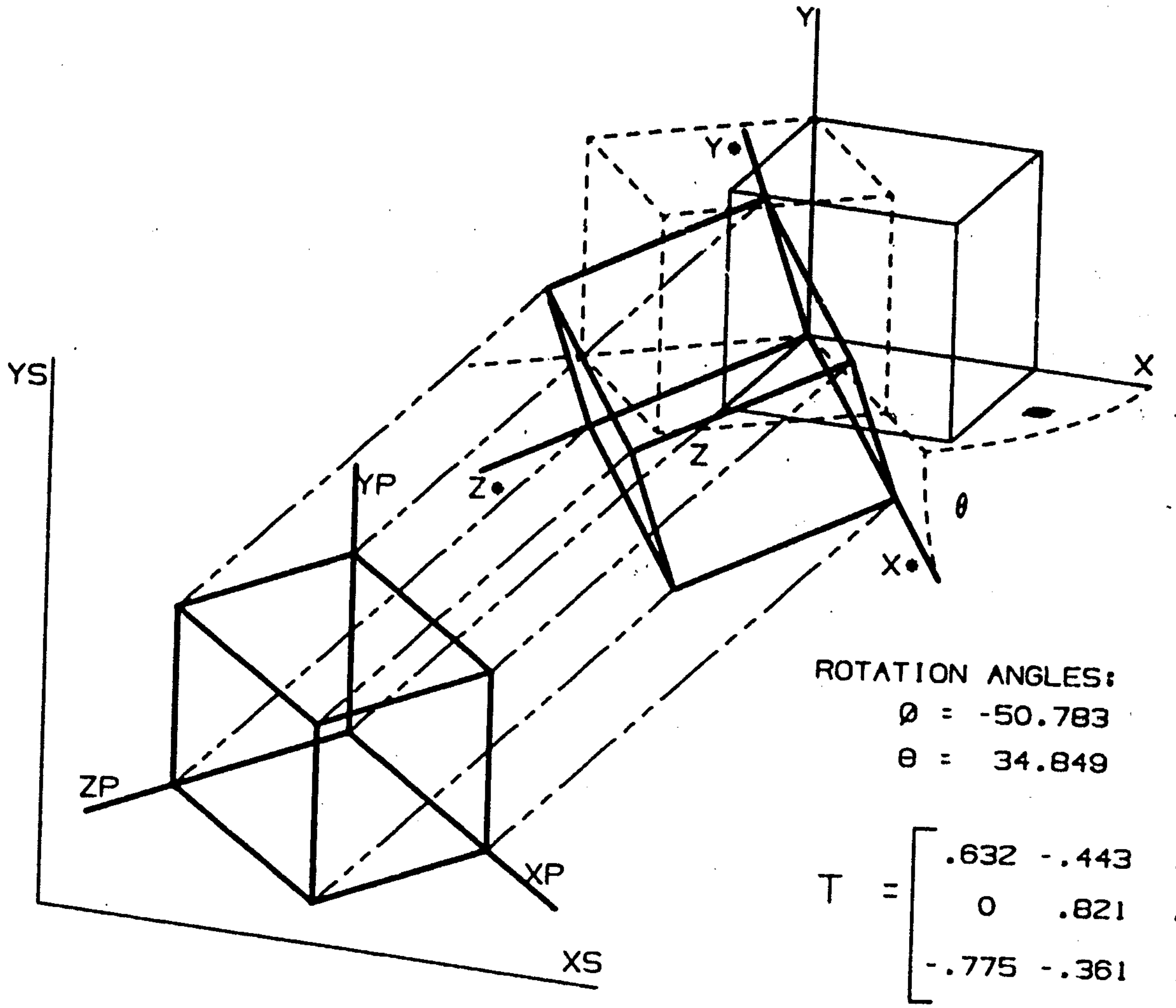
$$[T] = \left\{\begin{matrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{matrix}\right\}\left\{\begin{matrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{matrix}\right\} = \left\{\begin{matrix} \cos\phi & \sin\phi\sin\theta & -\sin\phi\cos\theta \\ 0 & \cos\theta & \sin\theta \\ \sin\phi & -\sin\theta\cos\phi & \cos\phi\cos\theta \end{matrix}\right\}$$

This method is frequently used to produce all three types of classical orthographic projection employed in engineering graphics, referred to as trimetric, dimetric and isometric [3].

The trimetric form is the most general of the three classifications. After rotation, distances measured along each of the three principal axes are foreshortened by unequal amounts. There are an infinity of possible rotations which satisfy this condition; figure 3-8 is a representation of the projection process for a cube, and figure 3-9 shows the resulting image as it would appear on the screen. The rotation angles and the members of the transformation matrix specified thereby also are given in figure 3-9, along with the angles between the axes and the foreshortening of distances in the resulting 2-D projection. The latter details are sufficient for a draftsman to construct manually such a projection on paper. Trimetric projection, however, has rarely been used by draftsmen because of the difficulty in laying out distances along the projected axes.

Dimetric projection is distinguished by the property that two of the three orthogonal planes are foreshortened by equal amounts, and two of the three axes are separated by equal angles. Figures 3-10 and 3-11 show the projection process and the resulting image for such a rotation . This class of projection has been more often used by draftsmen because the foreshortened distances along x_p and y_p are identical, and those along z_p are precisely half of the actual values.

In isometric projection the dimensions along all three axes are equally foreshortened, and the axes appear equally spaced, 120^o apart. Figures 3-12 and 3-13 show the projection process



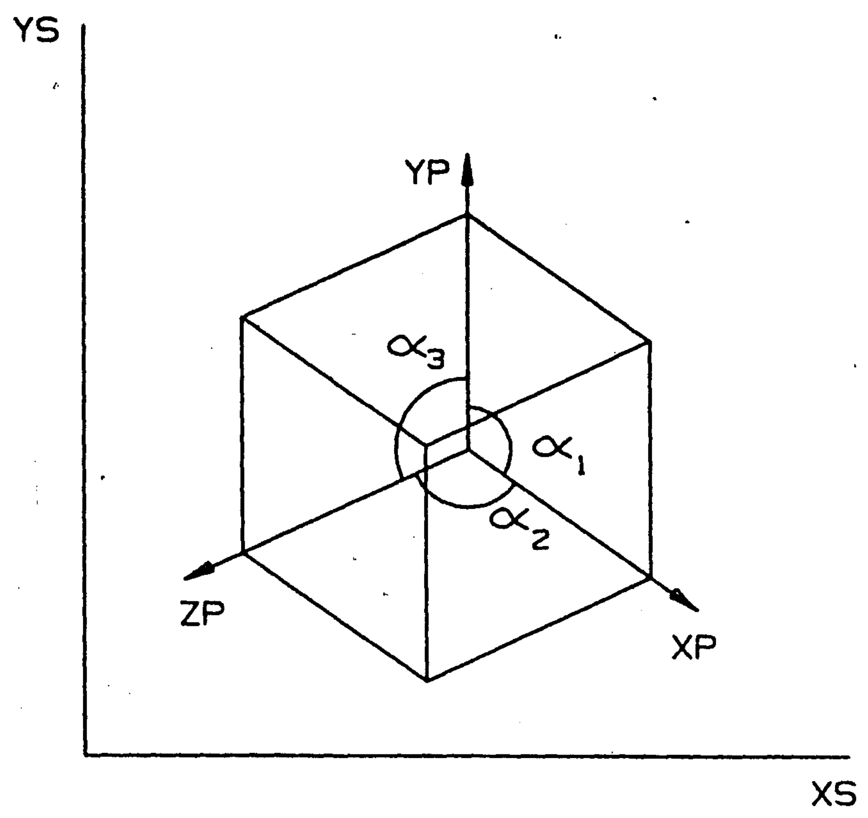
ROTATION ANGLES:

$\emptyset$ = -50.783

θ = 34.849

$$T = \begin{bmatrix} .632 & -.443 & .636 \\ 0 & .821 & .571 \\ -.775 & -.361 & .519 \end{bmatrix}$$

FIG. 3-8. Trimetric Projection Process



PROJECTED ANGLES

α_1 = 125.0
α_2 = 120.0
α_3 = 115.0

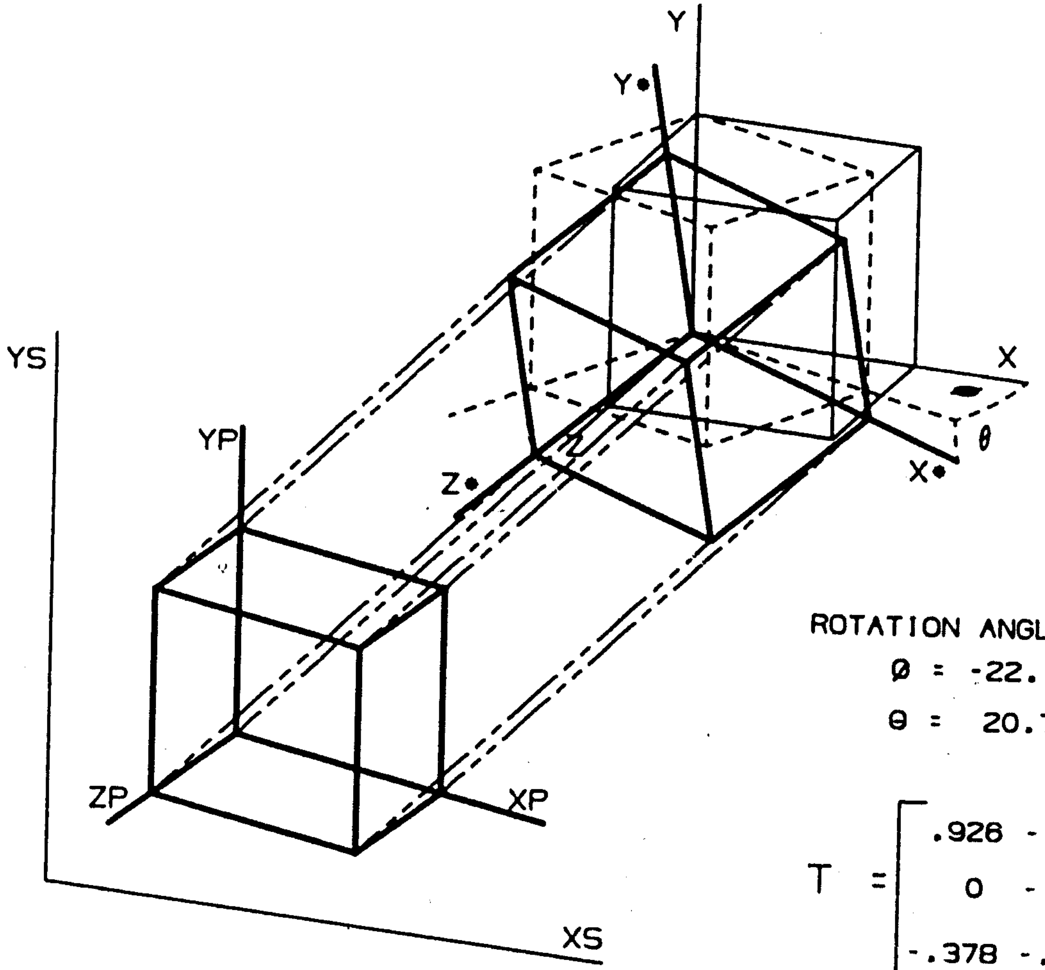
FORESHORTENED DISTANCES

XP: 0.772:1

YP: 0.821:1

ZP: 0.855:1

FIG. 3-9. Trimetric Screen Image



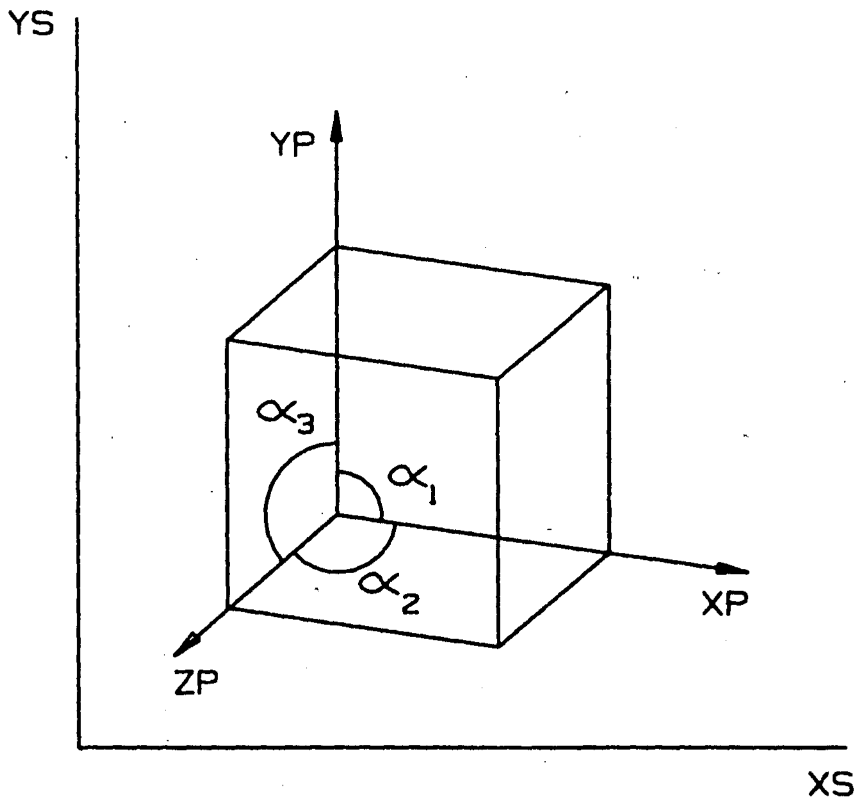
ROTATION ANGLES:

ϕ = -22.18

θ = 20.77

$$T = \begin{bmatrix} .926 & -.134 & .353 \\ 0 & -.935 & .355 \\ -.378 & -.327 & .866 \end{bmatrix}$$

FIG. 3-10. Dimetric Projection Process



PROJECTED ANGLES

$\alpha_1 = 98.2$

$\alpha_2 = 130.9$

$\alpha_3 = 130.9$

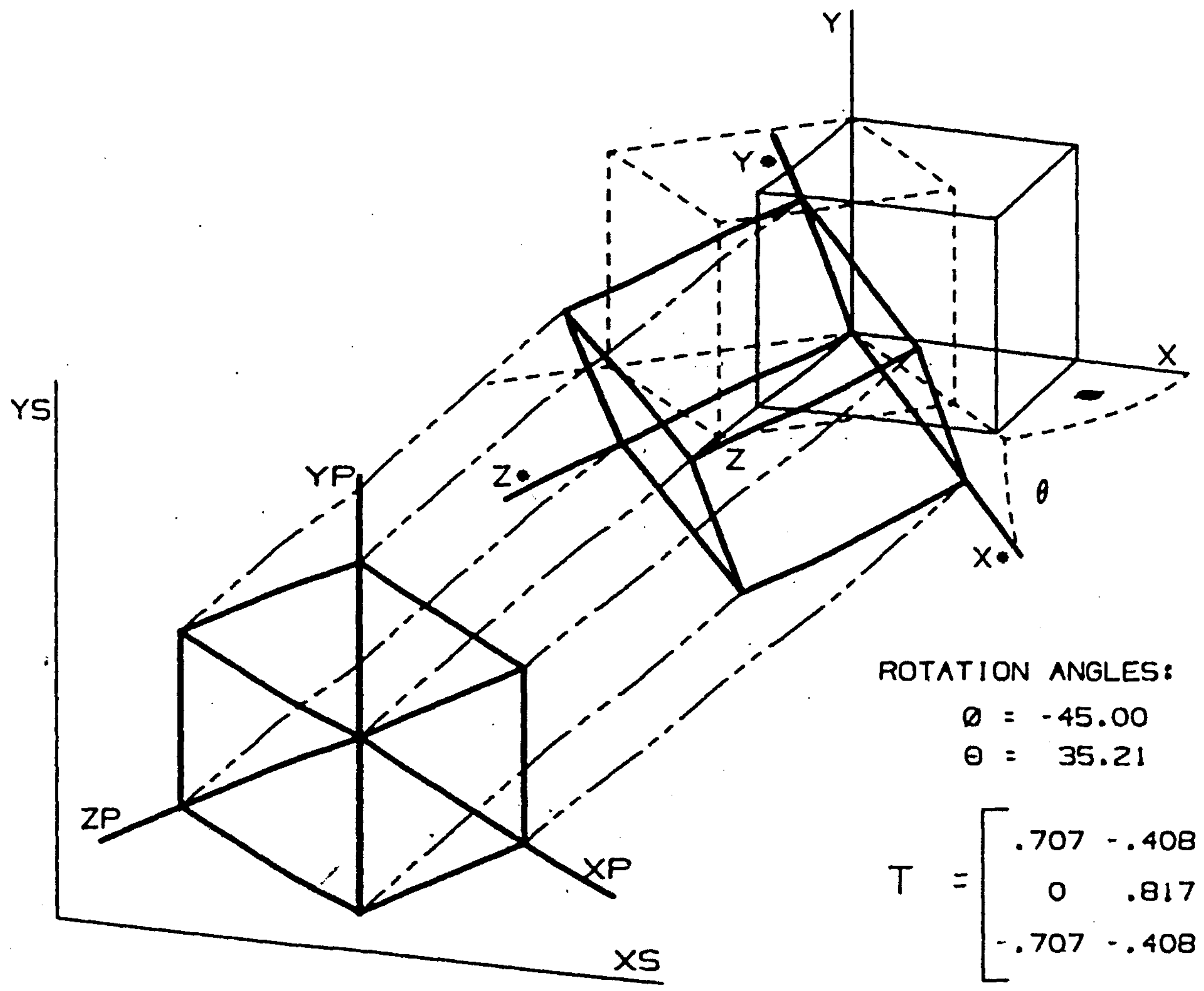
FORESHORTENED DISTANCES

XP: 0.935:1

YP: 0.935:1

ZP: 0.500:1

FIG. 3-11. Dimetric Screen Image



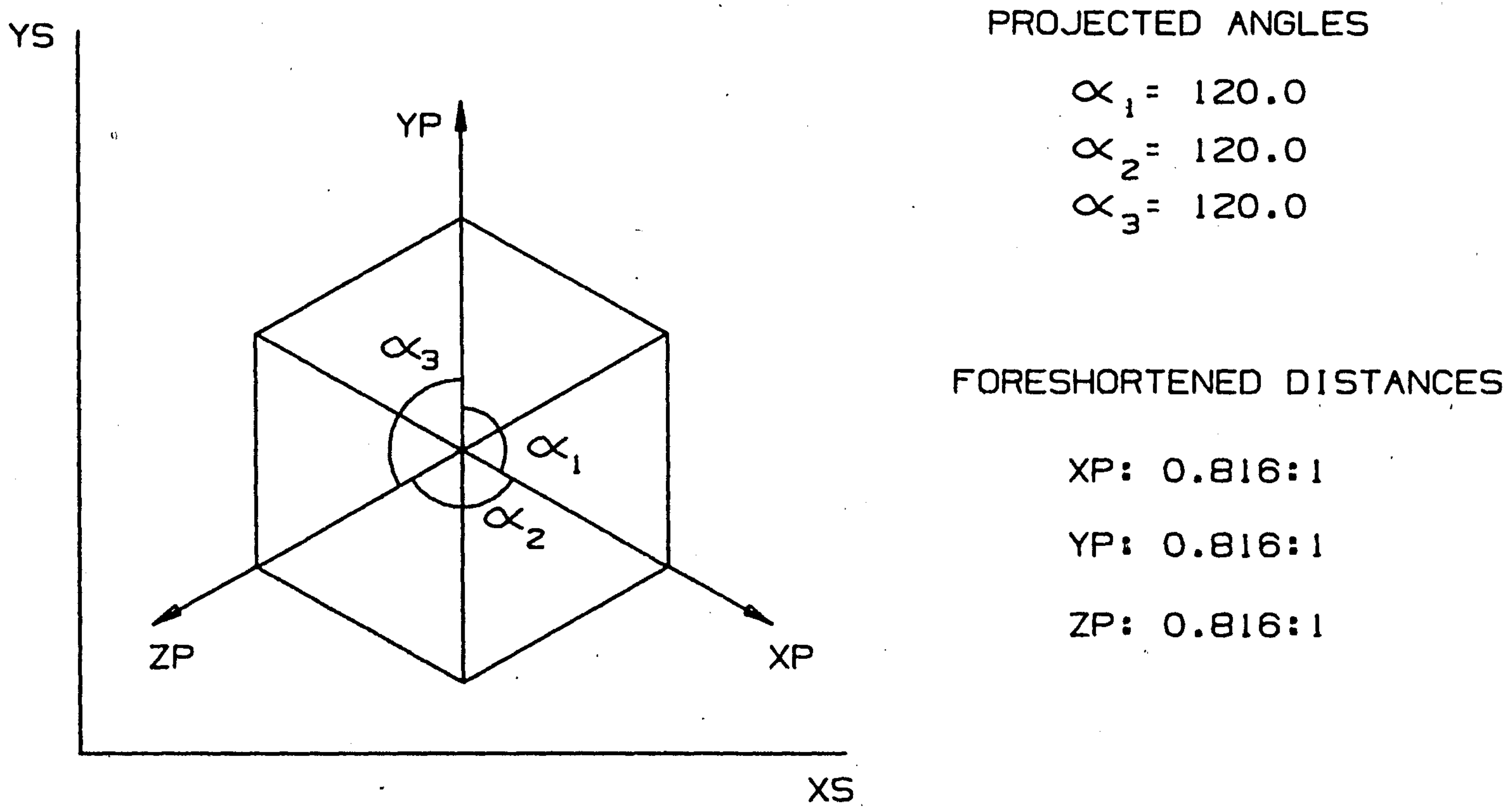
ROTATION ANGLES:

ϕ = -45.00

θ = 35.21

$$T = \begin{bmatrix} .707 & -.408 & .577 \\ 0 & .817 & .578 \\ -.707 & -.408 & .577 \end{bmatrix}$$

FIG. 3-12. Isometric Projection Process



PROJECTED ANGLES

$\alpha_1 = 120.0$
$\alpha_2 = 120.0$
$\alpha_3 = 120.0$

FORESHORTENED DISTANCES

XP: 0.816:1

YP: 0.816:1

ZP: 0.816:1

FIG. 3-13. Isometric Screen Image

and the resulting image. This procedure is most often used by draftsmen when pictorial drawings of 3-D objects are wanted, since the scaling factors applied along all three axes, being equal, can be ignored completely in the construction process.

Comparison of the three screen images illustrated in the preceding examples indicates that the trimetric and dimetric rotations, although difficult to produce on a drawing board, present more information about 3-D objects than the isometric. The more frequently used isometric leads to a somewhat ambiguous image, particularly for the cube in the examples. This observation has made the more arbitrary rotations desirable for pictorial applications in engineering graphics, and most systems accordingly provide, along with predefined front, top and side orthographic and isometric and dimetric view matrices, the means to obtain arbitrary rotations of these standard views.

The most common method performs rotation through a specified angle about one axis of a coordinate system (x_S,y_S,z_S) fixed to the display screen, rather than the model coordinate system. Such rotations are easily visualized, and through repeated execution can give any arbitrary viewing position. Using this technique the operator may, for example, obtain the particular trimetric view shown in figure 3-9 by starting from the system-defined front orthographic view (i.e., model x and y axes are parallel to the screen x_s and y_s axes) and subsequently rotating first -50.783^o about y_s, then 34.849^o about x_s (see figure 3-8). This particular function typically is handled as a compound rotation. The transformation matrix relating the new view to the stored model coordinates is found by pre-multiplying the given rotation matrix by that for the current view:

$$[T] = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix} \begin{Bmatrix} .632 & -.443 & .636 \\ 0 & .821 & .571 \\ -.775 & -.361 & .519 \end{Bmatrix} = \begin{Bmatrix} .632 & -.443 & .636 \\ 0 & .821 & .571 \\ -.775 & -.361 & .519 \end{Bmatrix}$$

Here the matrix on the left defines the front orthographic view, and is in this case simply the identity matrix; in this way the requested rotation reduces directly to that in figure 3-8.

In another common approach a desired axis of rotation and desired angle of rotation is specified. The axis usually is located by its endpoints in model space. This method is not as easy to specify or interpret as the one previously described, but is often useful, for example, in more easily obtaining images in which a skewed face is parallel to the plane of the screen and thus is viewed in true shape.

3.22 Perspective Projection

Some wireframe graphics systems also can generate perspective projections as well as the standard orthographic projections.

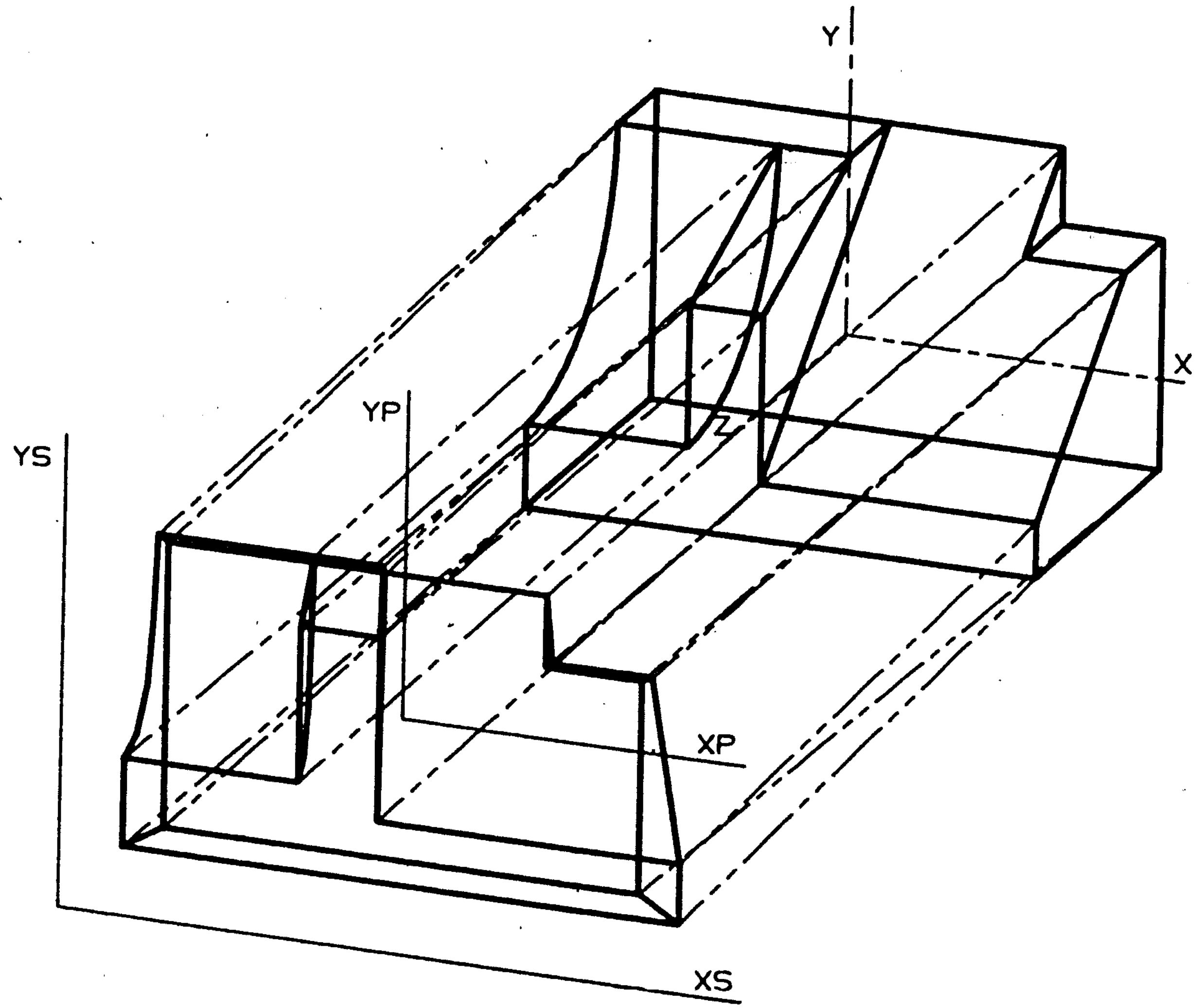
This feature is useful to engineers mostly in a pictorial sense, such as in the production of rotated views for assembly drawings. Figure 3-14 shows the perspective projection process for the same object as figure 3-7. In this form of projection the viewer is located a finite distance from the view plane; projection lines are thus no longer parallel to the line of sight as before, but converge to the specified view point. In this way the model vertex coordinates are modified based on their apparent distance from the viewer. Figure 3-15 shows the screen image as it would appear to the user. Perspective projection clearly is a valuable tool for more effectively conveying depth information in a wireframe image.

The process of calculating view plane x and y coordinates for perspective images is often reduced to the following equations, derivable from the general 4x4 homogeneous coordinate form

$$x_p(i) = x(i) \cdot \left[\frac{V_p}{V_p - z(i)} \right]$$

$$y_p(i) = y(i) \cdot \left[\frac{V_p}{V_p - z(i)} \right]$$

where V_p is the location of the viewer along the model space z-axis. The perspective effect becomes more or less pronounced as the viewpoint is moved closer to or farther from the model space



FIG, 3-14. Perspective Projection

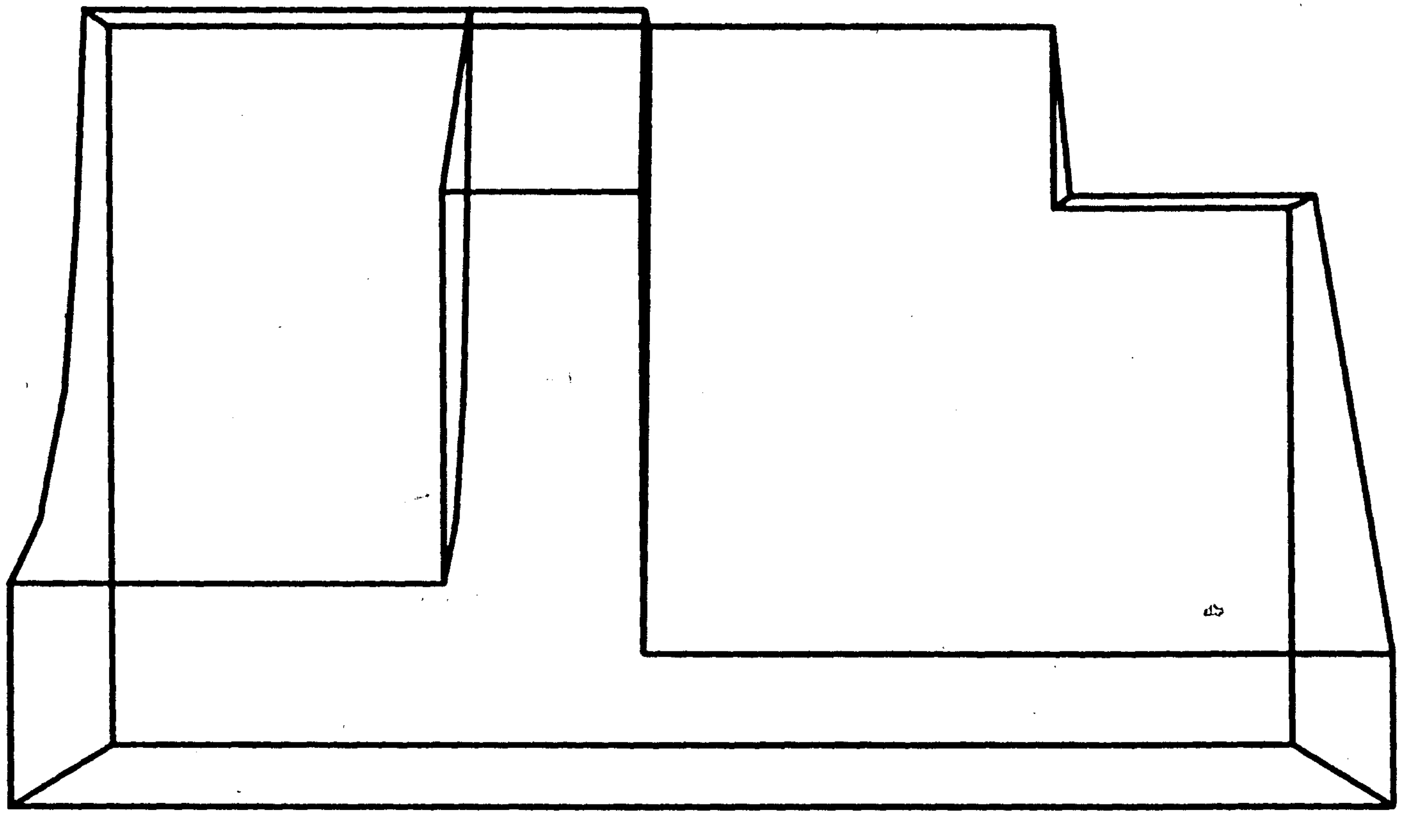


FIG. 3-15. Perspective Screen Image

origin. This calculation is quite costly in processor time, adding both further multiplication and division for each model vertex to the display file creation procedure.

3.23 Multiple Views

The generation of multiple views in one screen display, particularly sets of related orthographic views, is a practical necessity in engineering graphics. It is often an integral part of the model input process. The addressable screen area is partitioned into separate sections for the different views. These independent screen areas are called *viewports*; most current systems permit one, two or four views to be shown simultaneously in this fashion. Figure 3-16 shows such a screen display. In this figure the layout of the four views follows traditional American drafting practice as described previously, with the three orthographic projections (top, front and right side) correctly aligned in the second, third and fourth quadrants, respectively. The upper-right viewport is almost universally used for rotated views; in standard practice this view is developed from the front orthographic, or lower-left viewport. The axes orientation shown in the figure indicates a right-handed coordinate system, with the top view parallel to the x-z plane, the front view parallel to the x-y plane and the right-side view parallel to the y-z plane.

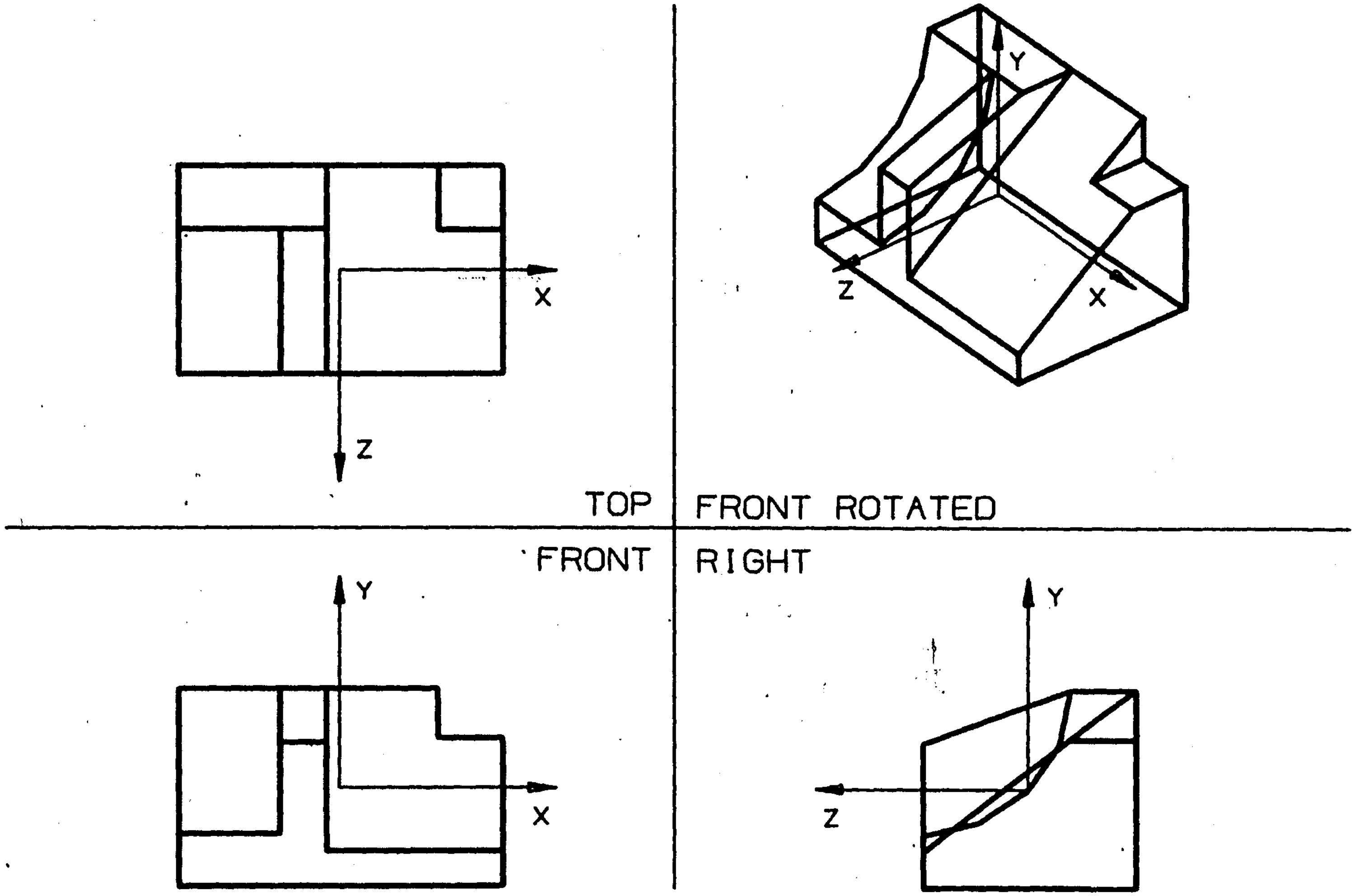


FIG. 3-16. Four-View Display, Front X-Y

Differences in the image generation procedures implemented in current wireframe systems are most noticeable in the properties of multiple-view displays. The more recent systems, in which manufacturing practices have contributed towards the design, typically use a different view labeling convention. In these systems the top view shows the x-y plane; this is consistent with the operation of many numerical-control machine tools, which assume the cutting tool rotates about the z-axis. The front view then becomes the x-z plane, and the right-side view the y-z plane. Figure 3-17 shows the appearance of such a display for the same model as figure 3-16.

A more important difference is that correctly aligned orthographic views are automatically produced only by a minority of these systems. Those which do not perform 3-D clipping, as mentioned in section 3.1, commonly apply a 2-D clipping operation independently to each view. As a result, orthographic sets will not be consistent unless the scale is manually forced to be identical in each view, and furthermore is such that no clipping is necessary in any of the views. A consistent set of orthographic views is always guaranteed regardless of scale only in those systems which use a single 3-D clipping operation to identify simultaneously visible data for all three views.

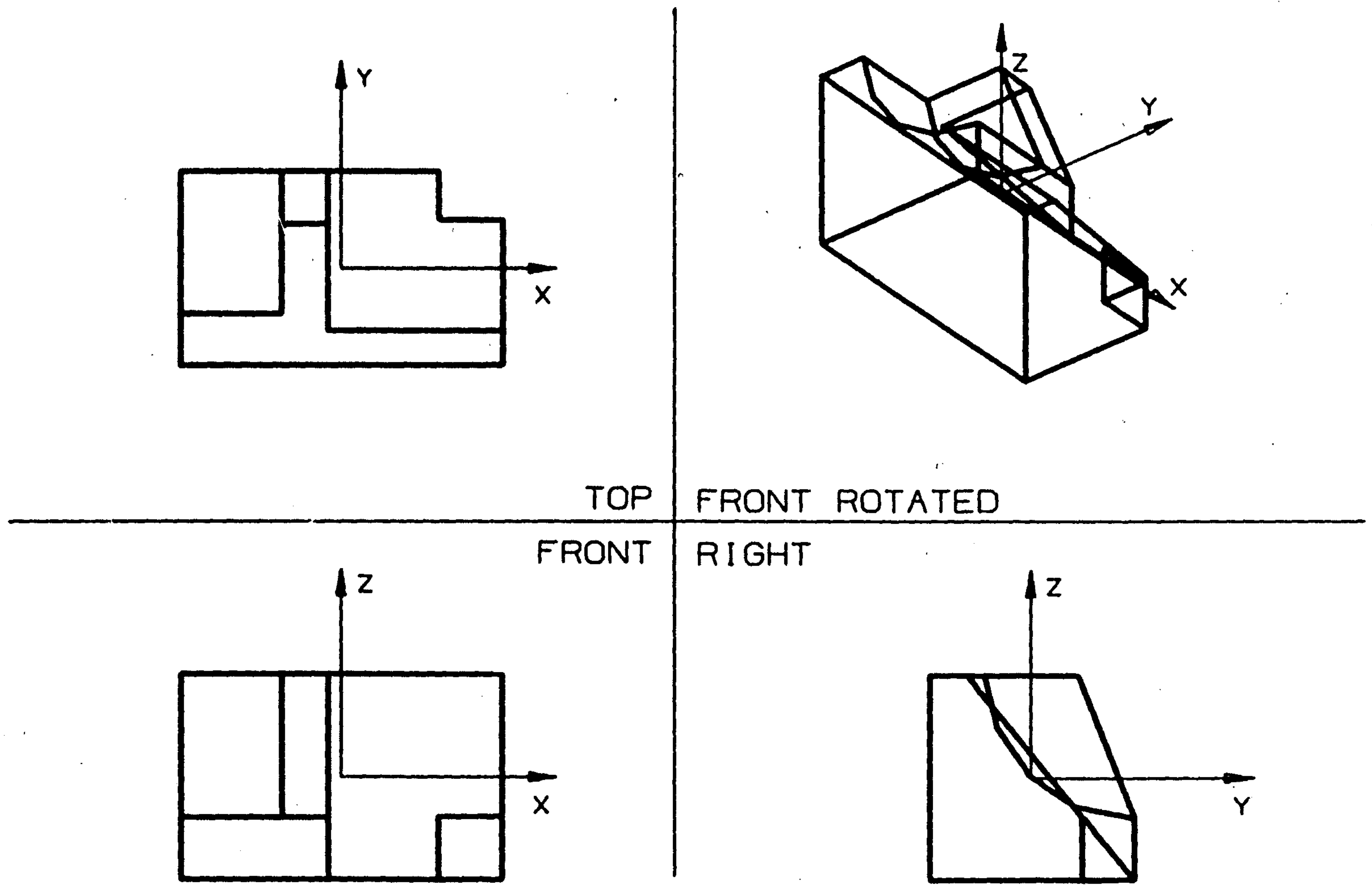


FIG. 3-17. Four-View Display, Top X-Y

3.3 Summary

The entire image creation process is accomplished through execution of a set of graphics procedures applied to the data base in a particular sequence. The processes responsible for creating a display file appropriate for engineering graphics are organized in a three-step procedure in figure 3-18 [4]. Data first is truncated in model space to that portion specified as visible by the location and width of a normally cubic view volume; output may take the form of a list of pertinent geometric entities. Next, the desired projective transformation is applied to the list of points provided by step 1, generating a new list of 2-D view plane coordinates. Finally, these view plane coordinates are mapped into device coordinates in the intended area on the display screen. If multiple viewports are requested, control returns to step 2, computing a different view and mapping to the corresponding viewport, until finished.

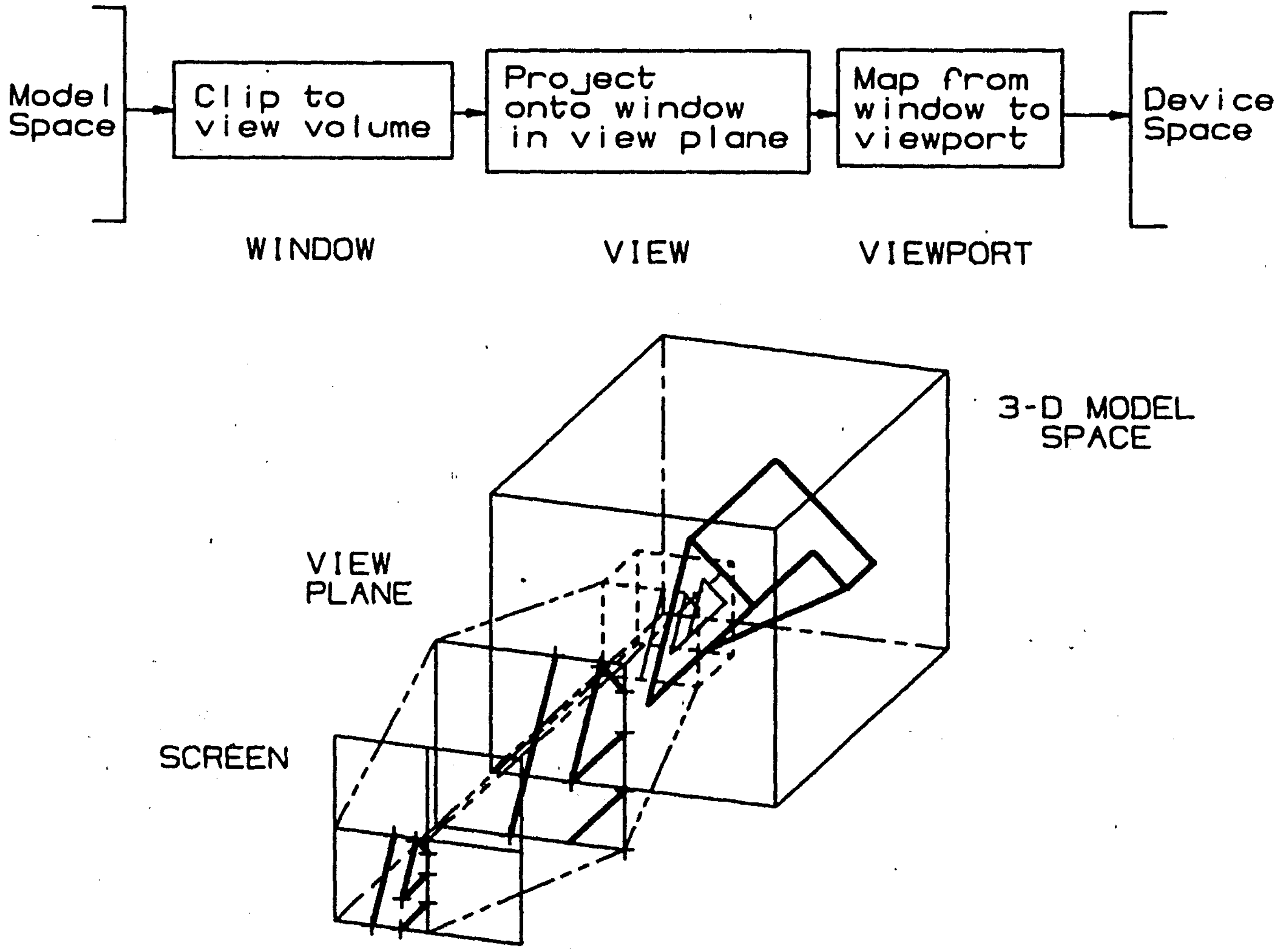


FIG. 3-18. Image Generation Sequence

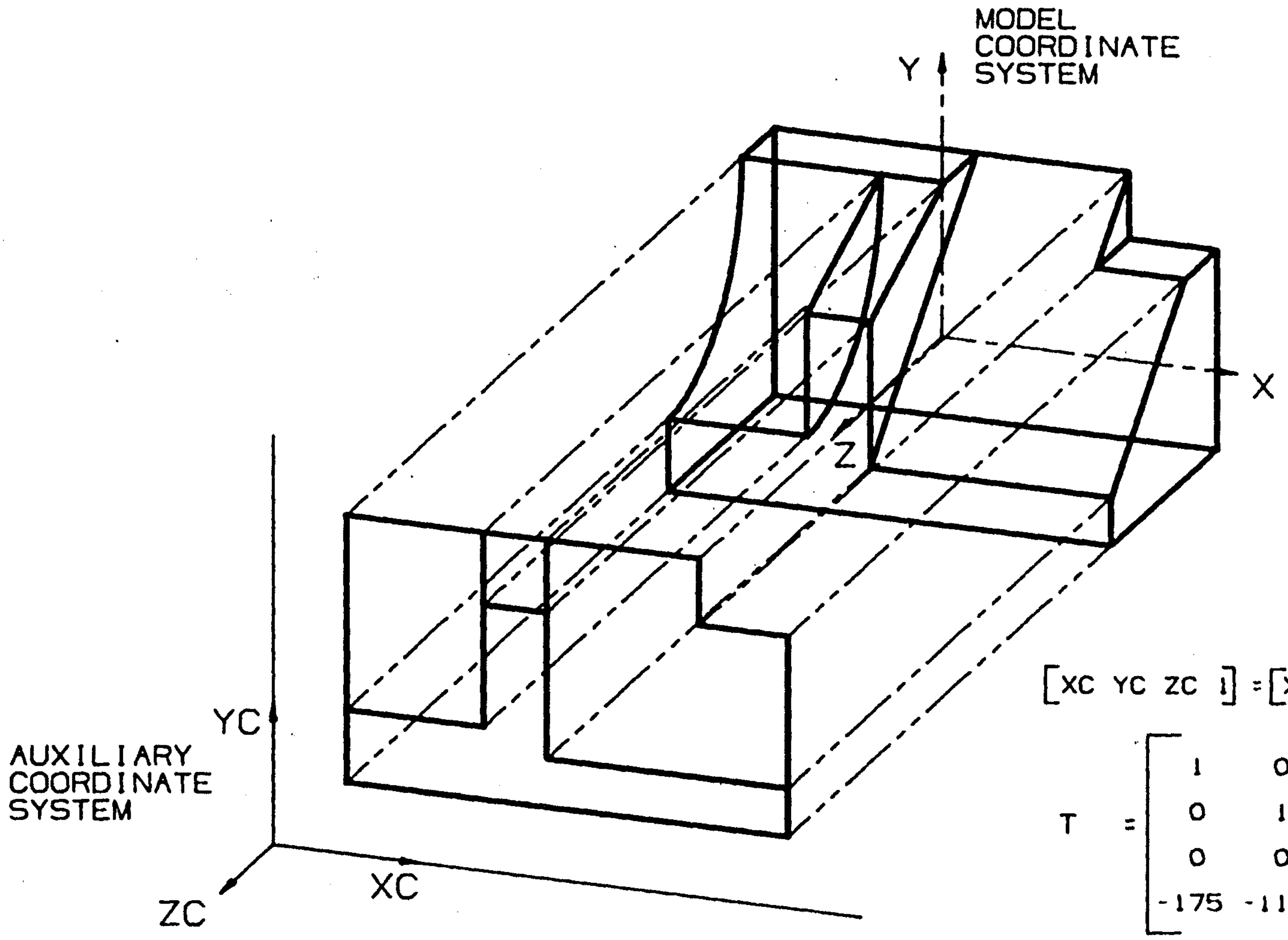
4.0 Interaction Between Operator and System

The software for interactive minicomputer graphics systems is closely related to its more general counterpart, the interactive operating system in larger minicomputer and mainframe systems. Both manage user access to the available hardware and software resources, and can be conceptualized as providing the means of selecting operations from a wide range of alternatives, ideally in the most natural, concise and consistent manner possible. Comparatively, however, interactive computer graphics systems are intrinsically more communication-intensive. The need for interactiveness has inspired a great deal of research and development of computer peripheral devices, particularly those which afford a means of geometric interaction. Currently available systems consequently exhibit a variety of interactive environments, perhaps better evaluated using subjective rather than objective criteria. The principal trade-offs typically occur between flexibility and degree of imposed structure; i.e. freedom in the selection and sequence of modes and options is often compromised with the possibility of more definitive prompting and reduction of user error.

The process of defining model geometry by entering model space point coordinates is usually augmented by one or both of the following two more convenient methods: use of auxiliary coordinate systems, and 2-D graphical input. In one approach auxiliary coordinates called *work coordinates* can be established freely by the user to simplify model construction. This auxiliary coordinate system can be translated or rotated with respect to the fixed model space system; all input is automatically transformed using matrix techniques before addition to the data base. Figure 4-1 shows the use of an auxiliary coordinate system in constructing the model for one of the figures used previously in Chapter 3. An essential goal of wireframe graphics systems is provision of a means of adding to or modifying the object model by interacting graphically with a 2-D image. For this purpose virtually all systems rely on screen cursors or crosshairs, controlled by 2-degree-of-freedom devices such as digitizing tablets, joysticks or thumbwheels.

4.1 Graphical Input

The use of interactive 2-D input in the construction of 3-D models immediately poses the problem of how to map 2-D information into the 3-D data base. Since the fundamental data item in wireframe graphics is the set of three cartesian coordinates of points in model space, 2-D digitizers can be used to supply cursor or crosshair coordinates in a plane which, in the most general case, is arbitrarily oriented in 3-D model space. Specification



$$[XC\ YC\ ZC\ 1] = [X\ Y\ Z\ 1]\,[T]$$

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -175 & -119 & 700 & 1 \end{bmatrix}$$

FIG. 4-1. Auxiliary Coordinate System

of the location of the digitizing plane with respect to the model space coordinate system then suffices to determine the three required model space coordinates. This specification is made conveniently using matrix transformation techniques similar in form to those used for view generation.

The most straightforward manner of handling 2-D input fixes the digitizing plane parallel to a plane formed by two of the three model space axes; e.g., the x-y plane. In this case the required transformation matrix becomes, like that for an x-y orthographic view, the identity matrix; i.e., model x-axis is aligned with the digitizer horizontal, model y-axis is aligned with the digitizer vertical. The missing third coordinate, commonly referred to as *depth* (here it is the model z-axis), is simply fixed at some prescribed default value. To enter a point entity in the model data base, the user simply positions a cursor superimposed on an x-y orthographic view at the desired location in that plane; the system then writes these x and y values, after the usual scaling and type conversion, directly into a new entity record. The current depth value provides the z-coordinate. Figure 4-2 represents this situation graphically. A square "screen" area with a small cursor is shown on the left; on the right the orientation of the digitizing plane in 3-D model space for default condition z=0 is shown, along with the

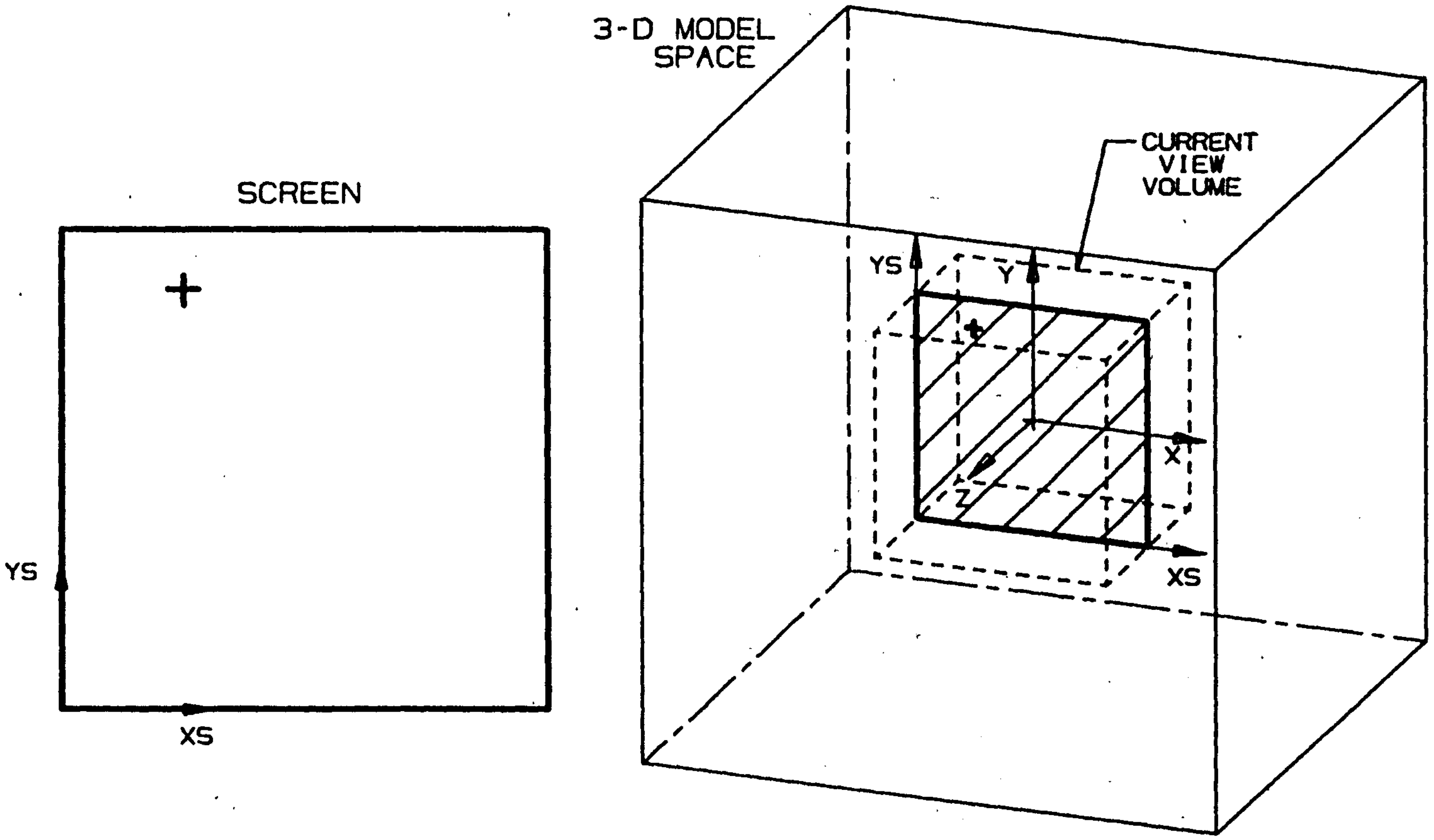


FIG. 4-2. Digitizing Plane Orientation

corresponding 3-D position of the screen cursor. By redefining the current depth value, the user can specify points anywhere in the visible portion of model space.

Most systems restrict graphical interaction to a single view, called the *work view*, even when multiple views are displayed. Some, however, permit simultaneous digitizing in two-orthogonal-view displays, with separate depth values maintained for each. In this case the digitizer surface is implicitly partitioned, like the display screen, into two independent sections. Both methods are most effectively used in conjunction with the predefined front, top and right-side views for defining line and arc entities in orthogonal planes. This property is usually exploited by suitable orientation of the object with respect to the model space coordinate system. The definition of geometry that is not parallel to one of the principal planes is possible but less convenient. A skewed line, for example, can be entered by first digitizing one endpoint; the current depth value for that viewport must then be altered, in the middle of the line definition process, to correctly digitize the remaining endpoint in a different plane. Figure 4-3 represents the process of defining such a skewed line in a largely orthogonal object.

The practical limits of device resolution and repeatability are serious disadvantages of direct analog positional input.

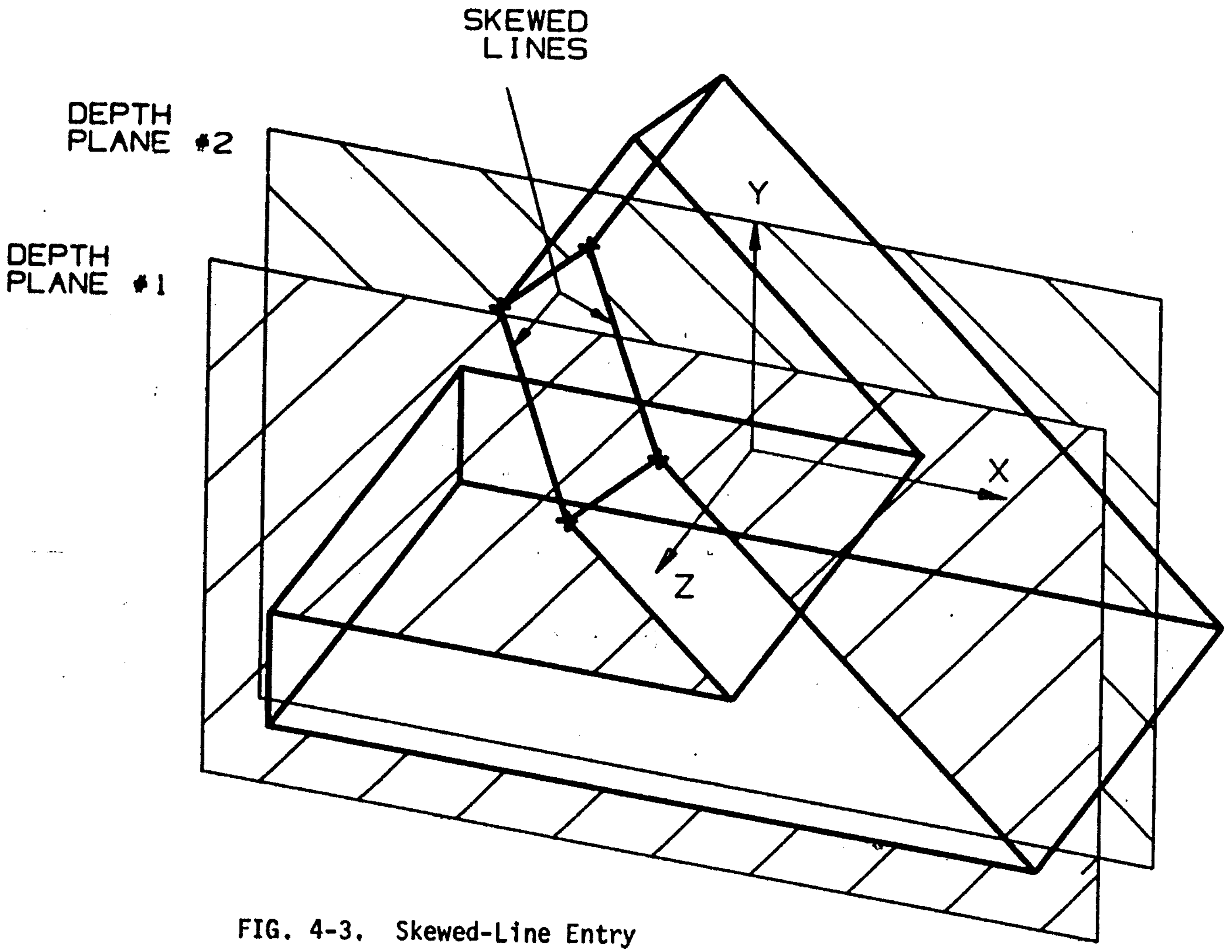


FIG. 4-3. Skewed-Line Entry

Designers using a graphics system to define manufactured parts will, furthermore, have little use for a coarse digitizing capability, as points and lines representing the corners and edges of a 3-D object must be located precisely in the model coordinate system. The inherent simplicity of point and line entry by "pointing" a cursor is such a desirable feature, however, that several techniques have been developed to retain its flexibility while still affording precise data entry.

Foremost among these is the artificial imposition of a modular geometric constraint, in the form of a software-generated digitizing *grid*. Such grid systems filter all positional input, adjusting it to the nearest "currently acceptable value" before addition to the model data base. Figure 4-4 shows a set of orthographic views from such a system; the grid is typically displayed as a pattern of dots which locate the currently acceptable positions. For example, specification of a grid with 1-unit spacing in x, y and z ensures that all subsequent positional input is adjusted to be precisely modulo 1 unit. The user need only set the depth value and then point a cursor anywhere within the box shown here in the front view to identify correctly point coordinates in integral units.

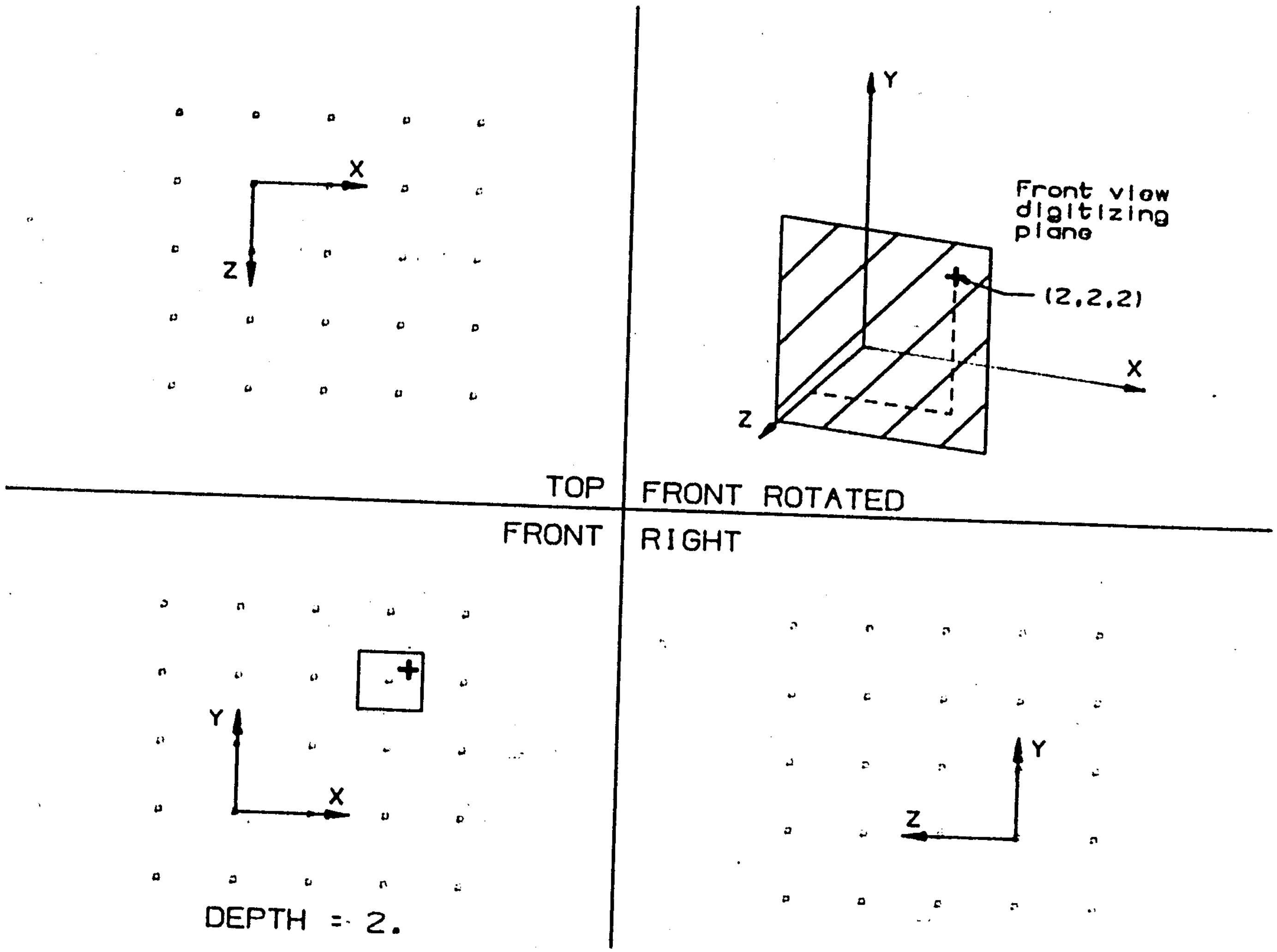


FIG. 4-4. Digitizing Grid

4.2 Entity Selection

In its other essential function, positional input references existing model geometry, either to edit the object model graphically or to define new entities related geometrically to those already entered (parallel, perpendicular, etc.). This procedure is similar to the grid function in that rapid and effective interaction requires the system to be tolerant of user inaccuracy in "pointing" at desired entities. Most algorithms define a region surrounding the indicated point and perform a linear search of the data base for entities falling within this area. Most also stop at the first match, though some complete an exhaustive search. All ultimately indicate in the displayed image the entity or entities found. Both processes are prone to failure; often in crowded images the data base search will consistently return with an unintended entity, or, in the latter case, the desired one along with several others. To alleviate this shortcoming, some systems allow the user to limit the search to particular entity types such as points, lines or arcs. Many systems also permit the user to perform selection in rotated views; although the required coordinate transformation introduces further inaccuracy, rotated views often present object edges most clearly. As a last resort, virtually all systems permit the user to group entities on separate *levels* or *layers*.

One or more layers then can be made ineligible for selection. As mentioned in Chapter 1, the identification of layers normally is included as part of the attribute information written in each entity record.

4.3 Function Selection

Current wireframe graphics systems are often evaluated by users according to the dominant characteristics presented by the user/system interface. The degree of inherent structure is perhaps the foremost of these characteristics; available systems can be generally categorized as menu-oriented systems, or as command language systems. The salient features of both types and the particular styles of interaction which each evokes are described below.

4.31 Menu-Oriented Systems

This group includes the most highly-structured systems. In this form of operation the user is prompted at all times with a brief, fixed list of options, from which a single function is chosen. The user then is presented with further sets of lower-level sub-menus until finally a particular operation supported in the software library is fully specified. The desired procedure is then executed, and the user is normally returned upon completion to the top of the menu structure. In this manner the user chooses from the available resources in fixed, predetermined patterns which ensure correct and unambiguous specification of all required parameters. Option lists in force at any particular time are often presented as character strings on the main graphics display, or alternatively on a small alphanumeric monitor. Choice is made by pointing a cursor or by pushing designated buttons on a function keyboard. In one such implementation the meaning of sets of function buttons changes with each menu level, always corresponding to options active in the current menu.

The extensive prompting issued by these systems makes them easy to learn, and beginners usually progress rapidly in this instructive environment. Skilled users, however, may come to find the hierarchical menu structure cumbersome.

4.32 Command Language Systems

In these systems a general command syntax is defined, and each function is assigned a unique keyword and parameter sequence. The user may choose at any time between all of the possible functions supported by the system; all pertinent information must be correctly specified with each command. In more primitive versions the user enters commands as alphanumeric strings using a standard keyboard, similar to mainframe interactive programming systems. More advanced implementations make use of function buttons or tablet menus, each button or digitizer area representing a predefined, or sometimes user-defined command string.

In these systems the user is prompted little if at all. The price of greater flexibility is the need to learn the syntax and data requirements of the full graphics instruction set. New users often require substantially more training than they would for menu-oriented systems. They must have available a far greater amount of reference information in order to use the system at all. Once this information is acquired, however, skilled users often can define complex models more rapidly without the exhaustive prompting structure of menu-oriented systems.

4.4 Graphical Interaction

The possibility of selecting menu options with the screen cursor was mentioned in the previous section. This is an example of using graphical input, not as geometric data, but to control system operation. Since the operator frequently alternates between selecting functions and selecting or defining geometry, it is convenient that the same device be used for both. This notion is extended further in systems using symbol recognition. This capability permits the user to associate freehand symbols sketched on a digitizing tablet with system command strings. By including procedures to extract specific positional information from symbols, as well as matching them with system commands, the processes of selecting functions and defining geometry can be combined. In this manner both a function and its parameters can be given simultaneously. As an example, the user may choose to associate a horizontal or vertical line with the procedure for defining line entities, requesting the extraction of the first and last points of the sketched symbol. Horizontal or vertical lines then can be entered very rapidly simply by sketching a line on the digitizer connecting two grid points in the displayed image.

Such features represent some direct advantages of computer graphics systems over manual drafting; indirectly there arises the added benefit of providing data for further use in analysis and manufacturing. Although the above example is peculiar to command language systems, the particular combination and utilization of hardware and software in many systems permits comparable economies in model construction. The following chapter will discuss more powerful procedures commonly installed in current systems to promote rapid construction of 3-D models.

5.0 Model Construction Enhancements

Several higher-level graphics functions are commonly provided in current wireframe systems to simplify the construction of 3-D models. These functions generally enable the user to reference existing entities in the definition of new ones, either to satisfy common geometric relations or to perform some type of spatial duplication. The set of functions implemented in any one system often determines the particular construction techniques most efficient for that system.

5.1 Geometric Relational Expressions

The use of geometric relations depends on the ability to extract and/or calculate the required information from data already defined in the current model. The principles of analytic geometry make possible a large number of such relations; Table 5-1 shows a list of methods provided for the specification of a new line entity in one particular menu-oriented system. All can be computed from the standard data maintained for each entity type; i.e., three cartesian coordinates for points, six coordinates giving the endpoint locations of each line, etc. In this kind of menu-oriented system, relations not provided explicitly frequently can be obtained by executing a series of the supported functions, creating and later deleting intermediate

```
CHOOSE LINE DEF MODE

1   TWO POINTS
2   TAN TO 2 CURVES
3   HORIZONTAL
4   VERTICAL
5   NORMAL
6   PT AND TAN CURVE
7   PT AND ABS ANGLE
8   PT - ANG FROM LINE
9   PT - PARALLEL LINE
10 PT - PERPEND CURVE
11 PARALLEL AT DIST
12 PARLEL LN - TAN CURV
13 PERPTO LN - TAN CURV
14 ANGLE - TAN CURVE
```

TABLE 5-1. Line Entity Definition Options

entities. This habit is typically encouraged by menu-oriented systems, and the given functions, in conjunction with a set of entity "editing" procedures, lead the operator into construction techniques similar to those conventionally used in manual drafting. Intermediate entities perform a role similar to construction lines drawn and erased during manual drafting. Though this approach often produces a model data base "cluttered" with redundant or useless entities in inefficient systems, operators with drafting experience are quickly trained in their use. The editing procedures mentioned above include such functions as "trimming" intersecting entities at the point of intersection, modifying one line endpoint, etc.

In contrast to the menu-oriented system methods, command language systems often provide an on-line "geometric calculator" enabling the immediate entry and solution of geometric relational expressions. These procedures function like interpreted algebraic calculators; the user commonly keys in expressions which reference both a temporary storage area and the model data base through selected entities.

A particular relation is solved by specifying temporary storage as output for the appropriate geometric function call, with the selected entity passed as an argument to this function. The user then can execute an entity creation command, adding that parameters are to be retrieved from temporary storage. For example, assume it is desired to add a line entity parallel to an existing instance, offset 5 units in the x-direction; the required sequence may appear as follows:

```
@L1 = PRLL(@S1,5,Ø,Ø)
 ADD LINE (@L1)
```

where @L1 refers to temporary line storage, PRLL is the identifier of the "parallel" function, and @S1 refers to the selected entity. This method obviates the necessity of creating and deleting intermediate entities, though it usually appears quite foreign to new operators experienced in traditional drafting techniques. Function call expressions, however, often can be assigned to user-defined function keyboards or tablet menus, thereby enabling rapid access during model construction.

5.2 Entity Duplication

Duplication of existing geometry is intended to take advantage of the various object symmetries which frequently appear in engineering design. This normally involves the application of coordinate transformations to existing entities in order to create scaled, translated or rotated copies. Translation transformations are most often useful; with this facility an operator can define 3-D objects by entering planar cross-sections and

MICRODEX CORRECTION GUIDE (M-9)

CORRECTION

The preceding document has been re-photographed to assure legibility and its image appears immediately hereafter.

[illegible]

REMINGTON RAND
OFFICE SYSTEMS DIVISION

that parameters are to be retrieved from temporary storage. For example, assume it is desired to add a line entity parallel to an existing instance, offset 5 units in the x-direction; the required sequence may appear as follows:

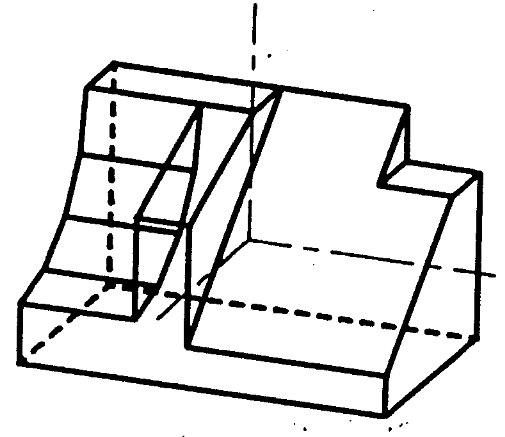
```
@L1 = PRLL(@S1,5,Ø,Ø)
 ADD LINE (@L1)
```

where @L1 refers to temporary line storage, PRLL is the identifier of the "parallel" function, and @S1 refers to the selected entity. This method obviates the necessity of creating and deleting intermediate entities, though it usually appears quite foreign to new operators experienced in traditional drafting techniques. Function call expressions, however, often can be assigned to user-defined function keyboards or tablet menus, thereby enabling rapid access during model construction.

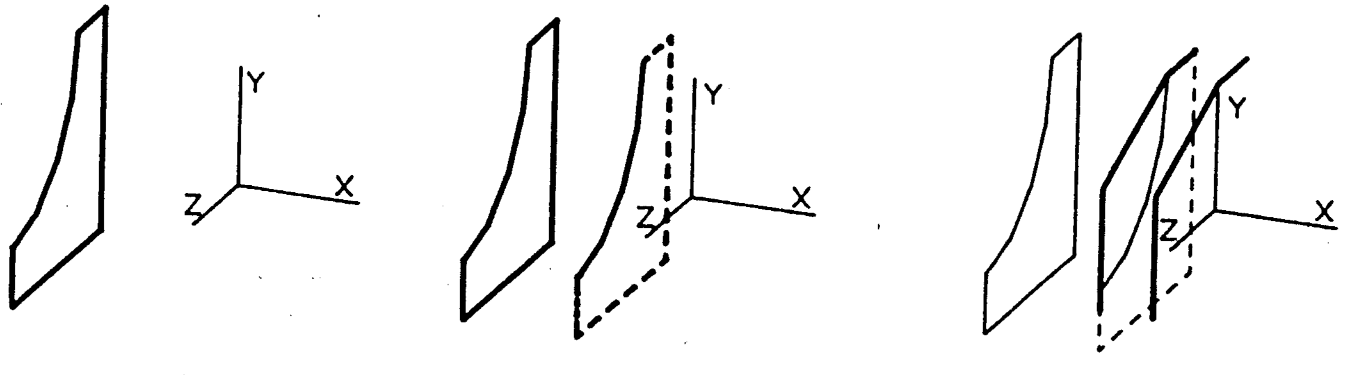
5.2 Entity Duplication

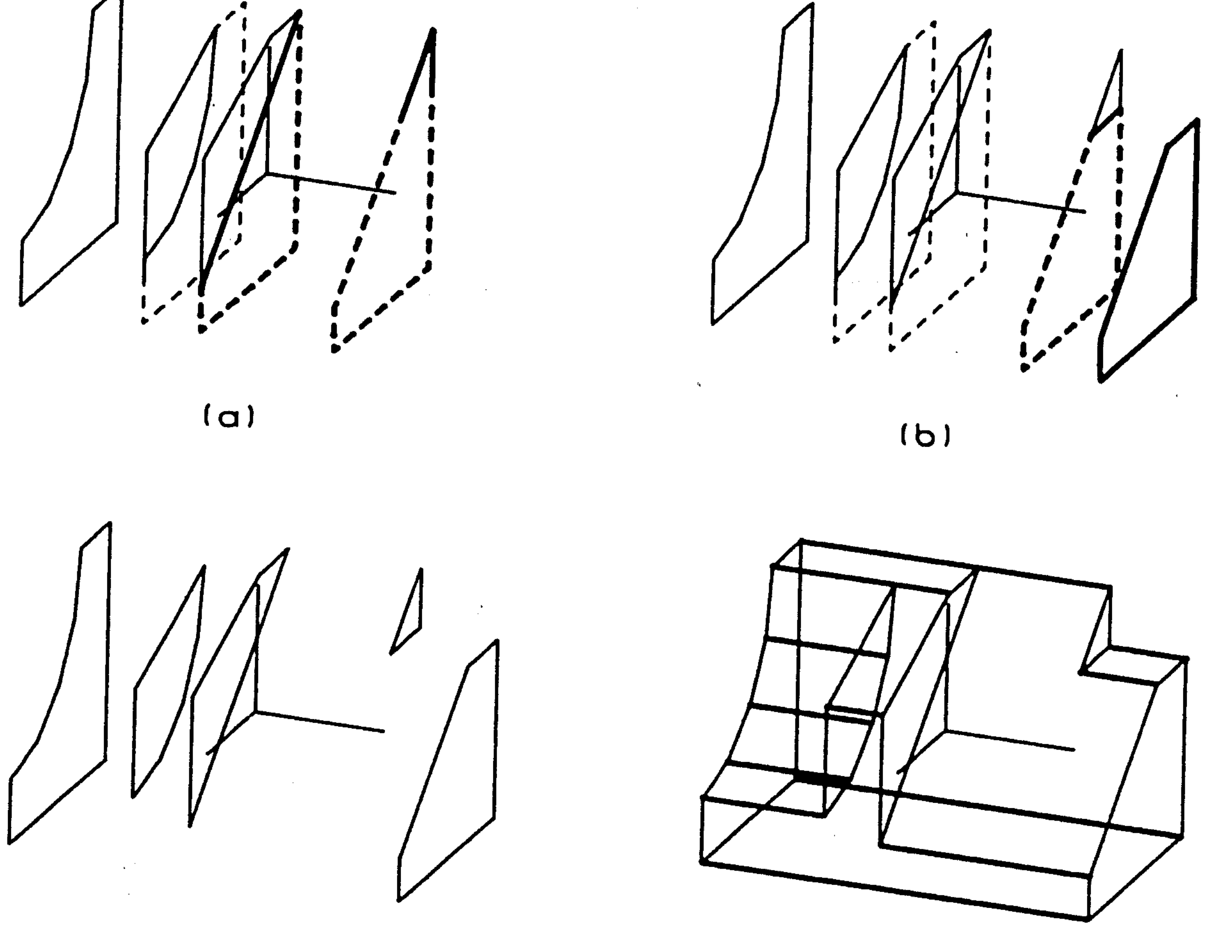
Duplication of existing geometry is intended to take advantage of the various object symmetries which frequently appear in engineering design. This normally involves the application of coordinate transformations to existing entities in order to create scaled, translated or rotated copies. Translation transformations are most often useful; with this facility an operator can define 3-D objects by entering planar cross-sections and then translating a copy in the depth direction. This step avoids re-entering identical, parallel faces. Figure 5-1 shows this process employed in the construction of an object presented earlier in Chapter 1. The orientation of the object in model space is chosen as shown in 5-1a, with the longest dimension parallel to the z-axis, along which four regions of uniform cross-section exist. This form of symmetry can be exploited during the construction process by entering manually only those lines which bound the three different cross-sections. In figure 5-1b the first outline is completed; figure 5-1c shows the result of translating a copy along the z-axis. The remaining figures 5-1d and 5-2a-b illustrate this sequence for the other 2 cross-sections. This process often carries along unwanted edges; the effort needed to delete them, however, usually is small compared to the time saved. Figure 5-2c shows the deletions and in 5-2d the remaining lines have been added, completing the object model. Appropriate choice of object orientation and careful attention to entry sequence is clearly advantageous in systems providing such duplication procedures.

A similar feature provides rapid generation of objects symmetrical about a central plane. In this case the user need only define one side of this plane; the system completes the model by creating entities that are a mirror image of those



(a)




(a)
(b)

already entered. This feature is called *mirroring*. Figure 5-3 shows a model easily constructed using this procedure. With the orientation shown in 5-3a this object is symmetric about the z=0 plane; the designer need only enter the entities shown in 5-3b. The model is completed automatically in 5-3c by the mirroring procedure.

Some systems also provide algorithms for generating planar rectangular or circular arrays of existing entities. These procedures automate the creation of multiple, regularly spaced entities or groups of entities. In rectangular arrays the operator specifies step sizes in two perpendicular directions, and the number of rows and columns. Figure 5-4a shows a rotated view of a ribbed housing constructed using a rectangular array procedure. The operator individually defines only the rib outlines shown in 5-4b; the remaining instances on that side of the central plane are easily added by array duplication. Figure 5-5 depicts the use of array duplication in the case of circular symmetry. The hole circle and centerlines are defined only once by the operator; the remainder of the circular bolt pattern is generated automatically by the array procedure.

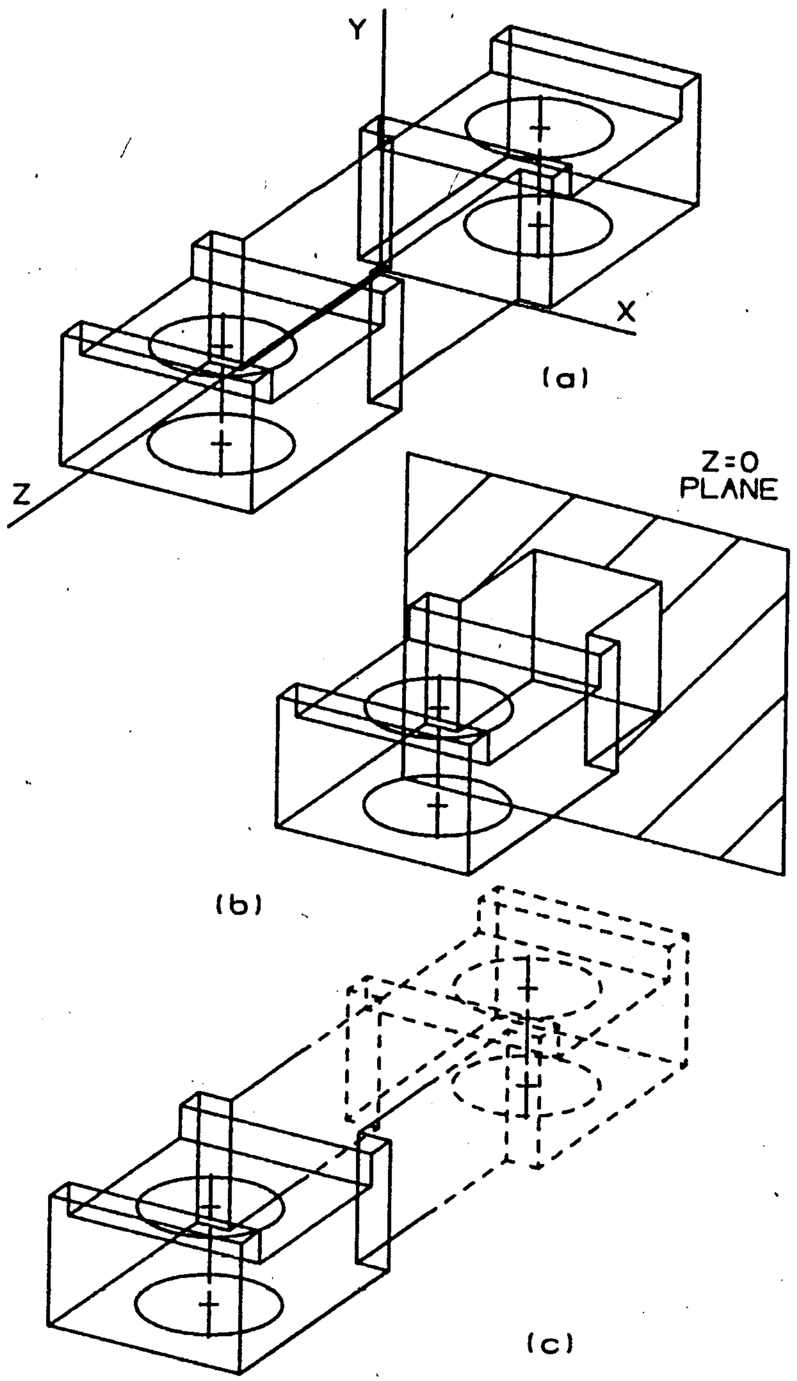


FIG. 5-3. Mirroring Procedure

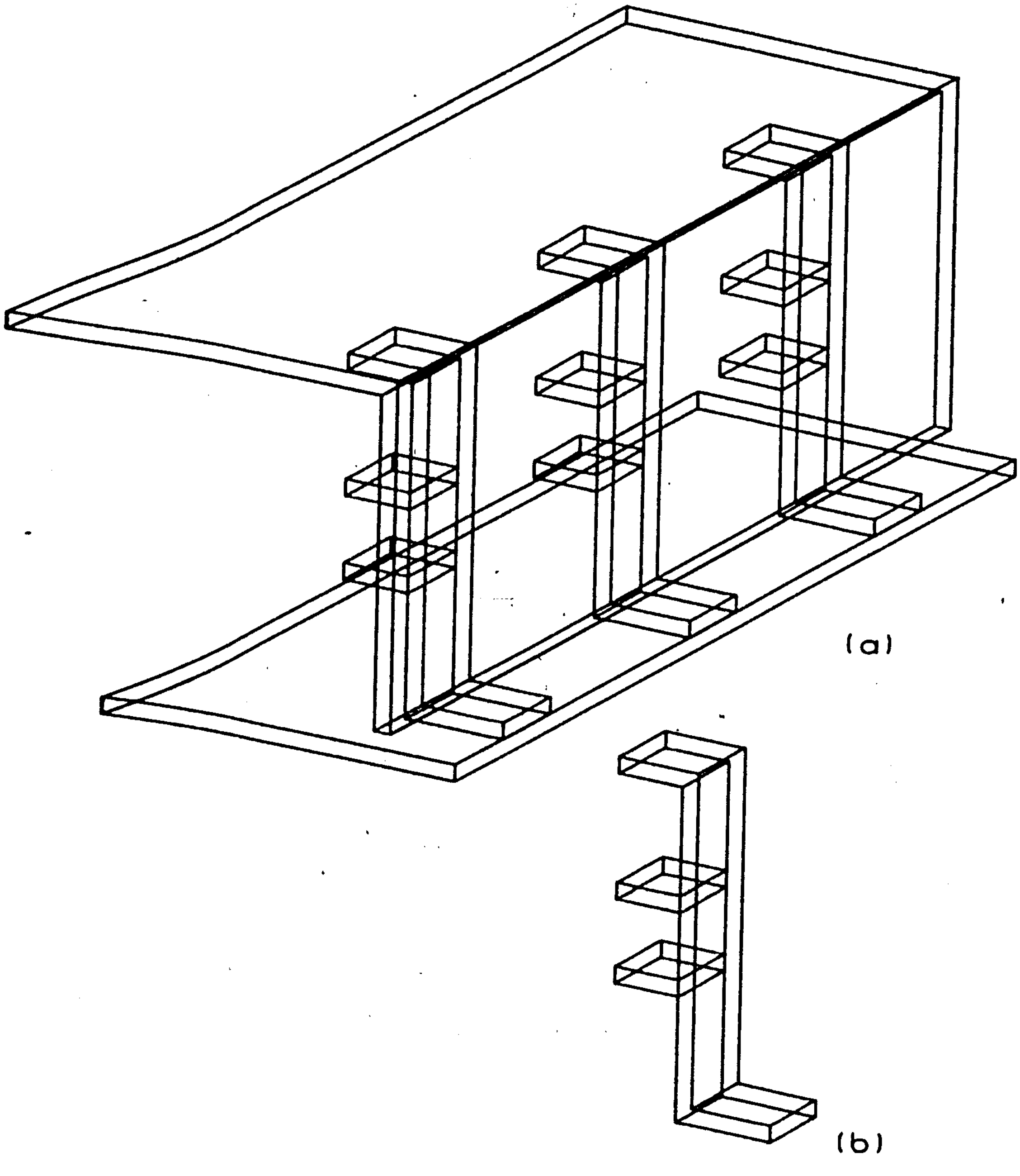
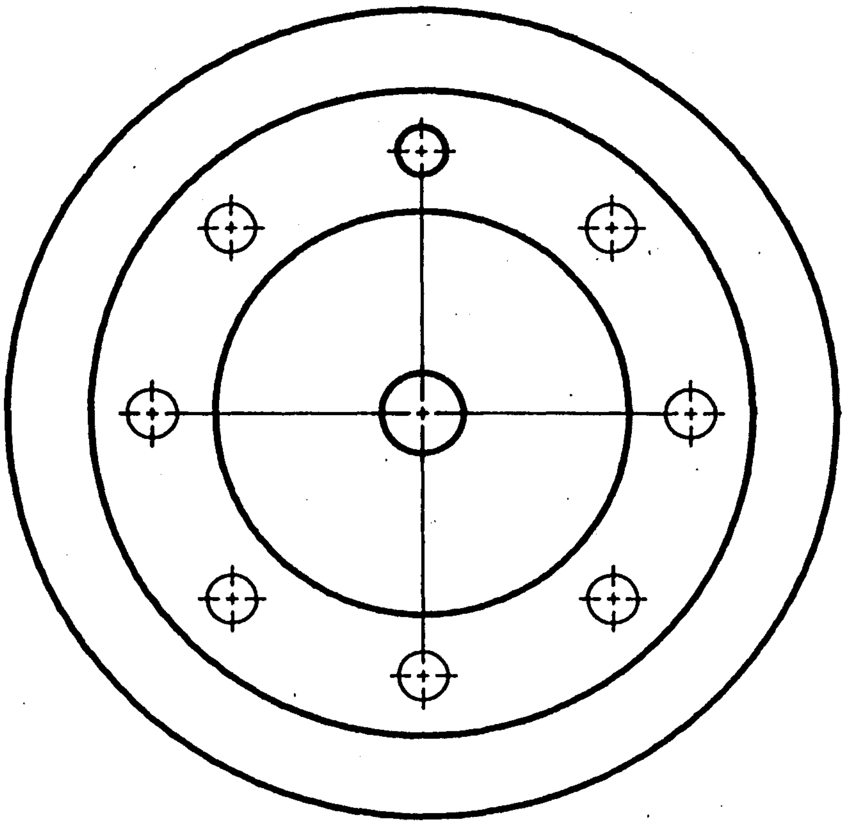


FIG. 5-4. Rectangular Array Procedure



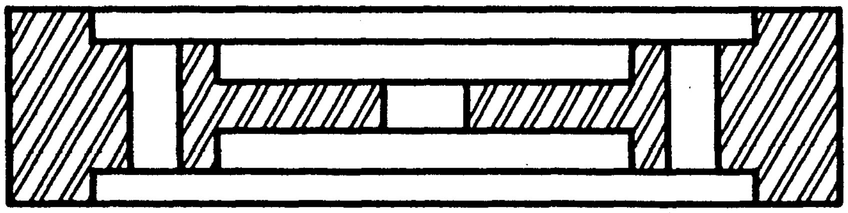


FIG. 5-5. Circular Array Procedure

5.3 Special-Purpose Procedures

The above general forms of automation described above effectively reduce the effort involved in constructing 3-D models only after entry techniques have been developed which permit their use. Each system, in a sense, trains the user in methods often peculiar to that system alone. This result is most frequently observed in menu-oriented systems, where rigid structure limits the scope of alternative construction methods. More advanced systems accordingly provide a means for the user to define special-purpose automation procedures. This feature is a natural extension of command language systems, and is implemented through provision of a macro facility. At least one menu-oriented system supports this capability through a unique 3-D geometric programming language which can freely access the model data base.

Macro interpreters permit the definition of new graphics functions in terms of a sequence of existing system commands; the user may thus, in effect, create his own individual interactive techniques for model construction. The geometric programming language approach is in comparison far more powerful; by working with a more primitive instruction set the user can efficiently execute more complex and extensive operations, perform numerical computation in a more familiar fashion, and define specialized data structures. However, the large amount of effort required to develop compilers or interpreters to implement high-level graphics languages, coupled with the present lack of standards governing graphical operations and data structures frequently has discouraged this approach. Furthermore, users of such facilities must be capable programmers, familiar with text editors and other utility software, which generally is not the case with new graphics system operators. Both methods nonetheless are valuable in helping to match general-purpose graphic systems more effectively to the user's particular application and preferred style of interaction.

5.4 Data Base Accessibility

Further advantages can be obtained through suitable organization of the system-wide data base. Most systems permanently store each model file under a unique alphabetic name in a graphics system-reserved area of mass storage. Since this is usually written in a compressed format for greater space-efficiency, the particular graphics system must itself provide all necessary file management utilities, such as storage and retrieval, generation of sorted lists, archival procedures, etc. Virtually all systems allow the user to retrieve a particular model into the temporary work area and subsequently store it under a different name. This simple procedure is a first step in providing flexible "internal" accessibility. An entire product-line of similar pieces can be built up quickly from a single base-line configuration, saving considerable operator effort.

This concept commonly is extended to provide separate storage of frequently encountered entity groups, called *patterns* or *cells*, again under a unique alphabetic name and in their own library file. In this manner pipe fittings, fasteners or any other commonplace subassemblies can be defined once and recalled at any time for addition to the current model. A reference point is specified when defining patterns, and the standard matrix transformations are normally made available when recalling. In this way the new instance can be located and oriented properly about the location indicated for the reference point in the current model. This feature also is convenient for documentation purposes, as all standard subassemblies referenced in a particular model often can be listed automatically by the system.

A more general data base accessibility also is desirable but is available much less frequently in current graphics systems. The trend is toward greater integration of computer-aided design and analysis, for which various specialized forms of data bases must be built from the geometric model for input to numerical analysis routines. Frequently, potential users of such systems already have purchased or developed analysis packages executing on separate mainframe computer systems. The ability to access the geometric data base for transfer outside of the graphics system is necessary to interface the two machines.

Most graphics systems execute under the supervision of the standard operating system for their particular machine, though many do not supply compilers or other utility software for generating and running user programs. In those that do the above problem is solved, in at least one instance, by providing a set of user-callable FORTRAN subroutines which user programs may reference to gain access indirectly to the geometric data base. Those that do not supply compilers usually substitute procedures which simply write an operating system-accessible data file from the model data base in a predefined format which can be transferred on tape to a mainframe system.

Few current minicomputer-based graphics systems are large enough to enable the execution of large-problem analysis packages, though some 32-bit machines equipped with floating-point processors can do this (with substantial increases in analysis execution time). The advantages of graphical output from analysis routines, instead of the traditional tables of numbers, however, can justify the use of a large machine to handle both interactive design and analysis.

REFERENCES

1. Besant, C.B.: "Computer-Aided Design and Manufacture," Ellis Horwood Limited, Chichester, England, 1980.

2. Chasen, S.H.: "Geometric Principles and Procedures for Computer Graphic Applications," Prentice-Hall, Inc., Englewood Cliffs, N.J., 1978.

3. French, T.E., and Vierck, C.J., "Graphic Science and Design," 3rd ed., McGraw-Hill, Inc., New York, 1970.

4. Giloi, W.K.: "Interactive Computer Graphics," Prentice-Hall, Inc., 1978.

5. Newman, W.M. and Sproull, R.F.: "Principles of Interactive Computer Graphics," 2nd ed., McGraw-Hill, Inc., New York, 1979.

6. Rogers, D.F., and Adams, J.A.: "Mathematical Elements for Computer Graphics", McGraw-Hill, Inc., New York, 1970.

7. Sutherland, I.E., Sproull, R.F. and Schumacker, R.A.: "A Characterization of Ten Hidden-Surface Algorithms," Computing Surveys, Vol. 6, No. 1, March, 1974.

8. Ullman, J.D.: "Fundamental Concepts of Programming Systems," Addison-Wesley Publishing Co., <u>System Manuals</u>
Reading, Mass., 1976.

Applicon, Inc.: "Applicon Graphics System/880 User's Guide," 3 Vols., Applicon, Inc., Burlington, Mass., 01803.

Control Data Corporation: "AD-2000 Reference Manual," 2nd Revision, Control Data Corporation, St. Paul, Minn., June, 1980.

McDonnell Douglas Automation Company: "User Function Manual, UNIGRAPHICS User Task Interface, VAX/VMS," MCAUTO, Cypress, Ca., 90630, September 1980.

MEGATEK Corporation: "TEMPLATE Reference Manual," MEGATEK Corporation, San Diego, Cal., 92121, 1981.

VITA

The author was born in Franklin, New Jersey, 5 January 1957 to Neil C. Cates, Jr. and Winifred A. Cates. He attended the Franklin Public School System through high school, graduating in June 1975. He attended Lehigh University, Bethlehem, Pennsylvania September, 1975 through May, 1979 receiving a B.S. in Mechanical Engineering and a minor in Computer Engineering. He was employed June, 1979 through May, 1980 by the Guidance Systems Division of the Bendix Corporation in Teterboro, New Jersey as an associated engineer in thermal analysis. He returned to Lehigh University June, 1980 as a research assistant to the PLATO CAD/CAM curriculum development project, receiving an M.S. in Mechanical Engineering, June, 1982.