1-1-1978

# A developmental graphics programming package for the Hewlett- Packard 1000 computer system.

James William Chamberlain

# A Developmental Graphics Programming Package for the Hewlett-Packard 1000 Computer System

by

James William Chamberlain

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

Lehigh University
1978

## CERTIFICATE OF APPROVAL

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

March 13, 1979
(date)

Professor in Charge

Chairman of Department

## ACKNOWLEDGEMENTS

I wish to thank Professor Peggy Ota for her help in the creation of this thesis project. I would also like to give credit for the creation of the "conceptual camera programming system" to Joel Katzen whose system I modified with my own ideas.

TABLE OF CONTENTS

## List of Figures

Abstract:

This thesis discusses the development of a software graphics package for the Hewlett-Packard 1000 computer system. The graphics software was written to form a foundation for further development work in two pertinent areas of computer graphics. The first area of development was in data plotting capability. The subroutines created allow a user to graphically represent two-dimensional data in a convenient analytically form. The second area of development was in the emulation of a "conceptual camera programming system," first conceived and developed by Joel E. Katzen, at the University of Pennsylvania. The system as developed in this thesis closely parallels its predecessor with a number of its own unique features. The system allows a user to design three-dimensional animated motion pictures of straight line bodies with ease and versatility on a computer. The software was written entirely in Hewlett-Packard's RTE Fortran IV.

The plotting subroutines aforementioned plot any two one-dimensional data arrays. The plot that is generated is graphed on a two-dimensional user-defined grided plane. The plot may either be a point by point or a line representation of the data.

The "conceptual camera programming system" is a system simulation of an actual animated motion picture camera. As

previously mentioned this system can only handle straight line bodies or straight line approximations to curved surfaces at the present time. The problems addressed in this area deal with system definition, initialization, data bases, system updating, actual body movements (translation and rotation), coordinate system transforms, clipping algorithms, and perspective projections. No actual camera movements are dealt with here although they may be easily integrated into the system. It is the hope that this phase of the system will be developed in the near future.

Therefore, it is the objective of this thesis to provide versatile capabilities in two pertinent areas of computer graphics on the Hewlett-Packard 1000 computer system and to make the systems as easy and convenient as possible to use.

# I. INTRODUCTION

This section contains a short discussion of the three areas of computer graphics that have been developed on the Hewlett-Packard 1000 Computer System.

## A. The "Conceptual Camera Programming System"

As the title implies this system allows the user to imagine the computer as a real and functioning camera. This "camera" can be programmed to perform various pre-determined functions. Since it is not a real camera but a camera simulated visually by the computer and imagined or conceived in the user's mind it can be labeled a "conceptual camera". This "conceptual camera" lends itself to the design and filming of three-dimensional objects much more readily than its real counterpart, since it is really a software system simulation.

The idea for this particular system was derived from a thesis by Joel E. Katzen entitled "A Conceptual Three Dimensional Camera for Computer Animation"(9) The system in this thesis was designed to nearly emulate Katzen's system on a different computer. The computer used was a Hewlett-Packard 1000 Computer System which is a 16-bit machine with 32,000 bytes of memory, a hard disc and tape unit for additional storage. The computer can support five languages Fortran IV, Fortran, Basic, Algol, and its own assembly

language. The four higher level languages are all Hewlett-Packard versions.

The main considerations given to this system simulation are: the ease and versatility of implementation by the user, the ease of visualizing the animated three-dimensional objects and their movements, and the ease of interaction between user and system. For this last consideration the system actively interacts with the user on a conversational basis for initialization of data bases and other information.

The conceptual camera programming system, referred from here on as the CCPS, is just the start of a simulation that in addition to handling complex camera actions, could handle the complicated problems of hidden line elimination, shading, curved surfaces and highlighting.

Thus, the content of this section will only by concerned with the problems of initialization, data base structure, system updating, body system movements, coordinate system transformations, perspective projections, and two-dimensional clipping algorithms.

B. The Plott package

This section deals with a set of subroutines allowing a user to graph two-dimensional data on a plotting plane. The plane is a gridiron surface whose parameters are defined by the user. The subroutines also scale the plotting surface

and can title the axes if the user desires. The sub-routine calls are written in Hewlett-Packard's Fortran IV.

The objectives in creating these software subroutines were to give an user, an easy to use package with some basic abilities in plotting data, ease in interpreting the data from the plot, and the ability to draw analytic judgements or conclusions from the data graphed. The routines were written to be easily implemented by the user with a minimum number of parameter definitions. These subroutines are only the beginning of a package which could include semi-log, log - log, three - dimensional, and multi - data plotting routines.

C. The Three-Dimensional Vector-Matrix Package

This subroutine package was developed primarily to support the vector and matrix manipulations in the CCPS. The subroutines do the basic mathematics common in vector and matrix algebra. Although developed primarily for the CCPS, all routines can be implemented quite easily by a user in other applications. Thus, this package can allow a user to manipulate three-dimensional quantities found in dynamics and statics, electro-magnetic waves and fields, partial differential equations, etc.

Again all the routines are written exclusively in Hewlett-Packard's Fortran IV.

## II. Mathematical Considerations Concerning the CCPS

### A. Introduction

This section is concerned with the mathematical considerations necessary to implement the CCPS. Consideration will be given to the following areas:

(1) Coordinate System Definition

(2) Coordinate System Transformation

(3) Perspective Projection in Three-Dimensions

### B. Coordinate System Definition

To define three-dimensional bodies, and to have these bodies rotate and translate in a three-dimensional space, three coordinate systems must first be defined and initiated These coordinate systems are,

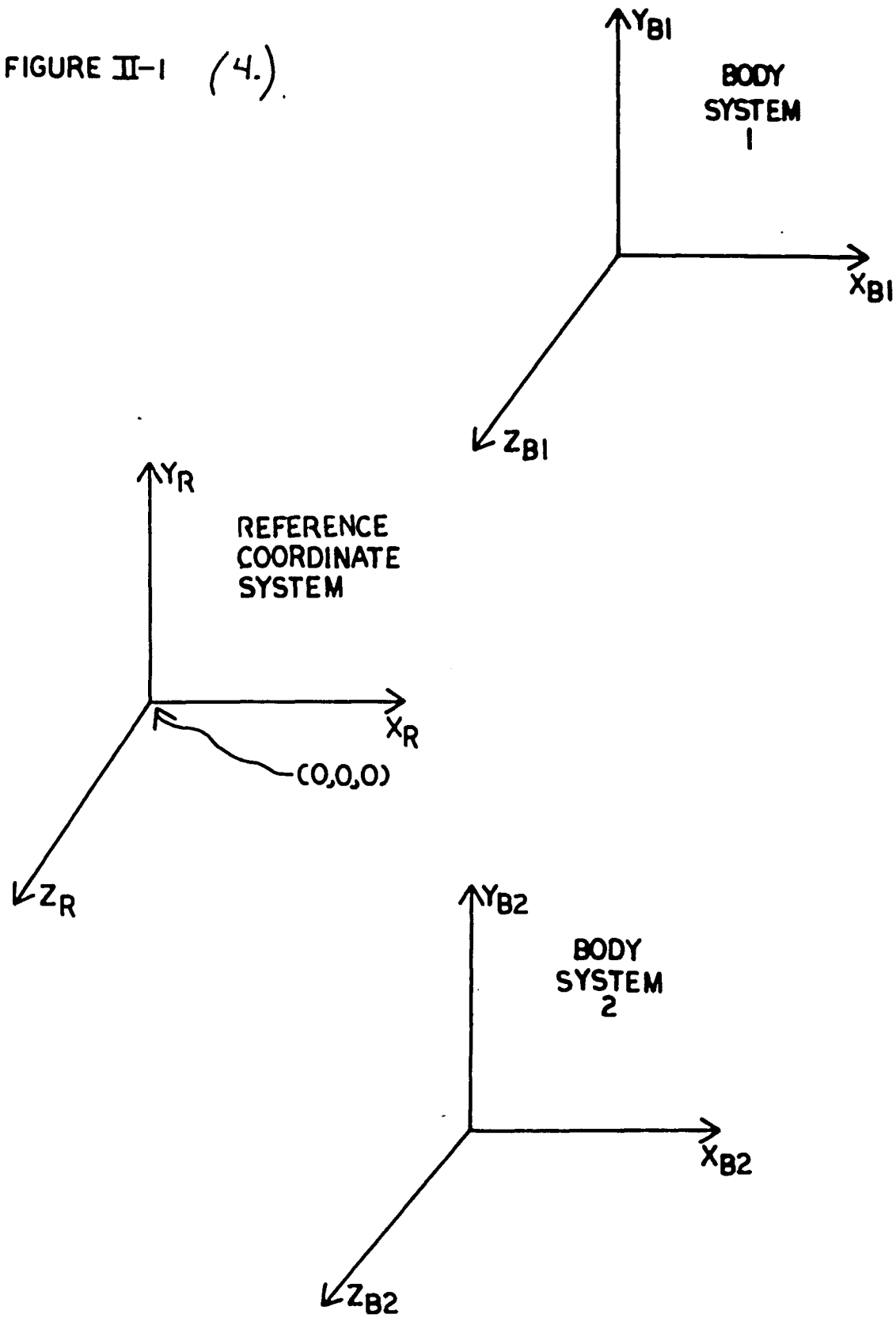
(1) The Body System

(2) The Reference System

(3) The Camera System

The reference coordinate system is the reference by which all other coordinate systems are defined and oriented. Thus its origin is always defined as (0,0,0) for X, Y and Z respectively and it has no angular displacements. The body coordinate system is the initial orientation of each body with respect to the reference system. This can be seen as in Figure II-1, for two bodies. The individual body in each body system is then defined with respect to its coordinate system. Up to four bodies may be defined in this manner in the CCPS.

FIGURE II-1 (4.)


$Y_{B1}$
BODY
SYSTEM
1
$X_{B1}$
$Z_{B1}$
$Y_R$
REFERENCE
COORDINATE
SYSTEM
$X_R$
(0,0,0)
$Z_R$
$Y_{B2}$
BODY
SYSTEM
2
$X_{B2}$
$Z_{B2}$

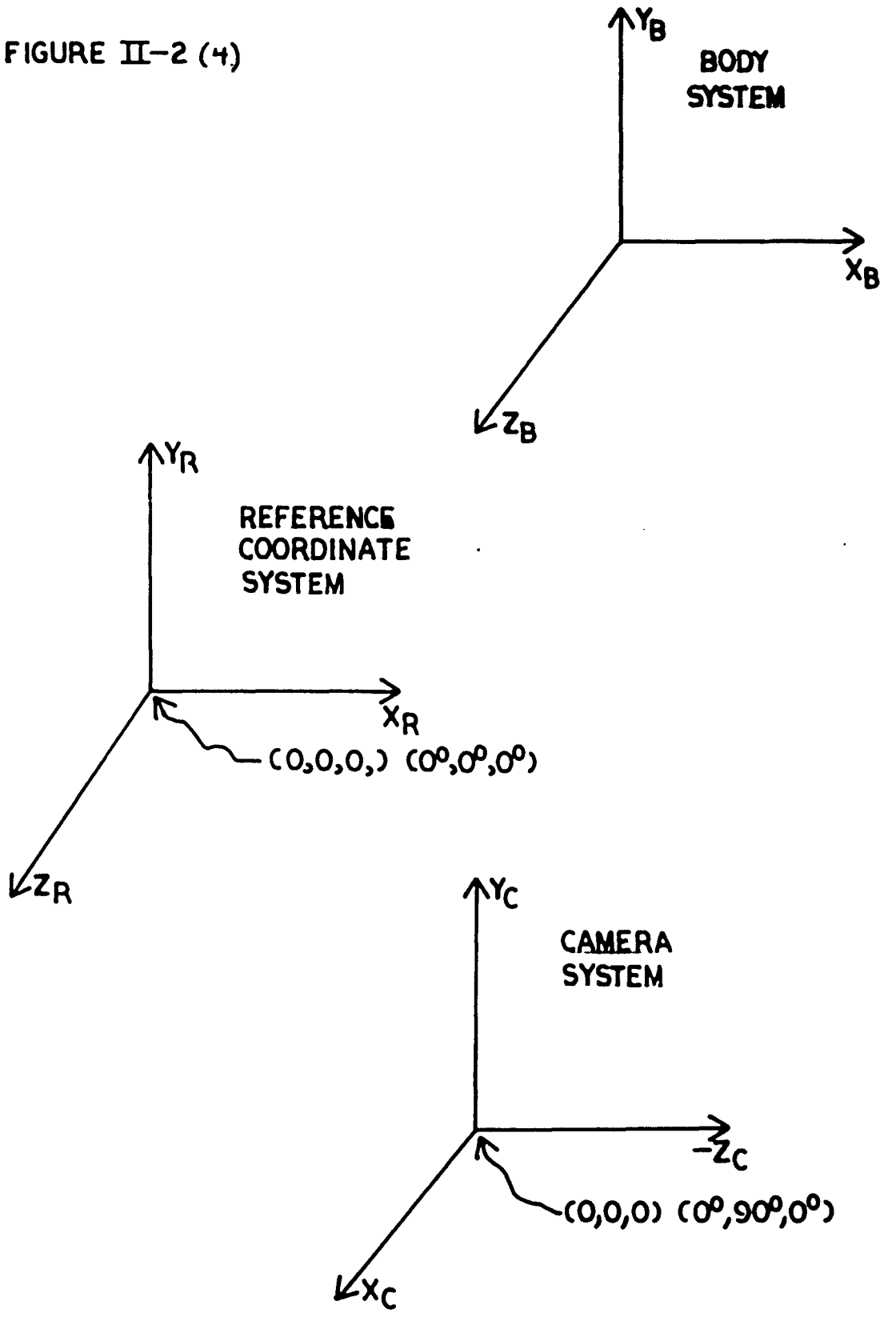
The camera coordinate system is the coordinate system defined for the conceptual camera with respect to the reference coordinate system. It has been established or standardized in most computerized camera systems that the camera's eye is pointed along the -Z axis of its coordinate system. This has been done so that the film plane upon which all bodies are projected (see Figure II-8) is always in the X - Y plane at the camera system and so that the film plane and the camera coordinate system are maintained as right-handed coordinate systems. (Thus, the reference and body coordinate systems must also be initialized as right-handed coordinate systems.) To visualize the camera's coordinate system with respect to the body and reference systems refer to Figure II-2.

To transform the coordinate points of a particular body from the body to the reference to the camera coordinate system, a three-dimensional coordinate transform must be realized.

C. Coordinate System Transforms

To perform a complete transform from one coordinate system to another, six independent variables must be taken into account. Of these variables, three make up the rotational transformation and three may up the translational transformation.

FIGURE II-2 (4)


$Y_B$
BODY SYSTEM
$X_B$
$Z_B$
$Y_R$
REFERENCE COORDINATE SYSTEM
$X_R$
(0,0,0,) (0°,0°,0°)
$Z_R$
$Y_C$
CAMERA SYSTEM
$-Z_C$
(0,0,0) (0°,90°,0°)
$X_C$

First, the rotational transform will be considered. If we are originally in the reference coordinate system and wish to transform a body's coordinates to the body system, and need only for the moment apply a rotational transformation, we must first define three Euler angles of rotation. These angles are $\theta$, $\phi$, and $\psi$ and are shown in Figure II-3. Looking first at $\psi$ rotation, we can visualize this rotation more clearly by looking straight down the Z axis as in Figure II-4. The $\psi$ rotation is defined as a counter-clockwise rotation around the Z axis. To transform a point in the (X, Y, Z) to the (X', Y', Z') coordinate system (or reference to body system), with a $\psi$ rotational displacement we must derive the three-dimensional matrix transform which performs this operation. To do this we can represent some point P in the X - Y plane by a vector OP as in Figure II-5.

We may represent the vector OP in terms of the (X, Y, Z) system as a linear combination of i, j, and k basis vectors and hence we may write,

$$OP = xi + yj + zk$$

We may also represent OP in terms of the (X', Y', Z') system with basis vectors $e_1$, $e_2$, and $e_3$.

$$OP = x'e_1, + y'e_2, + z'e_3$$

The unit basis vectors $e_1$, $e_2$, and $e_3$ in terms of i, j, k are found by using the rotational angle $\psi$. Thus, it can be seen by inspection that,

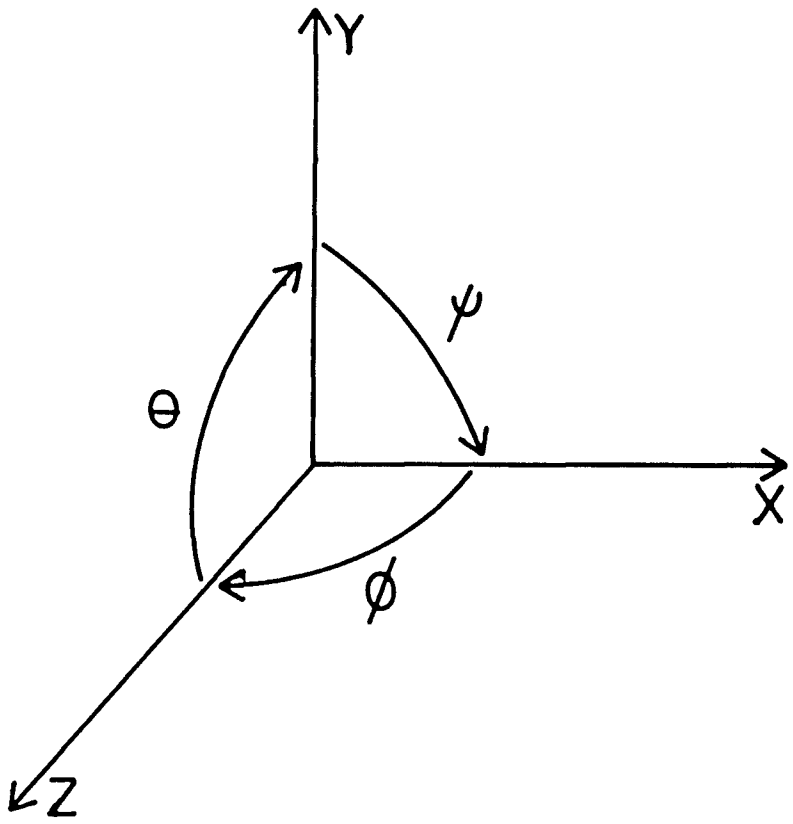



FIGURE II-3

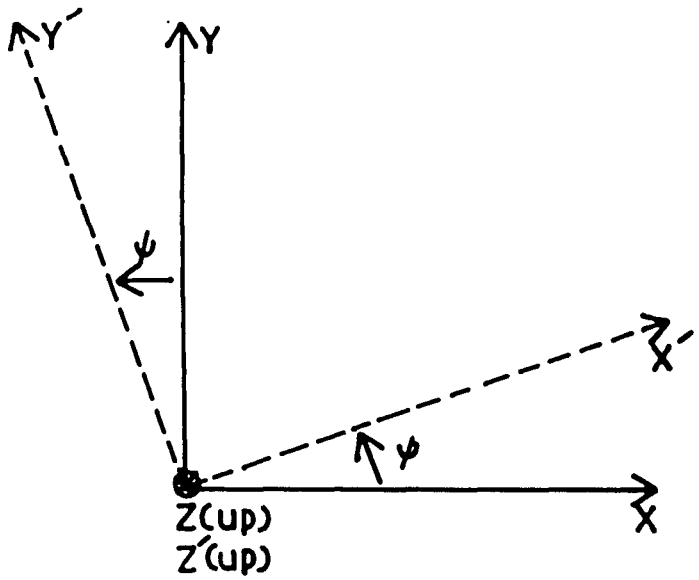



FIGURE II—4

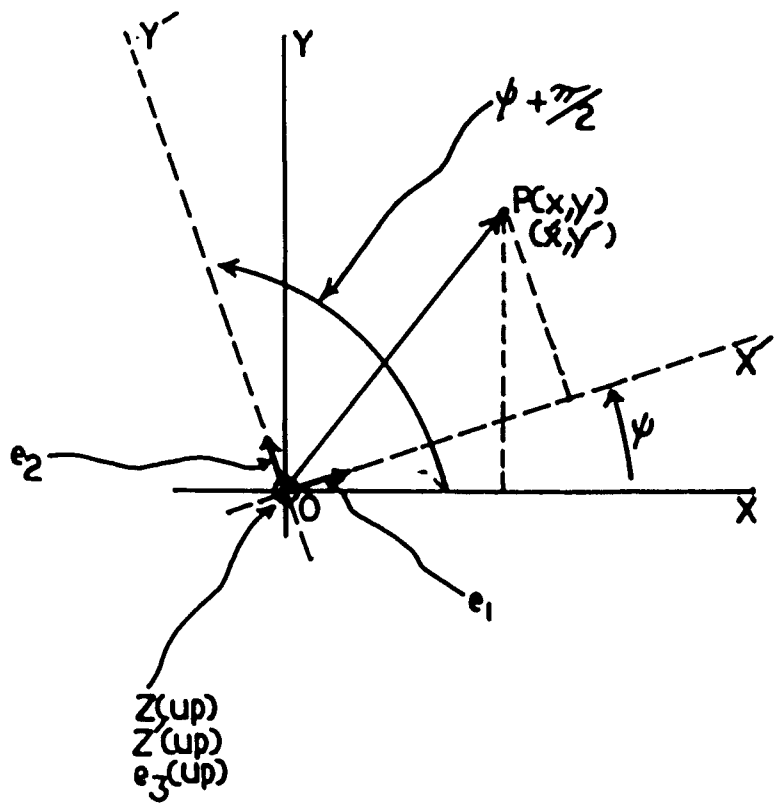



FIGURE II—5

$$e_1 = \cos\psi\ i + \sin\psi\ j$$

$$e_2 = \cos(\psi + \pi/2)i + \sin(\psi + \pi/2)j$$

$$= -\sin\psi\ i + \cos\psi\ j$$

$$e_3 = k$$

Therefore,

$$\vec{OP} = x'(\cos\psi\, i + \sin\psi\, j) + y'(-\sin\psi\, i + \cos\psi\, j) + z'k$$

$$= (x'\cos\psi - y'\sin\psi)i + (x'\sin\psi + y'\cos\psi)j + z'k$$

and since,

$$xi + yj + zk = (x'\cos\psi - y'\sin\psi)i + (x'\sin\psi + y'\cos\psi)j + z'k$$

The equations for the rotation of the axes are found by the equality of vectors,

$$x = x'\cos\psi - y'\sin\psi$$

$$y = x'\sin\psi + y'\cos\psi$$

$$z = z'$$

In matrix form, the above becomes

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [T] \times \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

where,

$$[T] = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

It is interesting and useful to note that,

$$\det [T] = \cos^2\psi + \sin^2\psi = 1$$

and thus,

$$[T]^{-1} = [T] \text{ transpose}$$

Therefore to transform a point in the(X Y Z) to the(X'Y'Z') system the following matrix multiplication is applied.

$$\begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

or,

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

but, if we follow the angle convention in Figure II-3 for we must replace the $\psi$ in the proof above with $-\psi$

Thus,

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos\psi & -\sin & 0 \\ \sin\psi & \cos & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

This is the $\psi$ rotational transform matrix.

The next rotational angle is $\Theta$ which is a counter clockwise rotation about the X axis as indicated in Figures II-3 and II-6.

A similar proof can be established for the transformation of a point in the (XYZ) system to the (X'Y'Z')

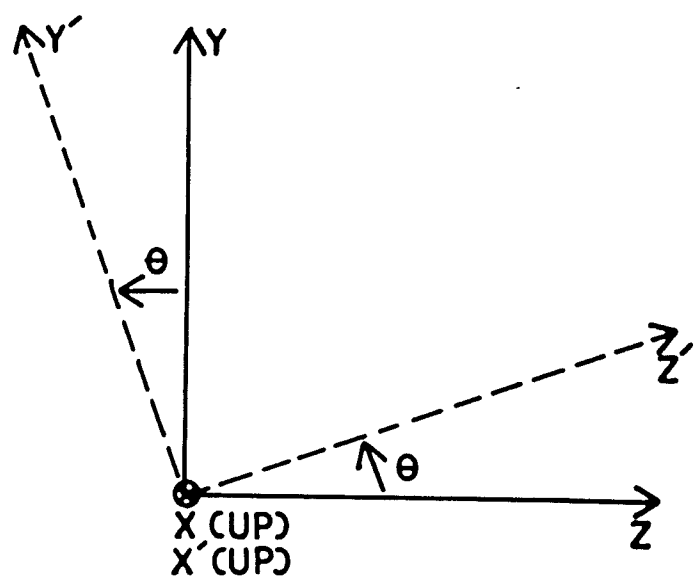



FIGURE II—6

system, with a $\theta$ rotational displacement. This transformation becomes,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}$$

Likewise for the final rotational angle $\phi$, in Figure II-3 and II-7,

$$\begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}$$

The complete rotational transform from the (XYZ) to the (X'Y'Z') system (or reference to body system) taking into consideration the rotational displacement of $\theta$, $\phi$, and $\psi$ altogether can be gleaned from a simple definition proved in Vector Calculus. The definition is,

$$[T_3] \cdot [T_2] \left( [T_1] \cdot (X) \right) = \left( [T_3] \cdot [T_2] \cdot [T_1] \right) \cdot (X)$$

A proof of the above is not included here but may be found in bibliography reference (1).

Thus a complete transform (X,Y,Z) to (X',Y,Z') is,

$$\begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

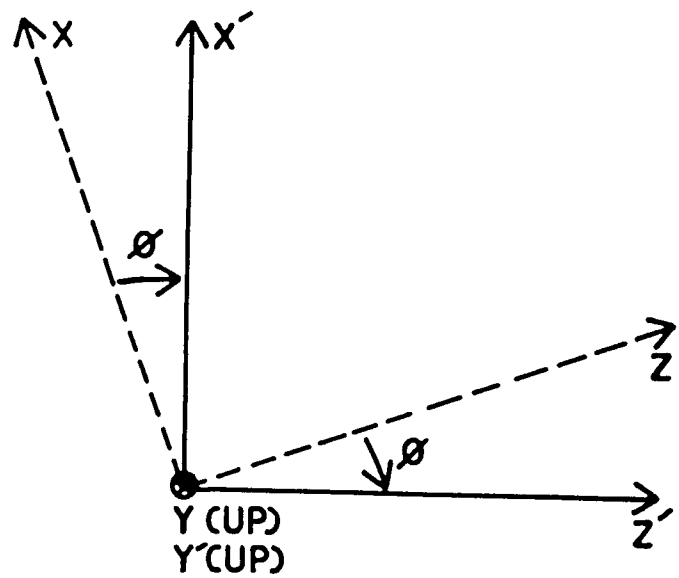



FIGURE II—7

From now on the three rotational matrices above will be labeled as the matrix, [TR2B], meaning the transform from the reference to body system. Thus,

$$[TR2B] \bullet \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}$$

It may be quite apparent now that what will be required in the CCPS is the rotational transform from the body to the reference system.

Thus,

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [TR2B]^{-1} \cdot \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}$$

but this is the same as,

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [TR2B]_{\text{transpose}} \cdot \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}$$

And thus we can define,

$$[TB2R] = [TR2B]_{\text{transpose}}$$

To make the transformation complete, the translational transform must be applied. If $X_{R2B}$, $Y_{R2B}$, and $Z_{R2B}$ are given the definition, as the translational displacement of a particular body from the reference to the body system,

and, if (X, Y, Z) and (X', Y', Z') are relabeled $(X_R, Y_R, Z_R)$ and $(X_B, Y_B, Z_B)$ respectively, the entire transform can be written as one equation:

$$\left[\left[TB2R\right] \cdot \begin{bmatrix} X_B \\ Y_B \\ Z_B \end{bmatrix}\right] + \begin{bmatrix} X_{R2B} \\ Y_{R2B} \\ Z_{R2B} \end{bmatrix} = \begin{bmatrix} X_R \\ Y_R \\ Z_R \end{bmatrix} \qquad (1.)$$

Any translational displacement of the body in $(X_B, Y_B, Z_B)$ results in an identical $(X_R. Y_R, Z_R)$ translation in the reference system given the initial displacement between both systems, thus, the above equation.

Next, it is appropriate to consider the transformation from the reference to the camera system, since our objective is to transform points of a body in its system to a pre-defined camera system. The rotational transform from reference system to camera system is developed in the identical manner as the $\left[TR2B\right]$ and thus, it will just be stated simply as,

$$\begin{bmatrix} \cos\Psi_C & -\sin\Psi_C & 0 \\ \sin\Psi_C & \cos\Psi_C & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_C & \sin\theta_C \\ 0 & -\sin\theta_C & \cos\theta_C \end{bmatrix} \cdot \begin{bmatrix} \cos\phi_C & 0 & \sin\phi_C \\ 0 & 1 & 0 \\ -\sin\phi_C & 0 & \cos\phi_C \end{bmatrix}$$

where $\Psi_C$, $\theta_C$, and $\phi_C$ are the rotational angles of orientation for the camera system.

The entire transform from the reference coordinate system to the camera coordinate system including the translational

transform is developed as before and can be stated as,

$$[TR2C] \cdot \left( \begin{bmatrix} X_R \\ Y_R \\ Z_R \end{bmatrix} - \begin{bmatrix} X_{R2C} \\ Y_{R2C} \\ Z_{R2C} \end{bmatrix} \right) = \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} \qquad (2.)$$

where again $(X_{R2C}, Y_{R2C}, Z_{R2C})$ is the translational displacement of the camera coordinate system with reference to the reference system.

Thus, if a body is defined in its body coordinate system it can be transformed point by point into the predefined camera coordinate system by the application of the above equations. These equations may be combined into one equation which entirely describes the transformation from $(X_B, Y_B, Z_B)$ to $(X_C, Y_C, Z_C)$ and it may be written as,

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = [TR2C] \cdot \left( \left( [TB2R] \cdot \begin{bmatrix} X_B \\ Y_B \\ Z_B \end{bmatrix} \right) + \begin{bmatrix} X_{R2B} \\ Y_{R2B} \\ Z_{R2B} \end{bmatrix} - \begin{bmatrix} X_{R2C} \\ Y_{R2C} \\ Z_{R2C} \end{bmatrix} \right) \qquad (3.)$$

D. The Theory of Perspective Projection

Once the coordinates of a particular body are known in the camera coordinate system, they must then be projected into a two-dimensional film plane or screen. There are several methods of accomplishing this; one method may be developed

by looking at Figure II-8. In Figure II-8, it is desired to project point P onto the $X_p$ - $Y_p$ plane and thus into point P'. The camera "eue" is located at Zc=o. and points along the -Zc axis. The $X_p$ - $Y_p$ plane is the film plane or screen. (Xc, Yc, Zc) is one point of a particular body which has been transformed to the camera coordinate system. $\theta_{FOV}$ is the field of view of the camera. In the CCPS, $X_{p(max)}$ will equal $Y_{p(max)}$, the X and Y field of view angles will be equal (this is more for a matter of convenience than any other reason).

Note that the orientation of the film plane is a righthanded coordinate system. Thus, after the perspective projection, no coordinates need further transformation or scaling on the film plane.

The perspective image of the lines of a body can be generated easily by transforming only their end points and drawing a line between the transformed end points in the proper order.

To visualize the perspective projection of the end points see Figure II-9. This figure is the same as Figure II-8, only looking down the $Y_c$ axis. $X_c$, $Y_c$, and $Z_c$ are known for point P, $0_{FOV}$ is known, and so is b, the film plane size. To find to $X_p$ and $Y_p$, the law of similar triangles can be applied.

Thus, the ratio of line segments in triangles OQP and OQ'P' are,

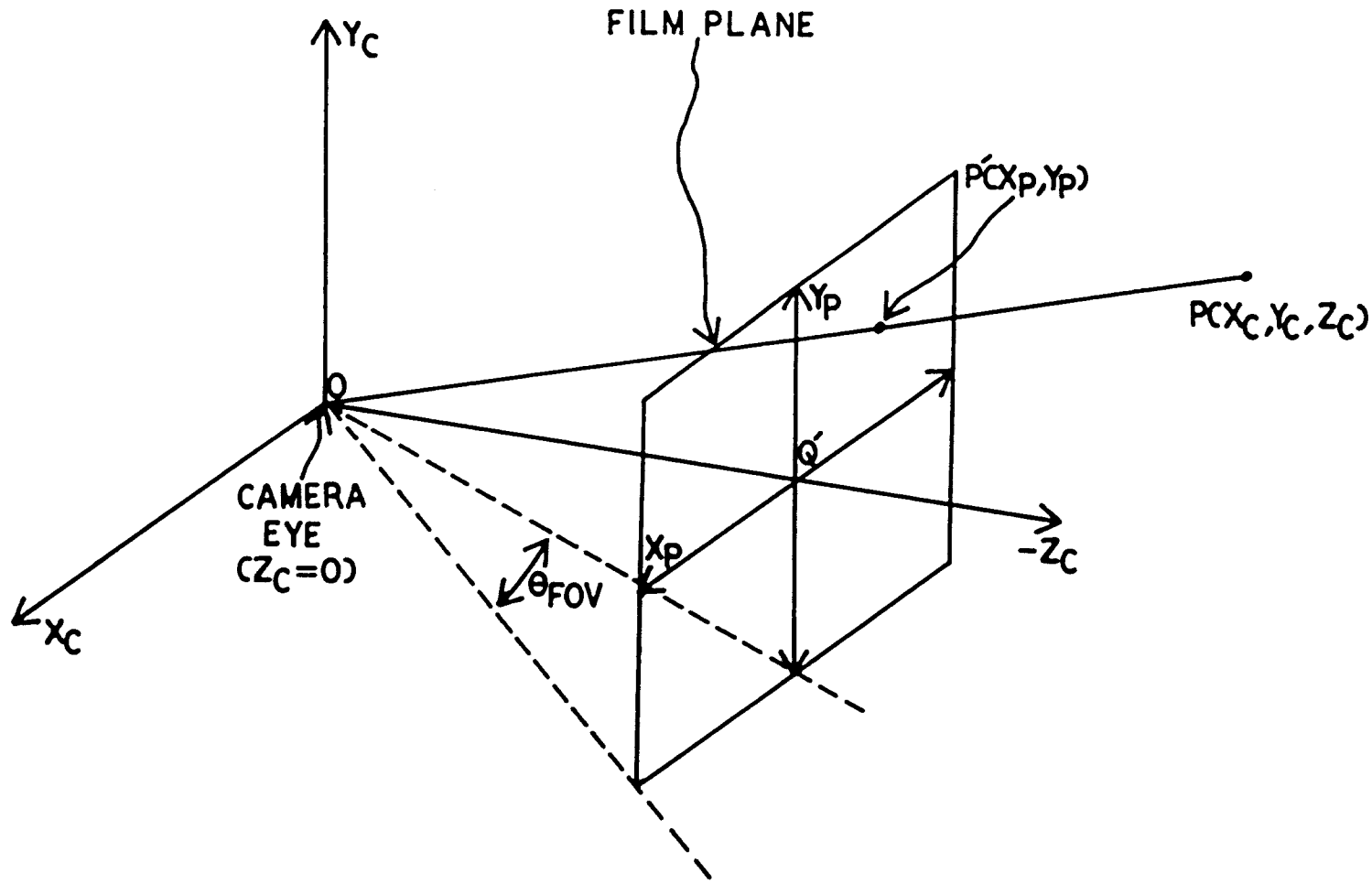

FILM PLANE
$Y_C$
$P'(X_P, Y_P)$
$Y_P$
$P(X_C, Y_C, Z_C)$
O
Q'
CAMERA
EYE
($Z_C$=0)
$X_P$
$-Z_C$
$\theta_{FOV}$
$X_C$

FIGURE II–8 (4.)

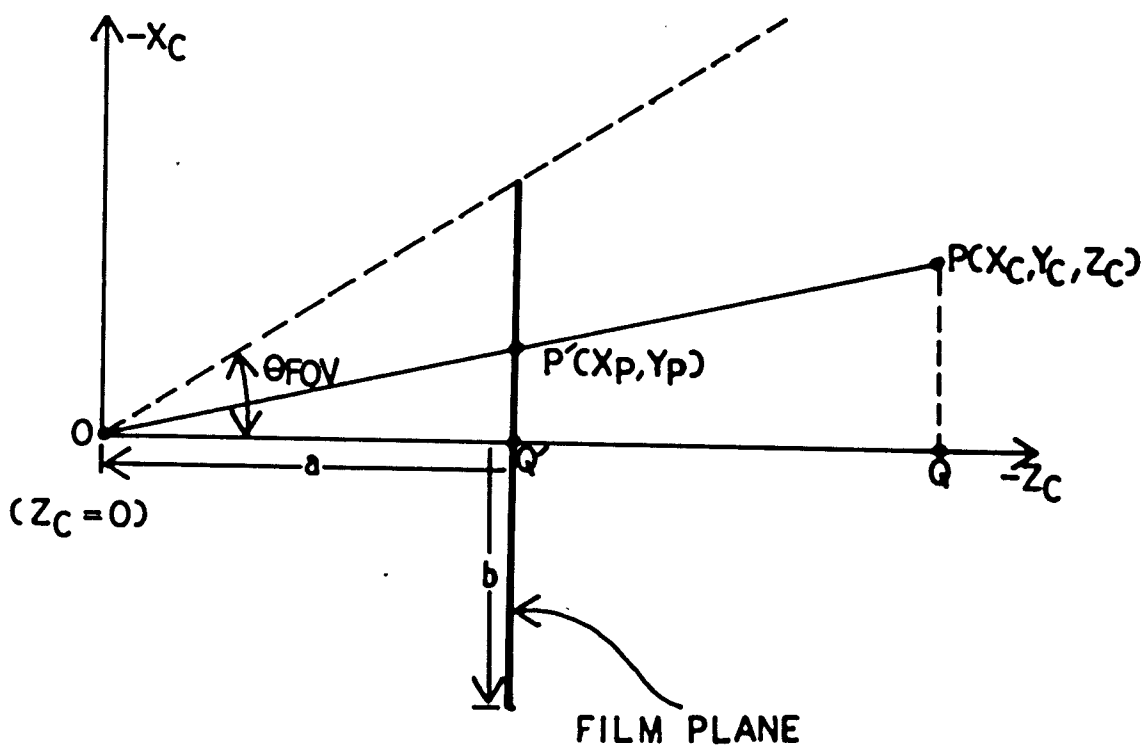



FIGURE II-9 (4.)

$$\frac{OQ}{QP} = \frac{OQ'}{OP'}$$

or,

$$\frac{Zc}{Xc} = \frac{a}{-Xp}$$

$$\frac{Xp}{} = \frac{aXc}{-Zc} \qquad (Xc \text{ and } -Zc \text{ are known})$$

but, $\tan(0_{FOV}) = \frac{b}{a}$ ($0_{FOV}$ and b are known)

$$a = \frac{b}{\tan(0_{FOV})}$$

thus,

$$Xp = \frac{bXc}{-Zc\tan(0_{FOV})} \qquad (1.)$$

By an identical argument it can be shown that,

$$yp = \frac{byc}{-Zc\tan(0_{FOV})} \qquad (2.)$$

It may be noted that due to the way the camera coordinate system and the film plane are oriented, a couple of projection limitations are present. One limitation is that the projection of a point is not possible if $Zc \not< 0$. This limitation is rather apparent; the camera cannot see an

object which lies behind it. The second limitation is that yp and xp must be /b/. In other words, the projected point must lie on the film plane. This leads to the consiceration of a clipping routine to protect against screen (film), plane overrun. This clipping algorithm is easiest accomplished in two-dimensions on the film plane itself. This algorithm is discussed in detail later.

## III. The Internal Structure of the Conceptual Camera Programming System

### A. Introduction

This section will deal exclusively with the internal workings of the CCPS. It will describe the program flow, data base, and subroutines necessary to implement this system.

The principle subroutines that will be outlined in this section are:

(1.) START - the CCPS initialization subroutine

(2.) UPDTE - the CCPS data base bookkeeper subroutine

(3.) TRANS - the CCPS coordinate system transform subroutine

(4.) IMAGE - the actual film image producing subroutine

(5.) CLIP1 - the three dimensional clipping algorithm

(6.) SEQ - the body connection scheme storage subroutine

(7.) Body motion subroutines -

| | | |
|---|---|---|
| XTRAN | YTRAN | ZTRAN |
| XTR | YTR | ZTR |
| RTX2Z | RTY2X | RTY2Z |
| X2Z | Y2X | Y2Z |

The CCPS procedural file will also be explained along with CCPS segmented programming technique.

B. The CCPS Program Flow

The program flow format can be realized by viewing Figure III-1.

The numbers referenced below refer to the numbers next to each box in Figure III-1.

1. CALL START - This subroutine is only called once in the CCPS. It initializes all the important system parameters. These parameters are the number of bodies, in the initial body and camera orientations, the body center definitions, the field of view angle, the number of lines in each body, and the frame size. It also initializes the data base and sets the frame counter to one.

2. In this step, the point connection scheme is defined totally by the user in the array MSEQ. This array is then stored in the data base via the subroutine SEQ.

3. At this point, all the body motions which are to be performed in this animation are now defined. Only one body motion of the six types may be defined for each body. This facet of the CCPS is explained fully later.

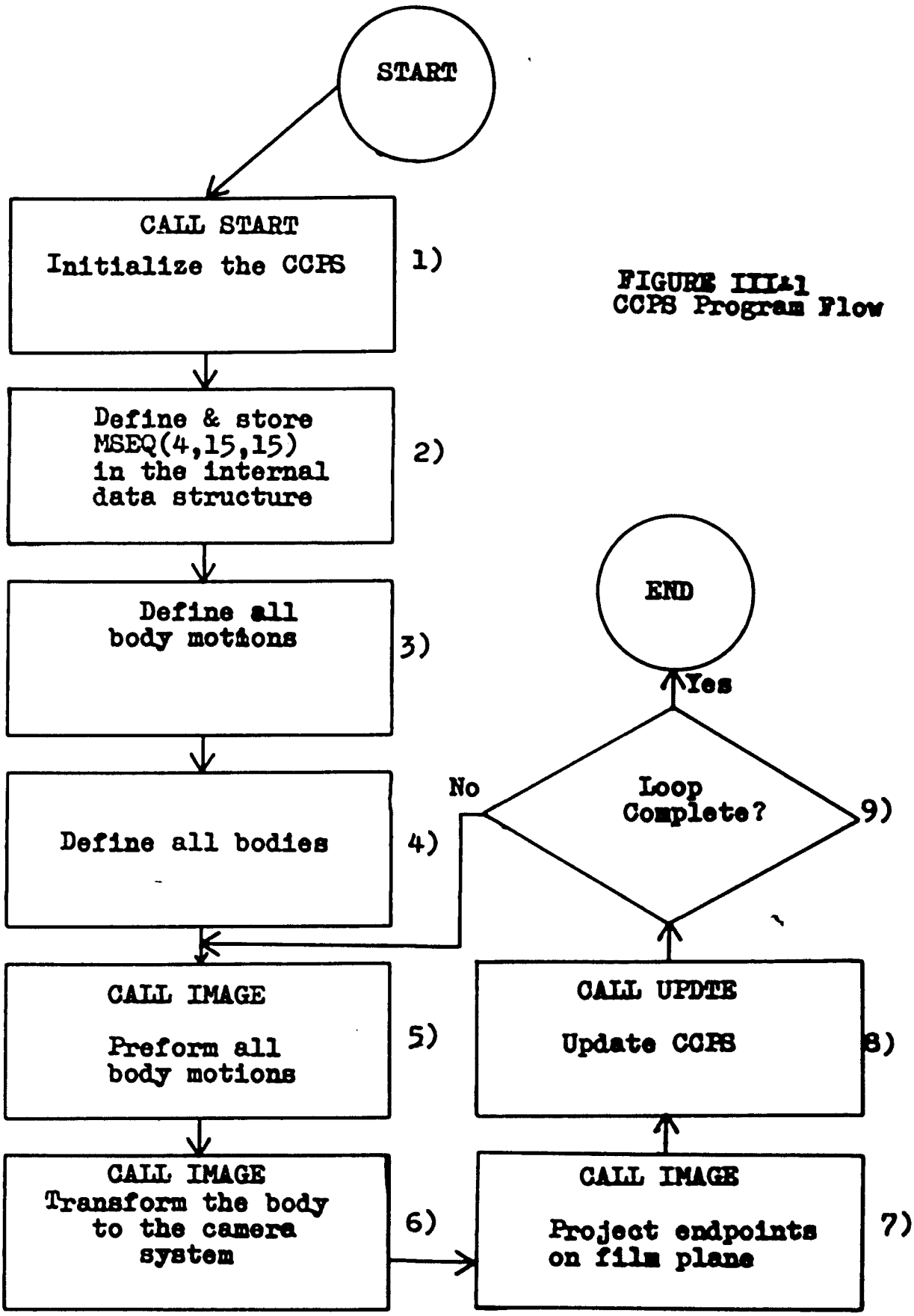



FIGURE III.1
CCPS Program Flow

4. At this point, each body is now defined in its own body system. This is accomplished by defining all its end points.

5. CALL IMAGE - At this point, all body motions are now performed upon the bodies that have been previously defined.

6. In this step, all body coordinates are transformed to camera coordinates within the camera coordinate system.

7. In this step, the resulting camera coordinates are now projected onto the film plane and clipped.

8. CALL UPDTE - The frame counter is incremented by one and all flags are checked and/or set.

9. In this step, the loop is checked for completion. This is a simple user defined DO loop in Fortran IV (ie DO 1 I = 1, N, where N is the number of frames). This step defines the number of frames in the animation process, and controls the looping format of the CCPS.

C. Data Base Structure

This section details the data foundation for the CCPS. Internal data structure processes will be described so as to form a foundation for the explanation of other CCPS routines involving the data base.

1) COORDS

The first element in the CCPS data base is an array called COORDS. As can probably be gleaned from

the name, this array contains all the information pertaining the coordinates of the camera system and of the various body systems. To better understand what this data base array contains refer to Figure III-2.

In Figure III-2, COORDS is a five by thirteen dimensional array. The rows of the array reference the camera and each of the bodies. The columns of the array reference data concerning the camera and each of the bodies. The data referred to by the first five columns are,

1.) Flag - an indicator that is set or reset telling the CCPS whether the camera or any body is in motion or not. (0-no motion, 1-motion in progress.)

2.) X - the initial X displacement of the camera of the bodies.

3.) Δ X - the incremental X displacement of the camera or the bodies.

4.) ST - the starting frame of the incremental X displacement.

5.) ED - the ending frame of the incremental X displacement.

Displacements are referrenced to the reference coordinate System. The above definitions are identical for the other

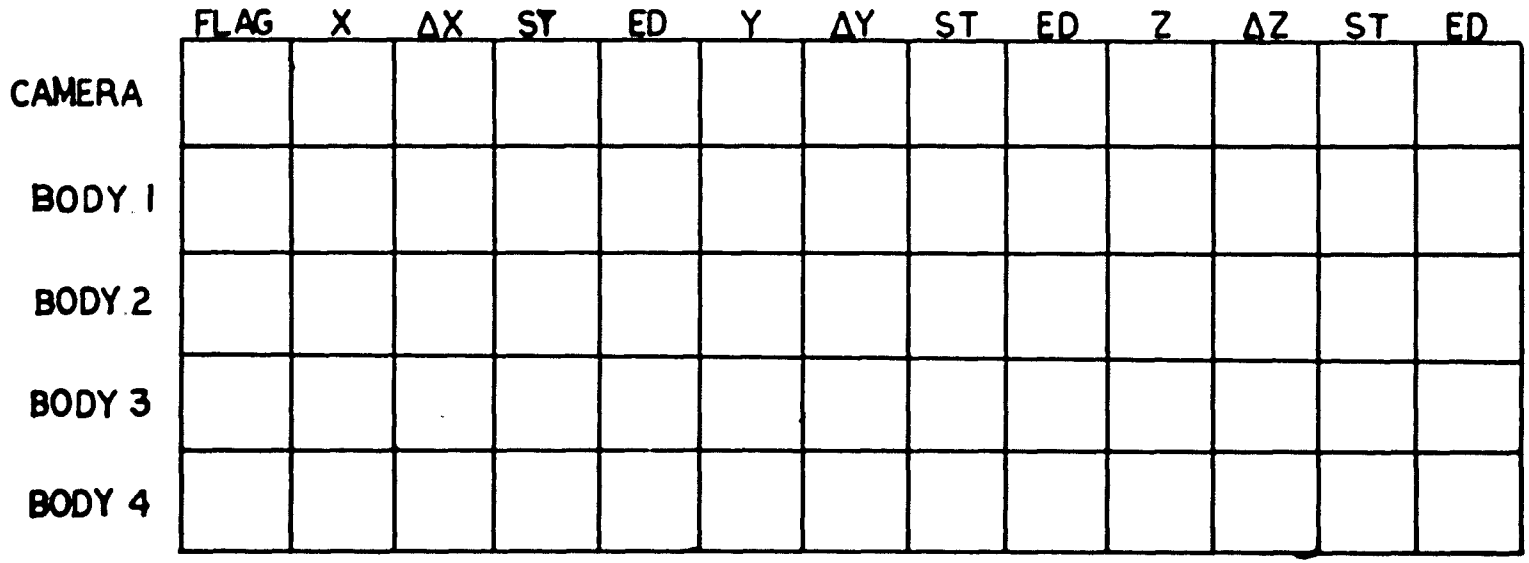

FLAG
X
ΔX
ST
ED
Y
ΔY
ST
ED
Z
ΔZ
ST
ED
CAMERA
BODY 1
BODY 2
BODY 3
BODY 4

FIGURE III-2

eight columns. As can be seen, storage has been alloted for four bodies and a camera.

If there are zeros in the Flag, D, ST, and ED columns for X, Y, or Z then no changes are specified for that parameter at the present time. If these parameters are non-zero, then a change is indicated for that specified coordinate at some time in the animation process.

2) ANGLE

The second element in the CCPS data base is an array called ANGLE. This array is identical in size to the array COORDS, but instead contains information concerning the angular displacements of the various bodies and the camera. To see how ANGLE is constructed, look at Figure III-3.

The rows of the array ANGLE reference the four bodies and the camera. The columns of the array reference information concerning the four bodies and the camera. The information referenced by the first five columns is:

1.) FLAG - an indicator, that is set or reset, telling the CCPS whether the camera or any body is in motion or not (0-no motion, 1-motion in progress).

2.) $\theta$ - the initial $\theta$ angular displacement of the camera or any of the bodies.

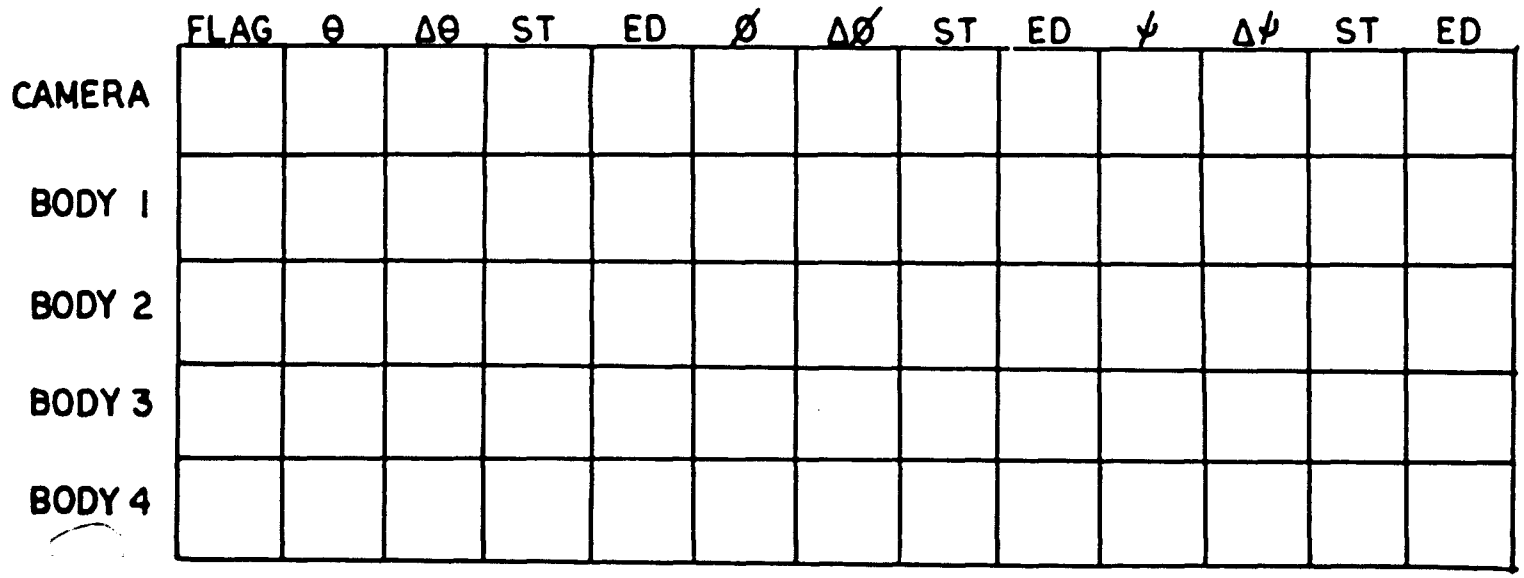

FLAG
θ
Δθ
ST
ED
ø
Δø
ST
ED
ψ
Δψ
ST
ED
CAMERA
BODY 1
BODY 2
BODY 3
BODY 4

FIGURE III-3

3.) $\Delta\theta$ - the incremental O angular displacement of the camera or any of the bodies of each frame.

4.) ST - the starting frame of the O angular displacement.

5.) ED - the ending frame of the O angular displacement.

Again, all displacements are referenced to the reference coordinate system, and the above definitions also hold true for remaining columns. If there are any zeros in i, 3, 4, or 5 above, there is at present no motion indicated. If there are any non-zero quantities in 1, 3, 4, or 5 then motion is indicated at some time.

As an example of what occurs in the above two arrays assume the following data exists in the both arrays:

COORDS (2, 6) = 10. COORDS (2, 7) = 1.

COORDS (2, 8) = 2. COORDS (2, 9) = 11.

COORDS (2, 1) = 1.

ANGLE (2, 2) = 30. ANGLE (2, 3) = 5.

ANGLE (2, 4) = 2. ANGLE (2, 5) = 4.

ANGLE (2, 1) = 1.

First, looking at the COORDS data we see COORDS (2,1) = 1. This indicates that some motion is presently in progress for body 1. COORDS (2,6) = 10., indicates that body 1 is displaced initially in the Y direction by ten units. Looking

one column over, COORDS (2, 7) = 1., means that body 1 is to be displaced by one unit per frame starting at frame, COORDS (2, 8) = 2., and ending at frame COORDS (2, 9) = 11.. Thus the total Y displacement from frames two to eleven is ten units. The units are user defined in subroutine START to be discussed later.

The ANGLE data can be analyzed in the following manner. ANGLE (2, 1) = 1., tells us that again body 1 is presently in some angular motion. ANGLE (2, 2) = 30., indicates that initially $\theta$ is equal to thirty degrees. ANGLE (2, 3) tells us that $\theta$ is rotated five degrees per frame starting at thirty degrees. The rotation is to start at frame ANGLE (2, 4) = 2. and end at frame ANGLE (2, 5) = 4.. Thus a total $\theta$ rotation of fifteen degrees occurs in three frames.

3) TRIG

In order to reduce the number of computations performed in the CCPS an array named TRIG was created. This array simply contains the sine and cosine values, for the camera and the four bodies, of $\theta$, $\Delta\theta$, $\phi$, $\Delta\phi$, $\psi$, and $\Delta\psi$. The array is drawn in Figure III-4.

Thus, TRIG (6, 3) is the storage location for the cosine of the initial $\theta$ rotational angle for body 2. Likewise, TRIG (9, 6) is the sine of the incremental $\psi$ rotation of body 4. If any

| | | $\theta$ | $\Delta\theta$ | $\phi$ | $\Delta\phi$ | $\psi$ | $\Delta\psi$ |
|---|---|---|---|---|---|---|---|
| CAMERA | SIN | | | | | | |
| | COS | | | | | | |
| BODY 1 | SIN | | | | | | |
| | COS | | | | | | |
| BODY 2 | SIN | | | | | | |
| | COS | | | | | | |
| BODY 3 | SIN | | | | | | |
| | COS | | | | | | |
| BODY 4 | SIN | | | | | | |
| | COS | | | | | | |

FIGURE III-4

displacement is rotationally zero the sine values are, of course, one and the cosine values are zero.

The reason for the existence of the TRIG array is to perform all trigonometric calculations only once, thus, increasing the CCPS speed and efficiency.

4) TB2R

The TB2R array is a three dimensional array containing the matrix for transforming a particular body from its body system to the reference system. This transform is discussed in section II-C, Coordinate System Transforms. TB2R is configured as in Figure III-5.

As seen the variable ,I, defines which body each array transforms. Each array is a three by three matrix since it is a three dimensional transformation. The values for TB2R are obtained from the array TRIG.

5) TR2C

Array TR2C as illustrated in Figure III-6, is the transform from the reference system to the camera system as discussed in Section II-C, Coordinate System Transforms. There exists only one

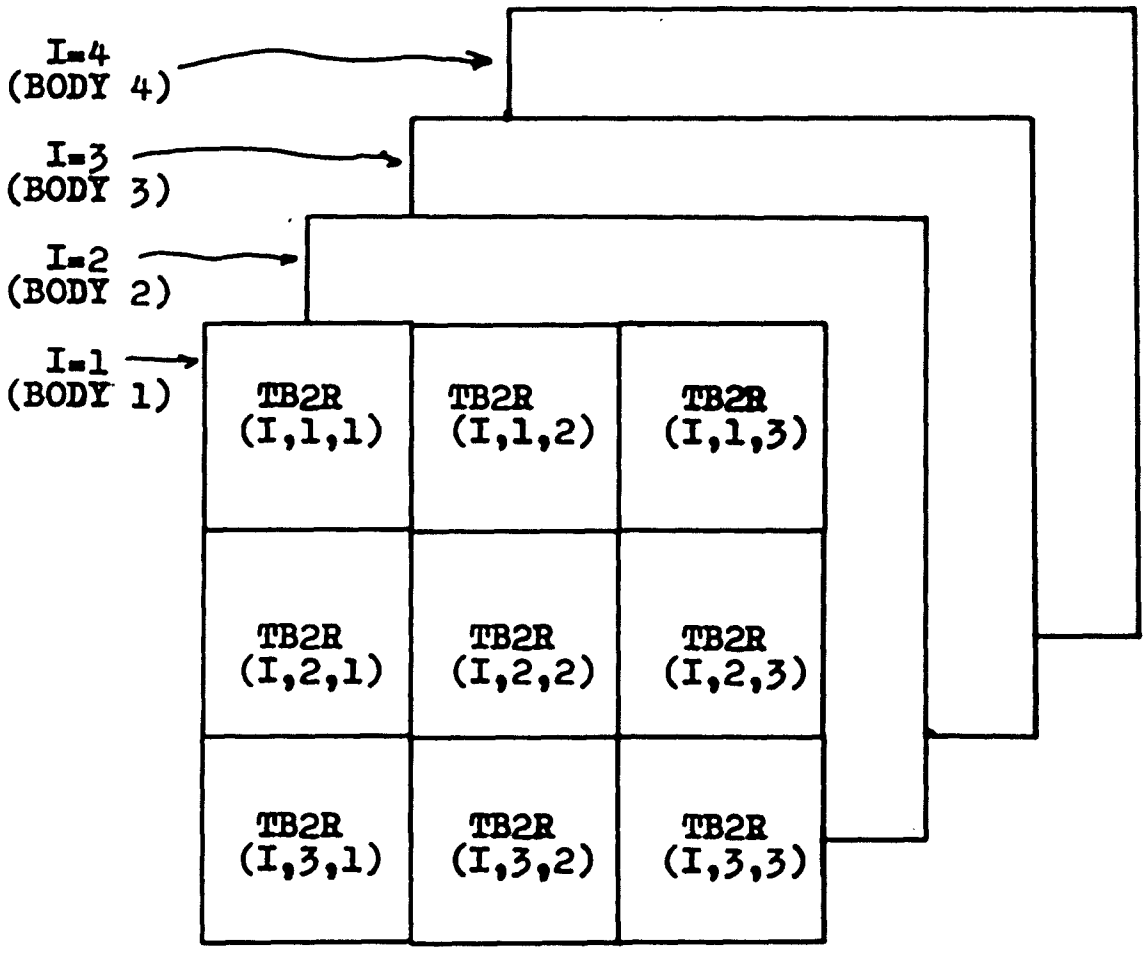

I=4
(BODY 4)
I=3
(BODY 3)
I=2
(BODY 2)
I=1
(BODY 1)
TB2R
(I,1,1)
TB2R
(I,1,2)
TB2R
(I,1,3)
TB2R
(I,2,1)
TB2R
(I,2,2)
TB2R
(I,2,3)
TB2R
(I,3,1)
TB2R
(I,3,2)
TB2R
(I,3,3)

FIGURE III-5

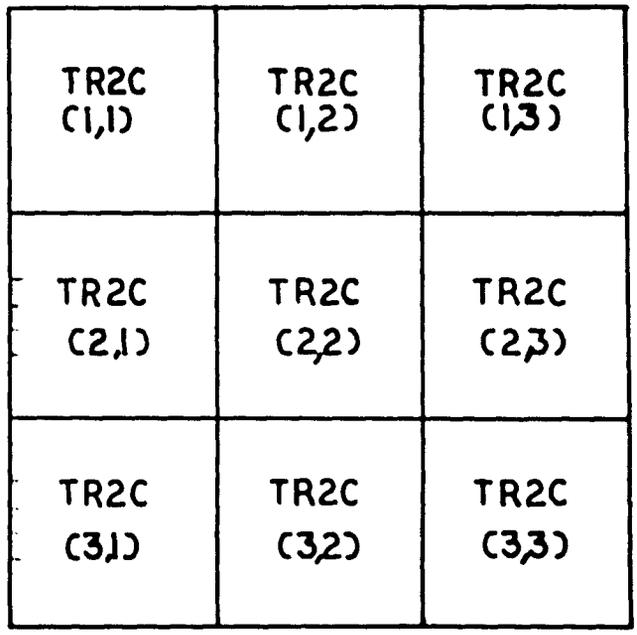

TR2C (1,1)
TR2C (1,2)
TR2C (1,3)
TR2C (2,1)
TR2C (2,2)
TR2C (2,3)
TR2C (3,1)
TR2C (3,2)
TR2C (3,3)

FIGURE III-6

camera in this system and thus only one three by three matrix transform. Again the values for TR2C are taken from the array TRIG.

D) Subroutine START

This subroutine is the initializing and data entry routine for the CCPS. Figure III-7 shows the program flow for the START subroutine. The programmer call for the subroutine is:

CALL START

No parameters need be passed to the subroutine. All data entered via START is stored in a common data block called CCPS which is discussed in full later.

The following numbered explanations correspond to the numbered blocks in the START program flow in Figure III-7.

1.) This step initializes the COORDS, ANGLE, and TRIG arrays by setting the values of COORDS and ANGLE to all zeros. Since all angles in the system are initially zero, the odd rows (sines) of TRIG will be set to zero and the even rows (cosines) will be set to one.

2.) In this step, the LSEQ array is initialized by setting all of its values to zero. The LSEQ array is explained in full later.

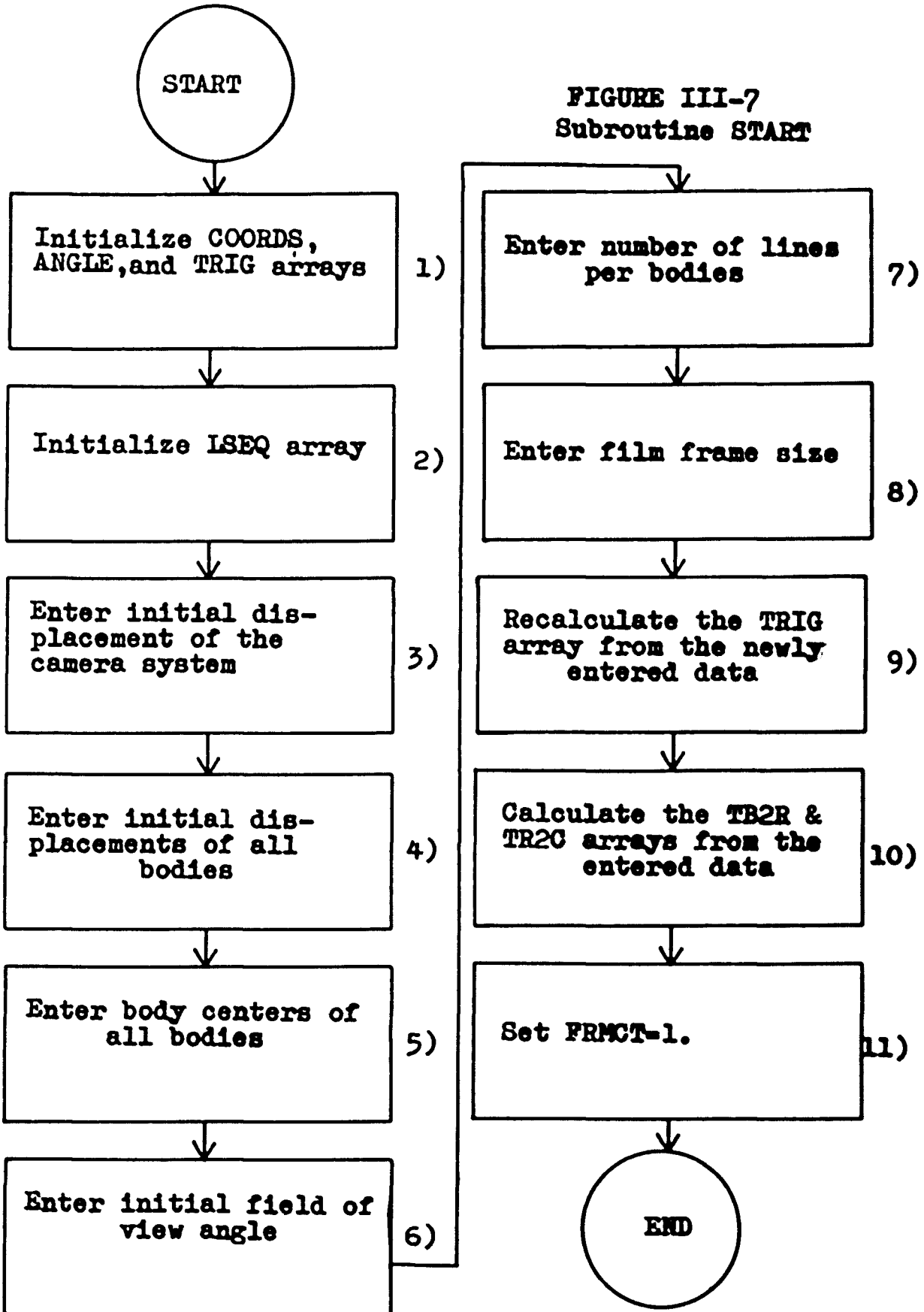



FIGURE III-7
Subroutine START

3.) At this point, the initial displacements of the camera coordinate system are interactively inputed and stored in COORDS and ANGLE. These parameters are the $X_C$, $Y_C$, $Z_C$, $\Theta_C$, $\phi_C$, and $\psi_C$ displacements as discussed in Section II-C. As discussed in Section II-C the values for $X_C$, $Y_C$, $Z_C$, $\Theta_C$, $\phi_C$, and $\psi_C$ will normally be set to (0,0,0) and (0,90,0) respectively. This gives the camera no translational displacement, thus placing the camera at the origin, but rotates X axis into the S axis ninety degrees, setting a right handed coordinate system, and thus, the proper orientation for the projection of the bodies onto the film plane.

4.) In this step the initial displacements of all body coordinate systems are interactively inputed and stored in COORDS and ANGLE.
Again the parameters inputed for each body are $X_B$, $Y_B$, $Z_B$, $\Theta_B$, $\phi_B$, and $\psi_B$. These parameters are displacements referenced to the reference coordinate system.

5.) At this level, the body center or the point of rotation for each body is entered. This point is the physical center of each body in its own body coordinate system or a point at which the

user desires the body to be rotated about (discussed later), again defined in its own body system.

6.) The camera's field of view is entered here. If, for example, a field of view of $30^{o}$ is desired then $\Theta_{fov}$ in Figure II-8 is $30^{o}$. The TANFOV value in the projection equation in Section II-D then becomes, TANFOV = $\tan(30^{o}) = 0.577$.

7.) In this step, the number of lines for each body must be defined.

8.) At this point the frame size is defined by the user. Normally the size of the film plane is set to two and thus b in Figure III-8 is one. This is mostly a matter of convenience. (Note that the film plane is always square.)

9.) At this time, the new values for the TRIG array are calculated from the new data entered into the ANGLE array in steps 3 and 4.

10.) Now the transformational matrices TB2R and TR2C are found and thus initialized from the data found in the TRIG array as discussed in Section II-C.

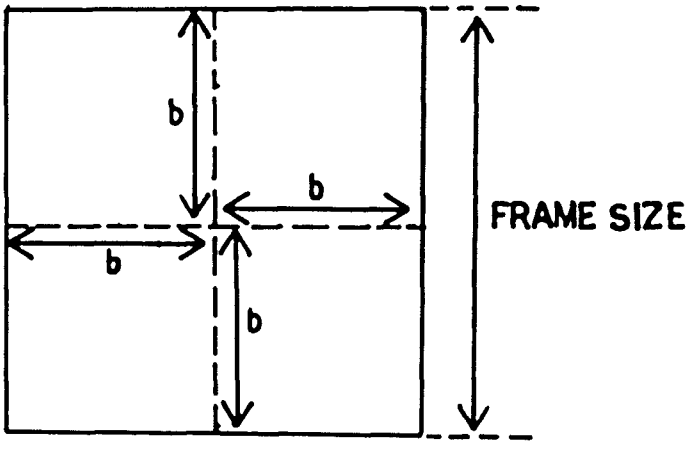



FIGURE III -8

11.) Finally, the frame counter, FRMCT, is set equal to one in preparation for the drawing of the first frame.

E) Subroutine UPDTE

This subroutine is the "bookkeeper" for the CCPS. Figure III-9 depicts the program flow for the routine. The programmer call for the subroutine is:

CALL UPDTE

The following numbered explanations correspond to the numbered boxes in Figure III-9.

1.) The subroutine first checks the data structure to see if any motion is ending in the present frame. This is accomplished by checking the END column for each body in the COORDS and ANGLE data arrays.

2.) If a motion is ending, the appropriate flag is set from a one to a zero in the FLAG column in the COORDS and ANGLE data arrays.

3.) In this step, the frame counter is incremented by 1. . Since CALL UPDTE is at the bottom of the loop procedure, all processing for the present frame is completed with the completion of Step 2. Thus, the present frame incremented by one in preparation is for the next frame.

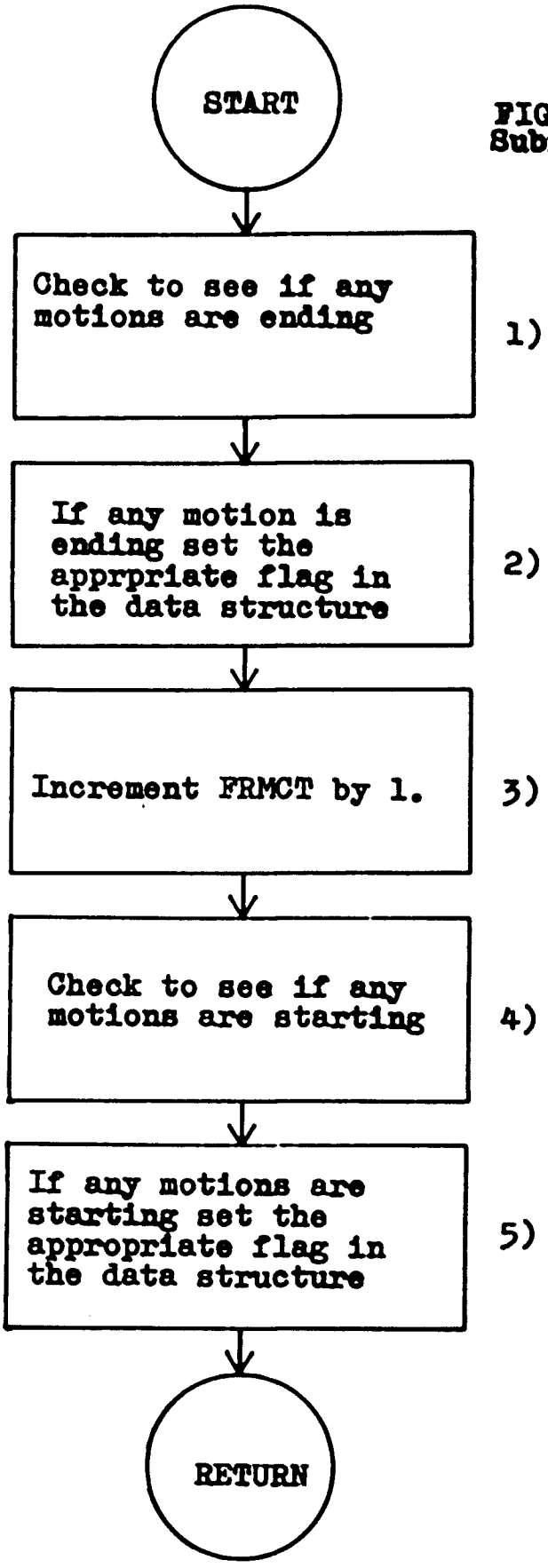



FIGURE III-9
Subroutine UPDTE

4.) In this step, the subroutine checks to see if any motion is starting in the upcoming frame. As before, this is accomplished by checking the ST column in the COORDS and ANGLE data arrays.

5.) If any motion is starting in the preceding step the appropriate flag is set from a zero to a one in the FLAG column of the COORDS and ANGLE data arrays.

When in the $i^{th}$ iteration of the loop in Figure III-1, subroutine UPDTE is called after the $i^{th}$ frame is completed. The system is then completely updated and set for the drawing of the next frame. Program control is then returned to the loop and the drawing of $i^{th} + 1$ frame then precedes.

F) Subroutine TRANS

This subroutine determines the transformational matrices TB2R and TR2C, as described in Section II-C, and Sections III-C-4 and III-C-5. This subroutine is invisible to the general CCPS user. It is called in subroutines START, to initialize both matrices, and in IMAGE to determine each matrix after a body rotation or translation. The subroutine call is:

CALL TRANS (IBODY)

where IBODY is the body for which TB2R is being determined, i.e. TB2R (IBODY, 3, 3). IBODY equal to zero determines the matrix TR2C, i.e. TR2C (3,3).

G) Subroutine IMAGE

The discussion in this section deals with boxes 5, 6, and 7 in Figure III-1 which draw the three-dimensional bodies on the film screen. The program flow for the IMAGE subroutine is outlined in Figure III-10. The programmer's subroutine call for drawing the bodies is:

CALL IMAGE (X, Y, Z, NCOOR, IBODY)

where X, Y, and Z are the coordinates for the particular body to be drawn, NCOOR is number of coordinates defining that body, and IBODY is the body system's designated number.

The numbered discussions that follow refer to the numbers beside the boxes in Figure III-10.

1.) In this first step, the subroutine decides whether any body motion has been specified to occur. This is accomplished by checking the flags in COORDS and ANGLE. If no motion is specified for the body in this frame, the subroutine returns program control to the loop process. If motion is to occur the program continues onto Step 2.

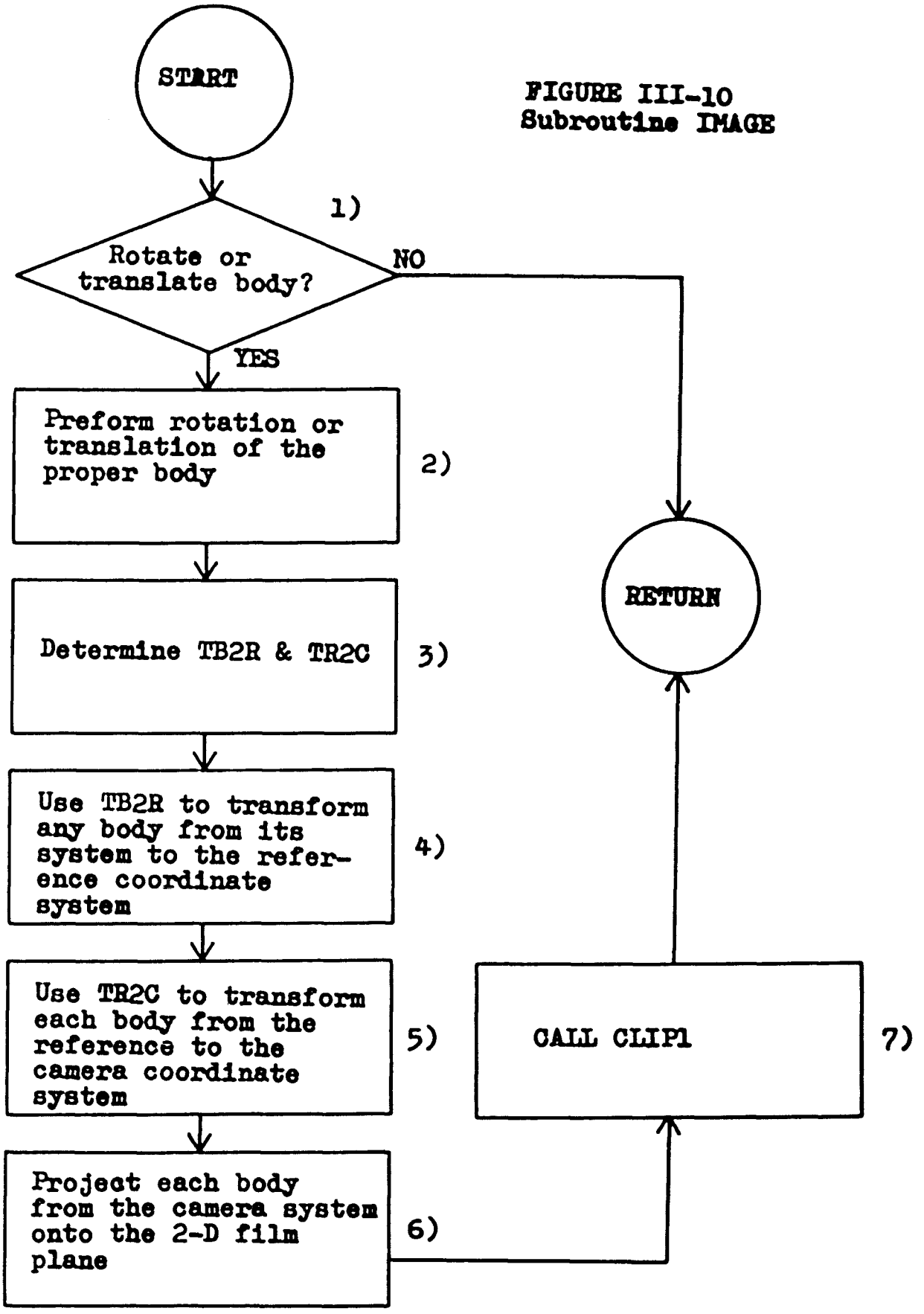



FIGURE III-10
Subroutine IMAGE

2.) Since motion is indicated for the particular body under consideration, the subroutine determines what motion is desired, rotational or translational, and in what direction. The body is then rotated or translated in its own body system by a program call to XTR, YTR, ZTR, X2Z, Y2X, and/or Y2Z depending on the motion desired. These subroutines will be discussed in detail later in Sections III-J-7 thru III-J-12.

3.) When the body has been rotated, and the new coordinates determined, TB2R and TR2C must then be recalculated to determine the transform from the body to camera coordinate system.

4.) TB2R is then employed to transform the coordinates from the body coordinate system to the reference coordinate system as done by Equation (1.), Section II-C.

5.) The coordinates are then transformed from the reference to the camera coordinate system via the use of TR2C as in Equation (2.), Section II-C. The total transform being Equation (3.) of the same section.

6.) In this step, once the coordinates lie in the camera's coordinate system they may be projected

onto the film plane via the projection equations (1.) and (2.) of Section II-D.

7.) In this step, the subroutine CLIP1 is called. This subroutine clips the projected two-dimensional body coordinates to the film plane boundaries, so that no screen wrap around occurs. If the body lies totally on the screen, no action is taken on the body's coordinates. CLIP1 will be discussed in greater detail later in Section III-G. Once the coordinate points have been projected and clipped to the film plane size, the body lines are drawn connecting the end point coordinates in proper sequential order. This order is determined by the array, MSEQ(4, 15, 15). Again, the drawing routine and the MSEQ array will be discussed in detail later in Sections III-I and IV-A. Once all of the above has been accomplished, the body is completely drawn on the film plane and the subroutine IMAGE returns control to the process loop.

H) Subroutine CLIP1

This subroutine retards any of the bodies projected onto the film screen from exceeding the boundaries of the film plane. If any line of a

projected body where allowed to exceed the film plane boundaries it would wrap around to the opposite side of the screen and be drawn there, giving a distorted representation of the body, see Figure III-11.

In Figure III-11, line AB has exceeded one of the film plane boundaries. The line segment B'B is the segment of line AB which has exceeded one boundary. This line segment has wrapped around to the opposite side of the screen and produced line segment CD on the screen.

To eliminate such distortions, each line that overruns a film plane boundary must be clipped so that it does not exceed that boundary value. Thus, in Figure III-11 line AB, must be clipped to line AB so that line CD does not appear on the screen. In other words, the clipping subroutine must determine that line AB exceeds the screen size, calculate point B', and replace point B with point B'.

There are four such screen overrun cases that the clipping algorithm must handle. These four cases are illustrated in Figure III-12.

The following discussion refers to the each case in Figure III-12 and what action must result in the clipping algorithm.

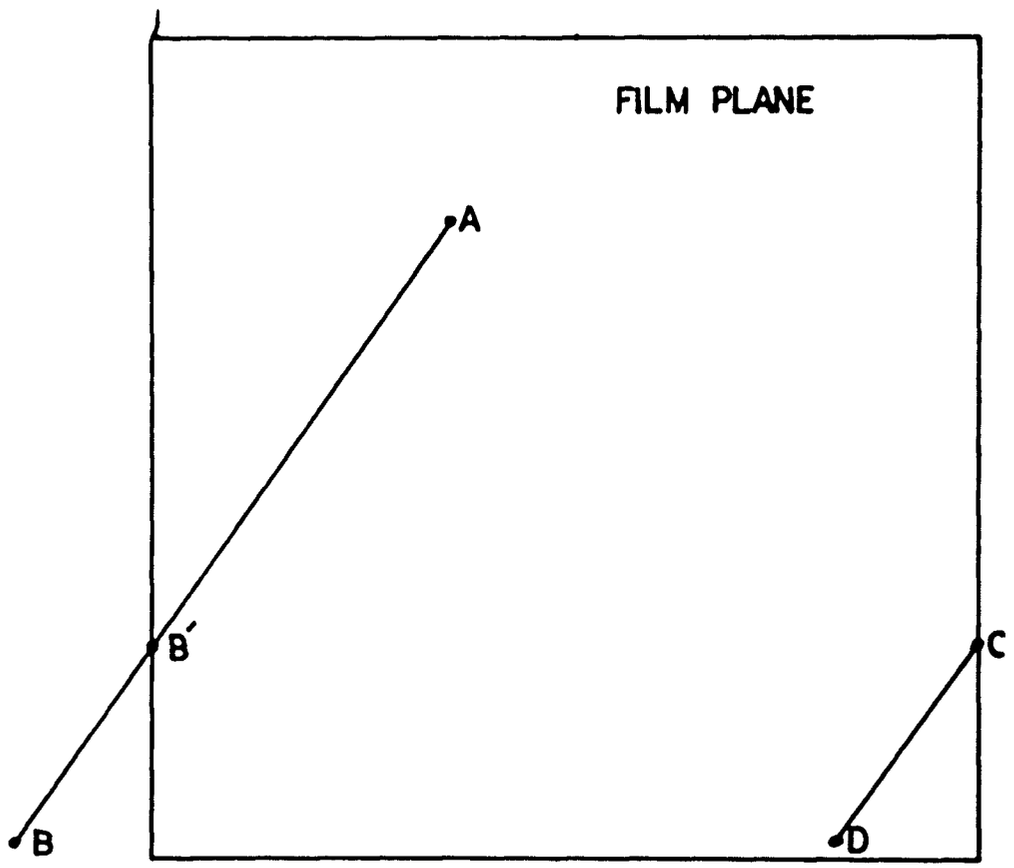



FIGURE III-11

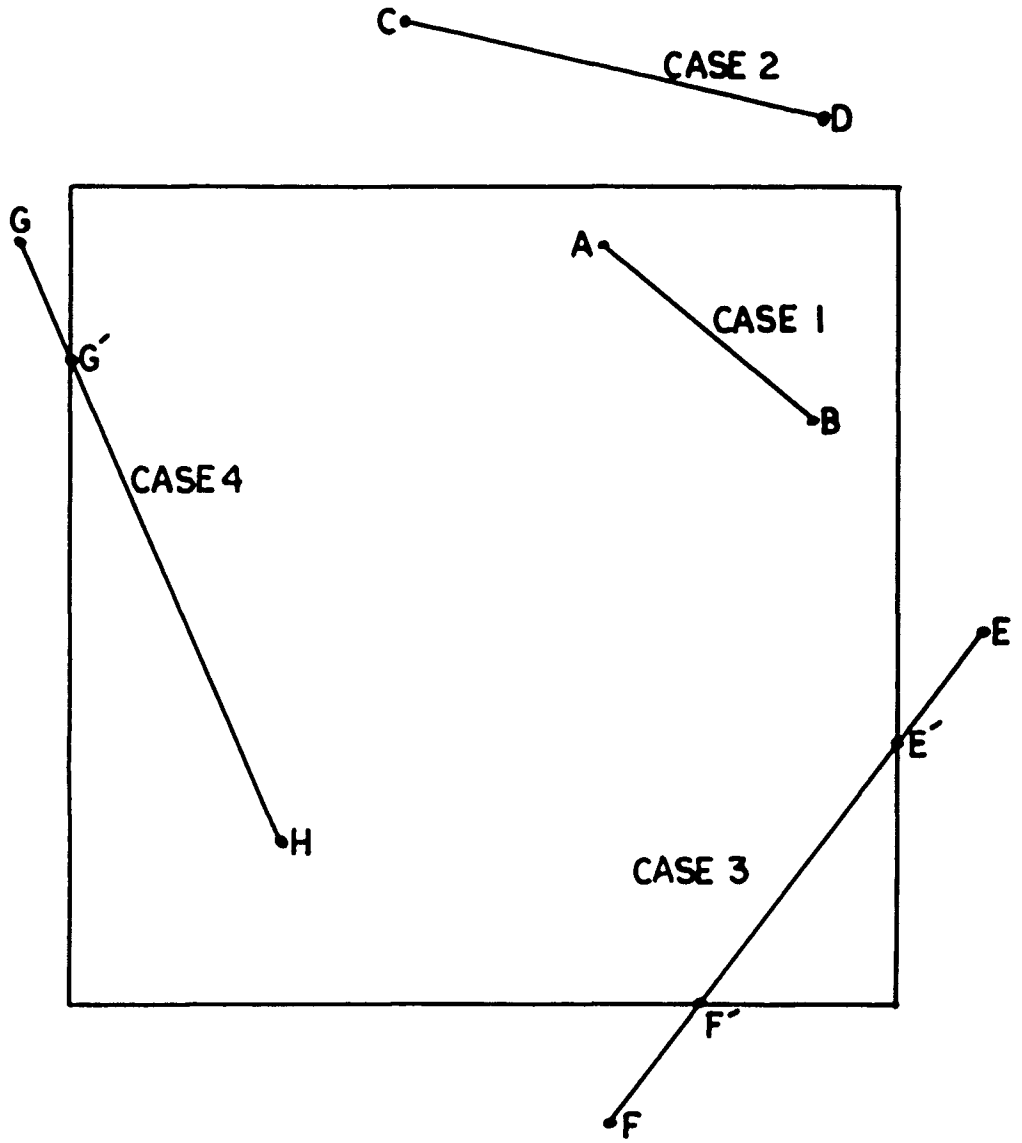



FIGURE III—12

MSEQ (1, 3, 4) = 5

This means that for body system 1, the 3rd line to be drawn connects points 4 and 5. Thus for the cube and the pyramid in Figure III-13, MSEQ would be defined, (for the cube),

| | |
|---|---|
| MSEQ(1,1,1) = 2 | MSEQ(1,5,3) = 4 |
| MSEQ(1,2,1) = 4 | MSEQ(1,6,3) = 7 |
| MSEQ(1,3,1) = 5 | MSEQ(1,7,6) = 5 |
| MSEQ(1,4,3) = 2 | MSEQ(1,8,6) = 7 |
| MSEQ(1,9,6) = 2 | MSEQ(1,11,8) = 7 |
| MSEQ(1,10,8)= 5 | MSEQ(1,12,8) = 4 |

(for the pyramid),

MSEQ(2,1,1) = 2

MSEQ(2,2,1) = 4

MSEQ(2,3,3) = 4

MSEQ(2,4,3) = 2

MSEQ(2,5,5) = 1

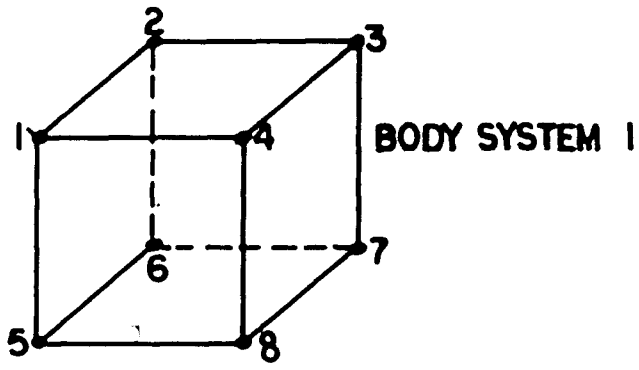
MSEQ(2,6,5) = 2

MSEQ(2,7,5) = 3

MSEQ(2,8,5) = 4

(The rest of MSEQ is set equal to zero by the subroutine START.)

This array must be defined immediately after subroutine START by the programmer as in Figure III-1. Immediately after defining MSEQ for all the bodies



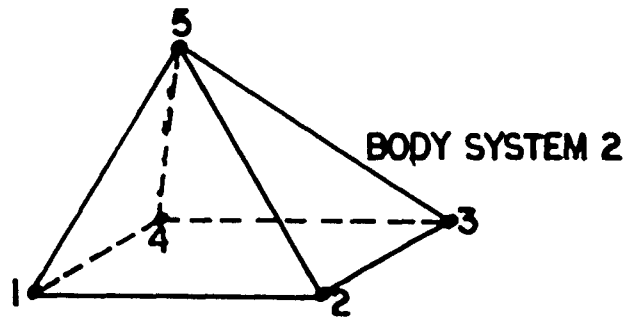




FIGURE III— 13

the programmer must store the array in the data structure. This is accomplished by the subroutine call:

```
CALL SEQ(MSEQ)
```

which stores MSEQ in the internal data base. Note that MSEQ must be dimensioned by the user as a four by fifteen by fifteen, three dimensional array. Thus, the programmer may only define four bodies with a maximum of fifteen lines per body.

J) Body Motion Subroutines

The discussion in this section deals with the box numbered 3 in Figure III-1.

There are six possible subroutine calls for defining rotation and translation of a body in the CCPS. Of these six motions, three are rotational and three are translational. The six subroutines are:

```
XTRAN       RT X2Z
YTRAN       RT Y2X
ZTRAN       RT Y2Z
```

Any of the above motions are performed within a particular body system before the body is transformed to another coordinate system. All of the above subroutines set up their respective motions but they do not actually perform the motion. The

subroutines that perform the actual rotations and translations are:

| | |
|---|---|
| XTR | X2Z |
| YTR | Y2X |
| ZTR | Y2Z |

Thus, the programmer uses the first set of subroutines to store information about how a certain body is to move in each picture frame. The second set of subroutines move the body as specified by the first set.

The second set of subroutines are invisible to the user and are called only by the subroutine IMAGE.

The discussion that follows will analyze each of the subroutines mentioned above.

1.) Subroutine XTRAN (IBODY, START, END, FEET)
When a user calls this subroutine a translation is set up in the X direction of the IBODY$^{th}$ body system. The translation will begin at frame START and end at frame END. FEET is the distance of the translation. FEET maybe negativ or positive. START, END, and FEET are assigned the same meanings in the following translational and rotational subroutines.

2.) Subroutine YTRAN (IBODY, START, END, FEET)
When this subroutine is employed a translation

is set up in the Y direction of the IBODY$^{th}$ body system.

3.) Subroutine ZTRAN (IBODY, START, END, FEET)
When this subroutine is referenced, it will set up a translation in the Z direction of the IBODY$^{th}$ body system.

The definitions of the body rotation subroutines match those defined rotations in Section II-C.

4.) Subroutine RT X2Z (IBODY, START, END, DEGS)
When this subroutine is called, it will set up a $\emptyset$ rotation of DEGS degree of the entire IBODY$^{th}$ body system.

5.) Subroutine RT Y2X (IBODY, START, END, DEGS)
When this subroutine is referenced, it will set up a $\Theta$ rotation of DEGS degrees of the entire IBODY$^{th}$ body system.

6.) Subroutine RT Y2Z (IBODY, START, END, DEGS)
When this subroutine is called, it will set up a rotation of DEGS degrees of the entire IBODY$^{th}$ body system.

Note that in the above three rotational subroutines that DEGS may either be a positive or negative number, or respectively, a positive or a negative rotation.

7.) Subroutine XTR (X, Y, Z, NCOOR, IBODY)
When this subroutine is employed, it will translate the IBODY$^{th}$ body coordinate system in the X direction as specified by the subroutine XTRAN. X, Y, and Z are the actual coordinates in the IBODY$^{th}$ body system. NCOOR is the number of coordinates in the IBODY$^{th}$ body system. All input parameters are supplied by the subroutine IMAGE. X, Y, Z, and NCOOR assume the same definitions in the following subroutines as above.

8.) Subroutine YTR (X, Y, Z NCOOR, IBODY)
When this subroutine is called, it will translate the IBODY$^{th}$ body system's body in the Y direction as specified by subroutine YTRAN.

9.) Subroutine ZTR (X, Y, Z, NCOOR, IBODY)
When this subroutine is called, it will translate the IBODY$^{th}$ body system's body in the Z direction as specified by subroutine ZTRAN.

10.) Subroutine X2Z (X, Y, Z, NCOOR, IBODY)
When this subroutine is referenced, it will rotate the entire IBODY$^{th}$ body system as specified by RTX2Z.

11.) Subroutine Y2X (X, Y, Z, NCOOR, IBODY)
When this subroutine is called, it will rotate

the entire IBODY$^{th}$ body system as specified by subroutine RT Y2X.

12.) Subroutine Y2Z (X, Y, Z, NCOOR, IBODY)
When this subroutine is called, it will rotate the entire IBODY$^{th}$ body system as specified by subroutine RT Z2Y.

When any one of the first six subroutines are called by the user data is stored immediately in the internal data structure. Therefore, it becomes apparent that only one of each of the six subroutines may be called for a particular body during Step 3 in Figure III-1. Thus, there can only be six body motion subroutine calls per body system during Step 3.

Finally, it should be mentioned that when the user employs these translational subroutines he must be very aware of where the bodies are moving in respect to the reference coordinate system and where the bodies are initially oriented. This is vital since these two parameters determined where in the camera coordinate system the body will be transformed and thus where, if at all, the body will be projected onto the film plane. To make the system easier to use, the new user should define all initial body orientations with zero displacement and zero angular rotation, thus placing his initial programming attempts in the middle of the film plane.

K) CCPS Programming Techniques

Due to the limited size of the H. P. 1000's core memory specialized programming techniques, restricted to R.T.E. software, must be utilized. Since the CCPS is such a large program, all of it cannot reside in the H. P. 1000's core memory at one time without memory overflow errors occuring.

To alleviate this problem, the CCPS's main structure is broken up into two substructures or programming pieces. These pieces are called segmented programs.

A.) Segmented Programs

This section will deal with what programming considerations are necessary to run an animated motion picture process with the CCPS, via segmented programs.

Segmented programs are discussed in some detail in Hewlett-Packard's "RTE: A Guide for New Users" manual pp. 3-13-3-17 and the "RTE/II Software System Programming and Operating Manual" pp. 3-24-3-25 and 4-33-4-34.

Program segmentation allows the user to load the CCPS from disc to core memory, a segment at a time and simultaneously execute the CCPS. An attempt to load the execute the entire CCPS from disc would

result in a loader memory overflow error due to the CCPS's length and the H. P. 1000's limited core storage capacity.

The CCPS is broken up into a main program and two segments. The main program's sole function is to allow the system to jump to its first segment, where upon it jumps to the second segment after it has finished execution. The second segment executes and ends the animation process. Thus, the super structure of the CCPS maybe pictured as in Figure III-14.

The user must therefore begin by writing a main program.

1.) The Main Program

The purpose of this program is to set up the segmentation of the CCPS and to start the system off by jumping to the first segment. The program flow for the main program is illustrated in Figure III-15. The numbered discussions refer to the numbered boxed in Figure III-15. The numbered discussions refer to the numbered boxes in Figure III-15.

a.) In this step, the user must name the main program and designate as a type 3 program. The following is an example of a typical program card meeting the above requirements:

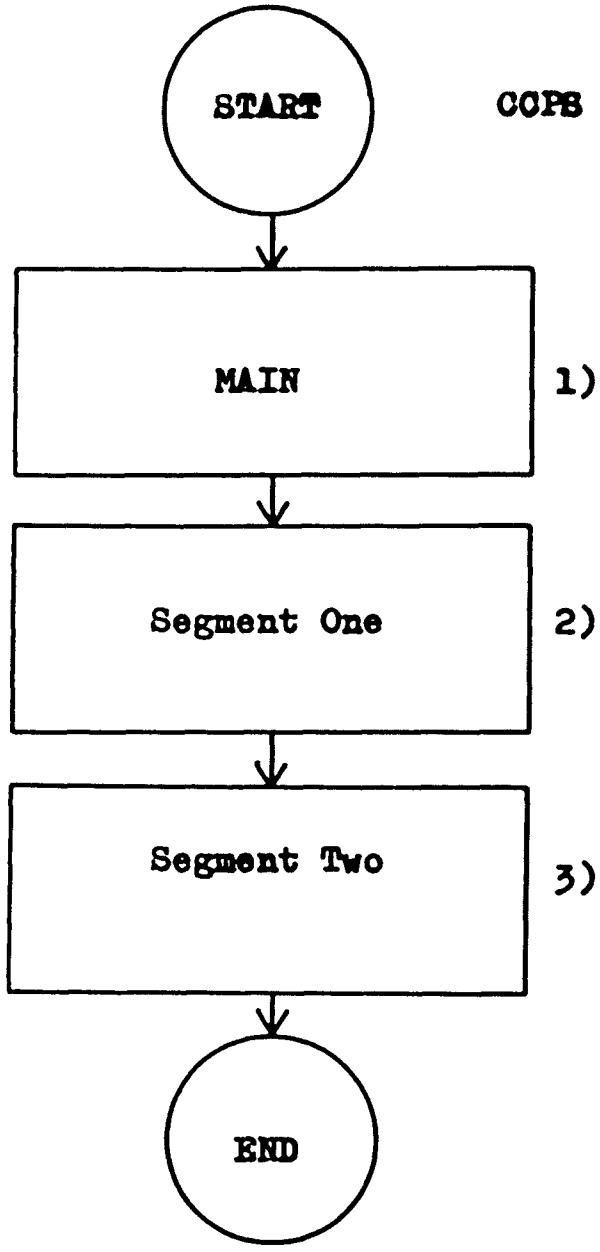



FIGURE III-14

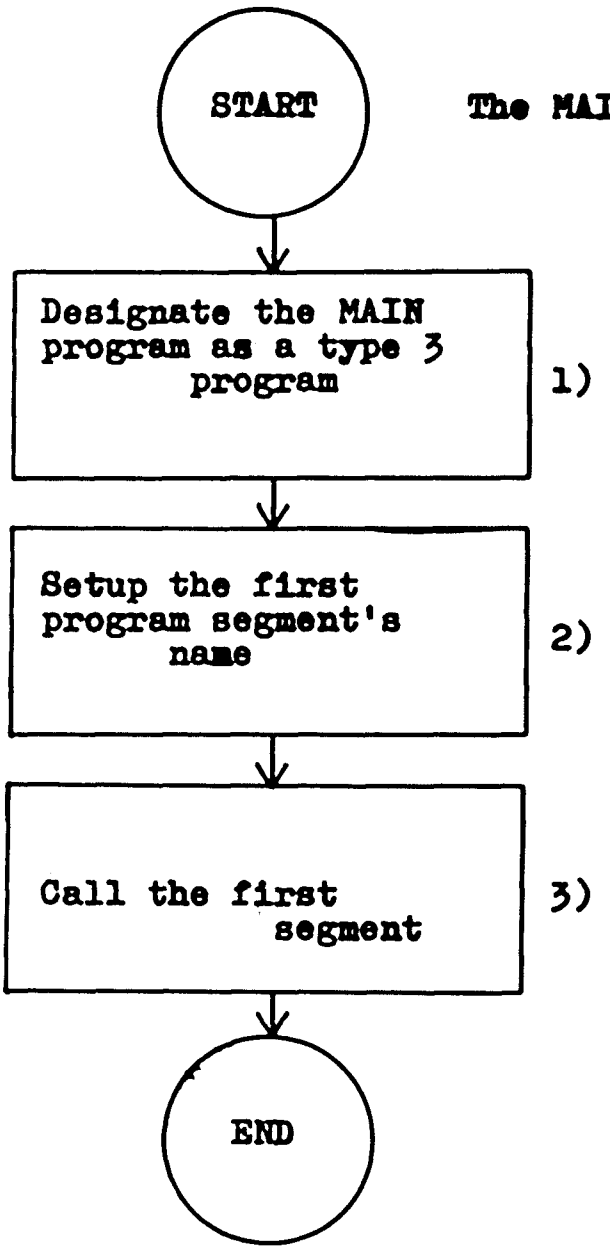



FIGURE III-15

```
PROGRAM MAIN, 3
```

b.) In this step the user must name his first segmented program so that the main program knows where to jump to. To accomplish this requirement requires the user to employ an INTEGER and a DATA statement to dimension and store the segment's name under another variable name. An example of the two statements required are:

```
INTEGER NAME (3)
DATA NAME/2HSG, 2HMT, 2HO/
```

Thus, the segment's program name, SGMTO is stored in a three dimensional array called NAME. This format must be followed to ensure proper program execution.

c.) In this step, the main program calls its first segment. The statement required is an executive call for a segment load. This statement is discussed in detail in the second reference given in this section above. The program then gives program control to its first segment and the segment begins execution. It must be noted that once program control is

given to any segment, control cannot be returned to the calling program whether it be a main or a segmented program. Thus, the main program cannot be reentered once its first segment is called. This is because only one segment can reside in core at one time. The program segment load executive call required in the main program MAIN would read:

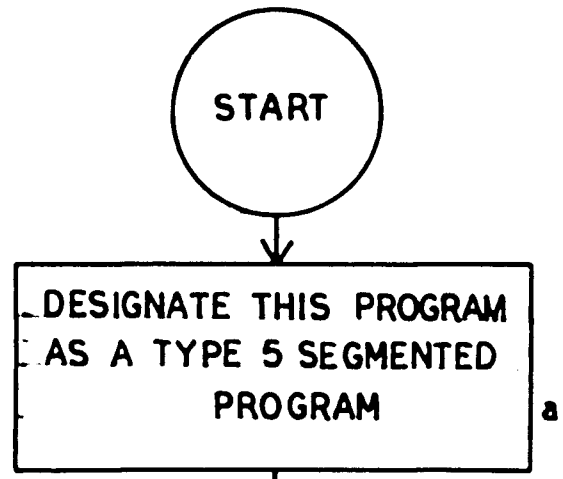
```
CALL EXEC(8, NAME)
```

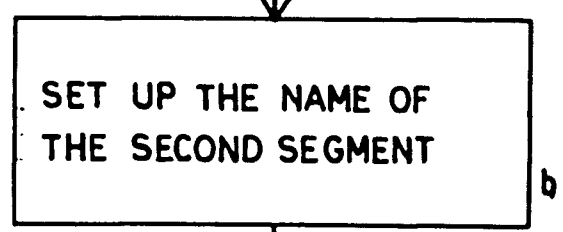
The number eight indicates that this is an executive segment load, and NAME is the variable containing the first segment's program name.

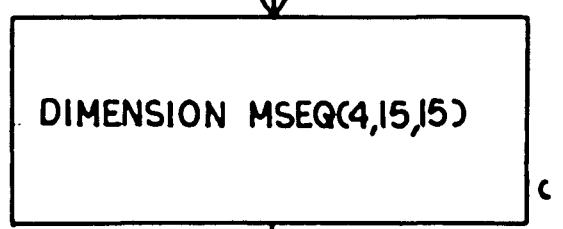
An example of a main program for CCPS segmentation is given in Appendix A, with the program name, MAIN. The user may store his main program under any file name he desires.

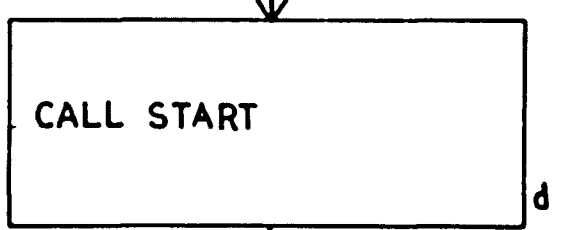
2.) Segment One

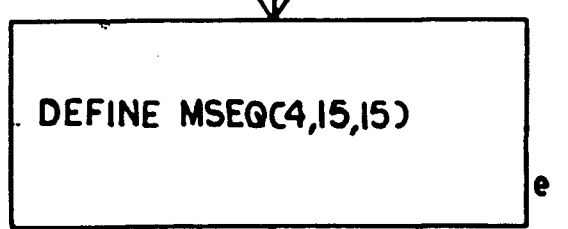
In this first program segment the user must define a fixed number of parameters. A flow diagram is given in Figure III-16 to help illustrate what parameters are required. The

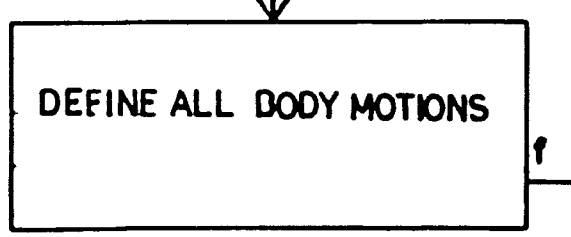


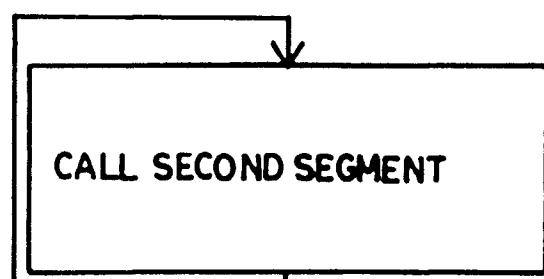



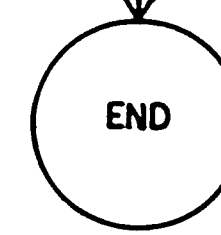




FIGURE III-16

FIRST PROGRAM SEGMENT

letters beside the boxes refer to the lettered discussions below.

a.) In this step, the user must define the program's name and specify it as a type 5 program, which is an indicator for a segmented program. In the previous example, the first segment was named SGMTO and thus the program card required here is:

```
PROGRAM SGMTO, 5
```

b.) In this step, the user must dimension the array MSEQ (4, 15, 15) for reasons stated in Section III-I. The statement must appear exactly as:

```
DIMENSION MSEQ (4, 15, 15)
```

c.) In this step, the user must set up the name of the second segment as he had done previously in the main program for the first segment. Following the same procedure, if the second segment is to be called SGMTT, then the two statements required are:

```
INTEGER NAME (3)
DATA NAME/2HSG, 2HMT, 2HO/
```

d.) In this step, the user must call the subroutine START. The subroutine call statement is simply:

```
CALL START
```

Then he will be asked to enter specific data as described in Section III-D.

e.) At this step, the programmer must define his point connection scheme for each body. What statements are required have been discussed in detail in Section III-I.

f.) At this step, the programmer must define all body motions. There can be only six types of motions specified for each body. These motions are described by the subroutines XTRAN, YTRAN, ZTRAN, RTX2Z, RTY2X, and RTY2Z. An example of the motions that may be described for two bodies would be:

```
XTRAN (1,2., 10., 10.)
RTX2Z (1,9., 15., 30.)

ZTRAN (2, 2., 10., 15.)
RTY2X (2, 5., 15., 45.)
RTZ2Y (2, 10., 12., 90.)
```

Each of the six body motions are described in detail in Section III-J.

g.) In this step, the programmer must make an executive segment load call in the same exact manner as he did in the main program, for the second segment. Again the segment call is:

CALL EXEC (8, NAME)

The parameters are identical to the parameters in the executive call in the program MAIN, except this time the segmented program SGMTT is loaded. At this point in the CCPS program flow, program control is turned over to the segment SGMTT.

Thus, program SGMTO has accomplished its three important facets of the CCPS. It has (completed steps 1, 2, and 3 in Figure III-1).

1.) Enter all initializing data via the subroutine START and has stored this data in the internal data structure.

2.) Define all line connection schemes, and has stored this data in the internal data structure.

3.) Set up all the body motions, translations or rotations, and has stored this data away internally.

Since SGMTO is essentially a mass data storage routine, and once this data has been stored, SGMTO is actually no longer necessary, and the second segment may be

loaded into core memory in its place. This then allows the CCPS additional core storage. As the first segment can be thought of as a mass data storage algorithm, the second segment may be thought of as the actually filming routine for the CCPS.

3.) Segment Two

In this segment the actual transformations, translations, rotations, projections, clipping, and drawing of all the bodies takes place. A program flow diagram is supplied in Figure III-17. The numbers appearing next to the boxes reference the discussions below.

a.) Again in this step, this program must be designated as a segment. Since in the first segment this program was named SGMTT, the program card for this program segment must read:

```
PROGRAM SGMTT, 5
```

b.) At this point, the programmer must define the X, Y, and Z coordinates of all of the bodies, he wishes to film, with respect to their own body coordinate systems. The

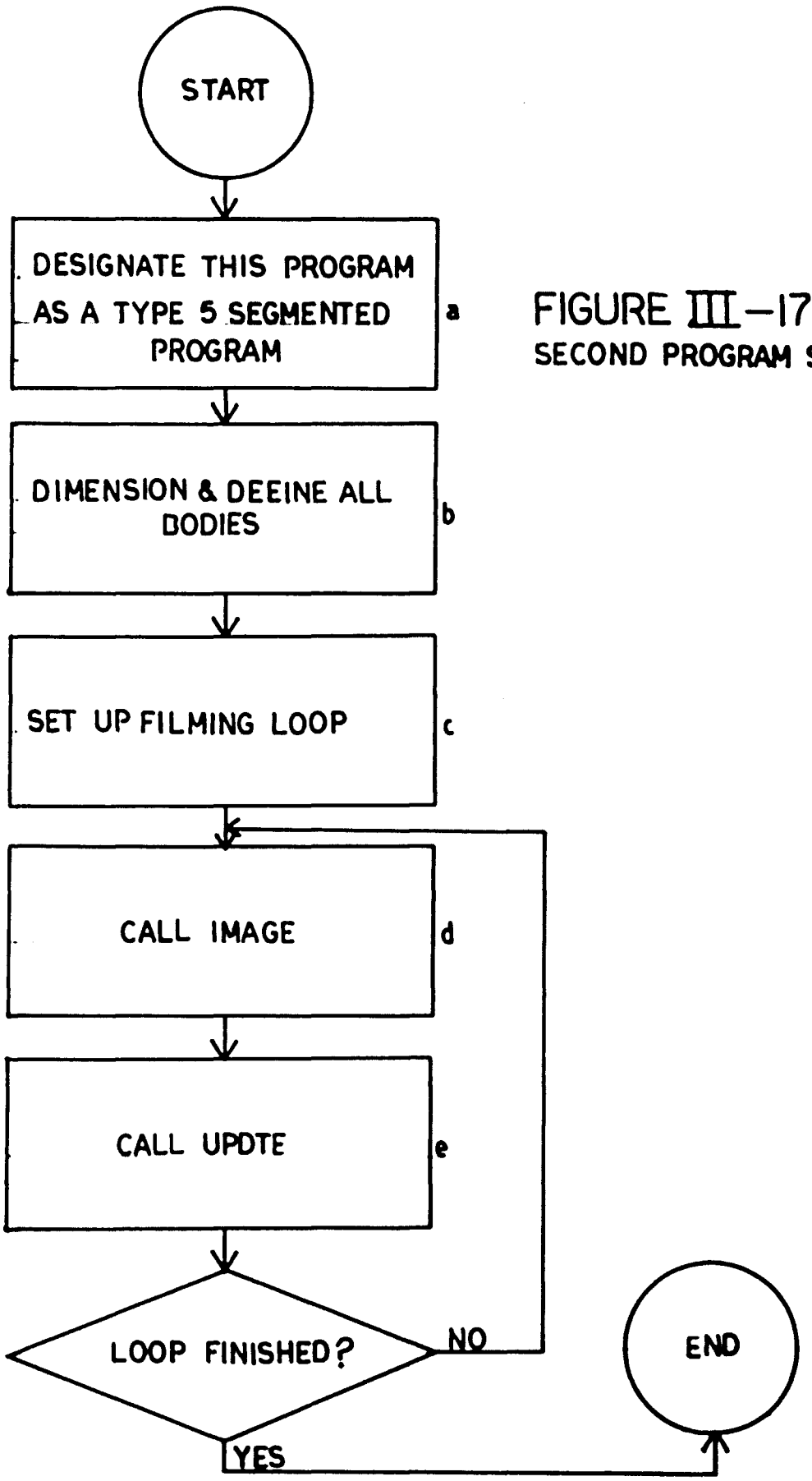



FIGURE III-17
SECOND PROGRAM SEGMENT

X, Y, and Z coordinates must be defined in DATA statements and therefore must be dimensioned.

As an example of how the bodies may be defined, refer to Figure III-13. In body coordinate system one is configured a cube. Point 1 of the cube could assume the coordinate value, (0, 0, 0), for X, Y, and Z respectively. Point 2 could be defined (0, 0, -5), Point 3, (5, 0, -5), etc. All of the points in body system one may be thusly defined with DATA statements as:

```
DIMENSION X(8), Y(8), Z(8)
DATA X/0.,0.,5.,5.,0.,0.,5.,5./
DATA Y/0.,0.,0.,0.,5.,5.,5.,5./
DATA Z/0.,-5.,-5.,0.,0.,-5.,-5.,0./
```

where X(1), Y(1), Z(1) defines point 1, X(2), Y(2), Z(2) defines point 2, etc. <u>All bodies must be defined this way in the CCPS</u>. For the pyramid in Figure III-13, the statements defining its coordinates could read:

```
DIMENSION XX(5), YY(5), ZZ(5)
DATA XX/0.,5.,5.,0.,2,5/
DATA YY/0.,0.,0.,0.,5./
DATA ZZ/0.,0.,-5.,-5.,-2.5/
```

Therefore, the arrays X(8), Y(8), and Z(8) define the body in body system one and the arrays XX(5), YY(5), and ZZ(5) define the body in body system two.

Up to four bodies and body systems may be defined in the CCPS in a likewise manner.

c.) In this step, the programmer must set up the filming loop. Each pass through this loop is one picture frame taken by the camera. The loop is defined by the user as a simple DO loop of the form:

```
DO n, I = 1, nf
```

where n is the terminal statement of the filming loop, always a CALL UPDTE statement, and nf is the number of frames the programmer wishes to process.

d.) In this step, a CALL IMAGE statement is required for each body coordinate system. If the bodies defined in step 2 are used as an example, the two CALL IMAGE statements in the filming loop must be written:

```
CALL IMAGE (X, Y, Z, 8, 1)
CALL IMAGE (XX, YY, ZZ, 5, 2)
```

where X, Y, and Z define the body residing in body system one, the number 8 states the

number of coordinates defining the body, and the number 1 states which body coordinate system the body belongs to. The parameters for preceding CALL IMAGE statement may be defined in exactly the same manner. The IMAGE subroutine performs all of the rotations, translationals, transformations, projections, clipping, and drawing of the previously defined bodies, a frame at a time, as defined in the body motion subroutines in segment one. For a detailed description of IMAGE, refer to Section III-G.

e.) At this step, the user must make a call to the subroutine UPDTE which updates the CCPS as defined in Section III-E. This subroutine call is the final statement in the filming loop. This statement is numbered with the terminal line, I, number specified by the filming loop's DO loop statement. An example of this statement would be:

```
10 CALL UPDTE
```

At this juncture, the CCPS has been updated. If the loop process is not finished filming it returns to its beginning. If the loop is

finished, the animated filming process is complete and the CCPS program ends.

Note that in this example of the filming loop, as each body is drawn on the film plane, it remains there throughout the filming process. If this is undesirable the user may insert a CALL ERASE statement within the loop to erase each successive frame. Also note, that the bodies are drawn in the order of their CALL IMAGE statements. Thus, in the previous example, body 1 is drawn before body 2.

An example of a segmented program illustrating the above discussion may be found in Appendix A, named SGMTT.

L) The CCPS Procedural File

To load all of the programs, subroutines, and segmented programs of the CCPS from the H. P. 1000's disc to core memory, a procedural filed name CCPS has been written.

To run this program the user must modify the program by first entering the names of his main and two segmented programs in the proper location in the program. To accomplish this, the user may enter the EDITR program and request source file, CCPS. Once this is done he replaces

lines 19, 25, and 34 with his main and two segmented programs object files. He then saves the file again under the file name CCPS, and types, when in the FMGR system program:

CCPS

The CCPS is then loaded into core memory and ready to run. The user may then run the animated film process as he would a normal program by typing RU main program , while in the RTE system program. The program will then run as discussed in all the previous sections. (It is assumed here that the user has a good general working knowledge of the RTE, FMGR, and EDITR commands and of creating Transfer Files.)

This concludes all discussion concerning the CCPS. The next section will deal with the data plotting subroutines developed for the H. P. 1000.

## IV. The PLOTT Package

This section of the thesis will address itself to a discussion of internal structure and the user implementation of the data plotting routines developed as part of the overall graphics package for the H. P. 1000.

### A) Introduction

The routines discussed here will draw a user specified gridiron plotting surface with high lighted divisions. The axes will be automatically scaled to the user's data, with the maximum and minimum values

of each of the one-dimensional data arrays displayed on the graph. The graph may then be titled as the user choses.

The routines to be discussed are:

1.) GRID

2.) SCALE

3.) TITLE

There will also be a discussion on the PLOTT procedural file and PLOTT programming technique.

B) The PLOTT Subroutines

1.) Subroutine GRID (JNX, JVX, MODE, IVX, IVY, NPT, IAN, X, Y)

This subroutine draws the actual gridiron surface along with its highlighting. The input parameters can be defined as follows:

a.) JNX - the number of major graph divisions along the X-axis. This number must divide into 200 evenly.

b.) JNY - the number of major graph divisions along the Y-axis. This number must divide into 200 evenly.

c.) MODE - the parameter sets the H. P.'s video field. A zero indicates that the graph and its lettering will be white on a

black background. A one indicates the reverse.

d.) IVX - is the number of X highlighted divisions. This number must divide evenly into JNX.

e.) IVY - is the number of Y highlighted divisions. This number must divide evenly into JNY.

f.) NPT - is the number of data points in the programmer's data arrays.

g.) IAN - determines whether the plot will be a point plot (IAN = 0) or a line plot (IAN = 1).

h.) X- the X axis data array.

i.) Y- the Y axis data array.

JNX and JNY must be even divisors of 200 because 200 point spacings have allowed for all graphs created by the subroutine GRID. The H. P.'s video display is defined by a 256 x 256 point matrix, allowing an even spacing of 28 points around the graph for displaying the titles and the scaling.

Therefore, it is apparent that the maximum value that NPT may be assigned is

200 and the maximum number of X and Y values is 200.

IVX and/or IVY may be set equal to zero, which indicates that no highlighting is desired along the X and/or Y axis.

The program flow diagram is illustrated in Figure IV-1 for the subroutine GRID. The numbered discussions below refer to the numbers beside the boxes in the figure.

1.) In this first step, the subroutine checks to see that JNX or JNX do not equal zero. This check is made to ensure that no quantity is divided by zero. If JNX or JNX is zero a warning is printed on the system console of the form:

JNX OR JNX CANNOT EQUAL ZERO!!!

and the subroutine stops execution. Otherwise, the subroutine proceeds on to step 2.

2.) In this step, the H. P.'s video display screen is initialized. This is accomplished by two subroutine calls which inform the computer what device it is to write to and in what mode (white on black or black on white). These two subroutine calls are, (from the H. P.'s video library)

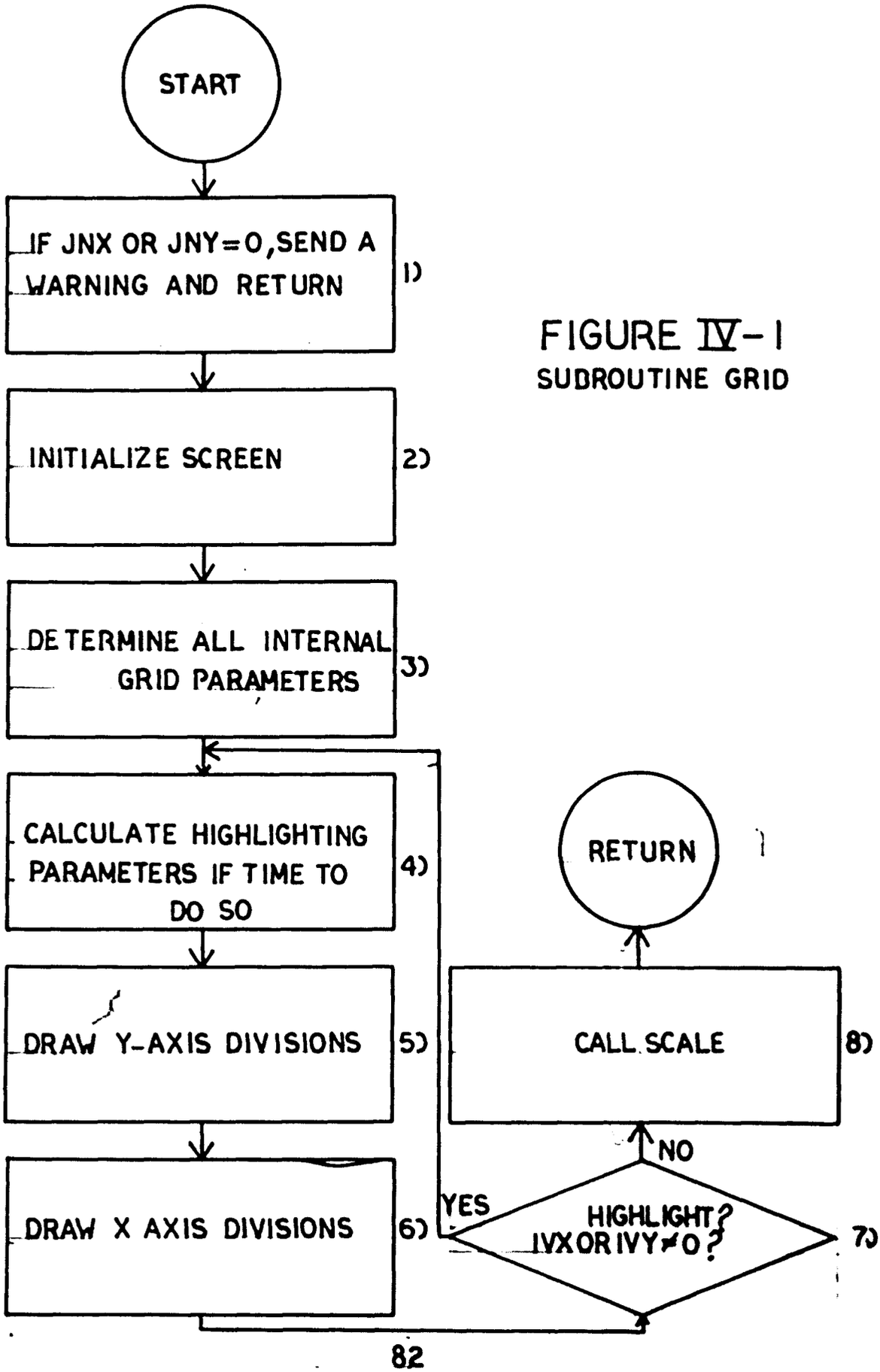



FIGURE IV-1
SUBROUTINE GRID

CALL VIDLU (9, MODE)

CALL ERASE

The number nine indicates that the computer is to write to device #9, the television monitor, and MODE is one of the previously defined parameters. The subroutine call CALL ERASE clears the screen of any previous displays.

3.) In this step, all the required internal variables are calculated for the subroutine GRID via the passed parameters JNX, JNY, IVX, and IVY. These passed parameters calculate the point spacings necessary between each division of either axis and determine the length of each divisional line.

4.) In this step, if it is necessary to determine the internally variables to draw the highlighting on the grid, these variables are then found via the parameter IVX and IVY. The grid's highlighting is always performed on the second pass thru the program loop in Figure IV-1. The loop pass draws the main grid itself.

5.) In this step, the Y-axis divisions are either drawn for the grid or the grid's

highlighting thru the use of the previously defined variables and parameters. The subroutines used to draw the Y-axis divisions are, VECTR and VEND, two of the H. P.'s video sub-routines found in the H. P.'s video library.

6.) At this step, the X-axis divisions are drawn in the identical manner as above.

7.) At this juncture, if highlighting is to occur, the subroutine returns to step 4, to determine the highlighting parameters. If the highlighting has occured or isn't to occur (IVX and IVY equal zero) the routine proceeds to step 8.

8.) In this step, the routine GRID makes a subroutine call to SCALE. The subroutine SCALE takes the parameters passed to it and scales the displayed grid and plots the user's X-Y data on the grid as either a line plot or a point plot.

The subroutine SCALE will be discussed in Section IV-B-2.

Once the subroutine SCALE has executed, program control is returned to the calling program.

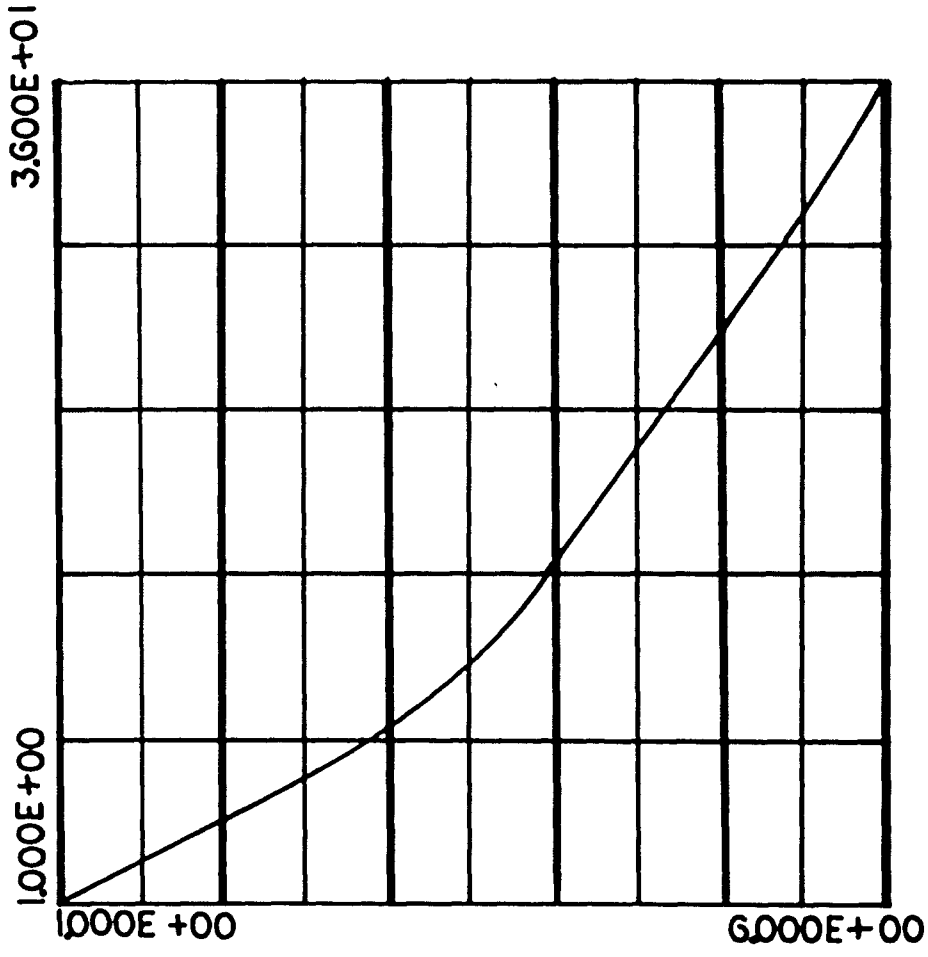
As an example of how the subroutine GRID functions, suppose the following call were made in a user's program with the following statements:

```
    DO  10  I  =  1, 6
    X(I)    =  I
10  Y(I)    =  I * * 2
    CALL GRID (10,5,0,2,0,5,1,X,Y)
```

It is obvious from the program that the X-Y data will take on the values in Figure IV-2. The resulting gridiron plotting surface is a line plot (IAN = 1) of the equation, $Y = X^2$, where X varies between one and six. The X-axis is broken up into ten divisions (JNX = 10) with a highlighted division line, every second division (IVX = 2). The Y-axis is broken up into 5 divisions (JNY = 5) with no highlighted lines (IVY = 0). The number of points is six (NPT = 6).

2.) The Subroutine SCALE

This subroutine scales the X and Y axes of the plotting grid created in the GRID subroutine and plots the user's X - Y data either as a point plot or a line plot. The subroutine is only called by the subroutine GRID and is never used by the general programmer. The subroutine call SCALE is:




| X | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Y | 1 | 4 | 9 | 16 | 25 | 36 |

FIGURE IV-2

CALL SCALE (X, Y, NPT, IAN)

where X, Y, NPT, and IAN assume the same specifications they did in subroutine GRID.

3.) Subroutine TITLE

This subroutine titles the grid created in the two previous subroutines. It will title the X and/or the Y axis at the user's discretion. The user must call this subroutine right after he has called GRID. The subroutine call for the program is:

CALL TITLE

There are no parameters to be passed to the subroutine.

When the subroutine is called, it will reply with two requests for input on the system console. The first request is:

**TYPE THE X-AXIS TITLE (10 CHARACTERS MAXIMUM)**

The user then types in the X-axis title he desires and presses return. The second request appears on the system console as:

**TYPE THE Y-AXIS TITLE (10 CHARACTERS MAXIMUM)**

The user replies in the obvious manner, and presses return.

A flow diagram for the TITLE subroutine is depicted in Figure IV-3. Again the numbers beside the boxes refer to the numbered discussions below.

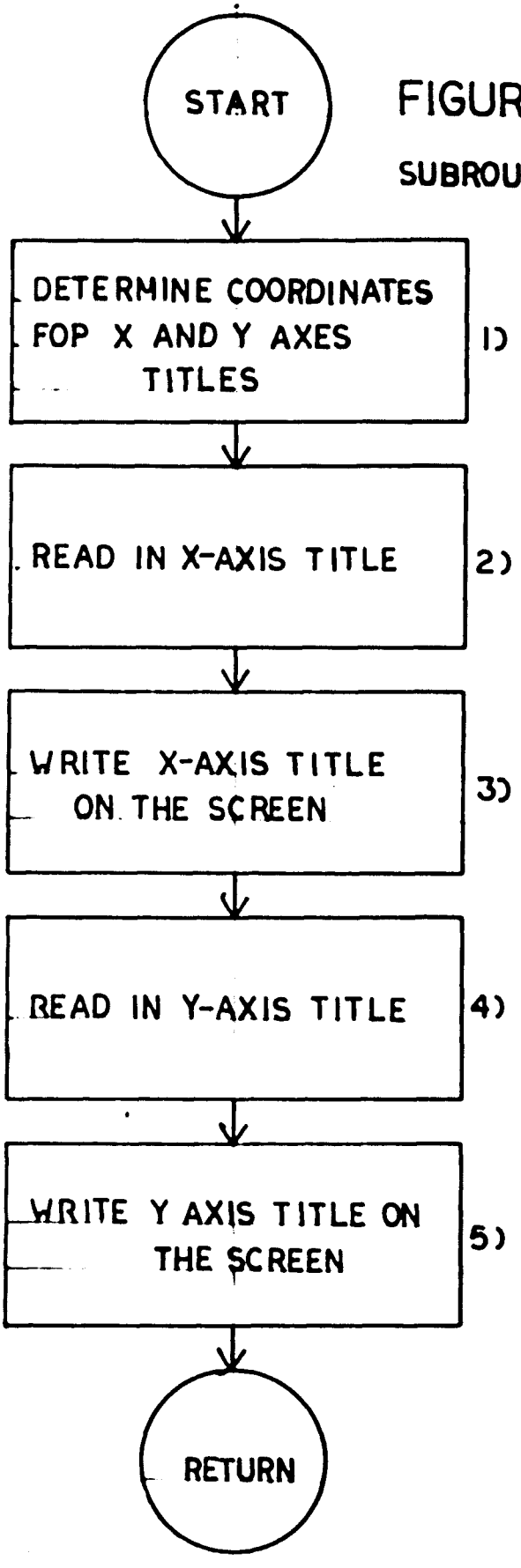



FIGURE IV-3

SUBROUTINE TITLE

1.) The subroutine first sets the coordinates where the X and Y axis titles are to be displayed. These coordinates are predetermined due to the graph's size.

2.) In this step, the X-axis title is read in at the system console. The title is requested by the computer using a WRITE statement whose format is the first program request above.

3.) In this step, the X-axis title is displayed on the screen using one of the H. P.'s video subroutines named CHAR. This subroutine will print any of the legal ASCII characters for the H. P. 1000, anywhere on the screen, in any size.

4.) In this step, the Y-axis title is read in using the second computer request above in the identical manner.

5.) Again in this step, the Y-axis title is displayed as in step 3.

A complete discussion of the H.P. 1000's video library may be found in the "HP91200A TV Interface Kit, Programming and Operating Manual" in particular pp. 4-1 - 4-7 on software programming.

A complete example program using the subroutines GRID, SCALE, and TITLE may be found in Appendix B with the program name, DEMO.

C) Block Data K

This program is the common block storage routine for the PLOTT package. It allows the parameters LL, MM, LENX, and LENY to be passed from the GRID subroutine to the SCALE subroutine via the COMMON statement labeled LABEL.

D) The PLOTT Procedural File

In the PLOTT package there exists a file named PLOTT which is the procedural file for the system. This file will load from the disc into core memory the object files needed for the programmer to use the GRID and TITLE subroutines. It will also attach these object files to the object file of his main program. The easiest way to show how the file PLOTT works is by an example. If the programmer's main program was named JIM, and his object file, %JIM, and this program called the subroutines GRID and TITLE, he could type the following:

```
LG, 6
MR, %JIM
 PLOTT
```

at which the computer would reply:

```
MR, %GRID
MR, %SCALE
MR, %TITLE
MR, %K
```

```
MR, %LINE
MR, %TVLIB
RU, LOADR, 99,,,,3
/LOADR  JIM READY
/LOADR  $END
```

This would indicate that the PLOTT package has been properly loaded and attached to the main program JIM. The user may now run the program JIM as he normally would with a RUN command:

```
*RU, JIM
```

Thus, the PLOTT file contains the following FMGR system commands:

```
MR, %GRID
MR, %SCALE
MR, %TITLE
MR, %K
MR, % LINE
MR, $TVLIB
RU, LOADR, 99,,,,3
```

It is assumed the user has a general knowledge of the FMGR and RTE system commands necessary to run a program.

This concludes discussion on the PLOTT graphing package.

## IV-A) The DRAW Subroutine

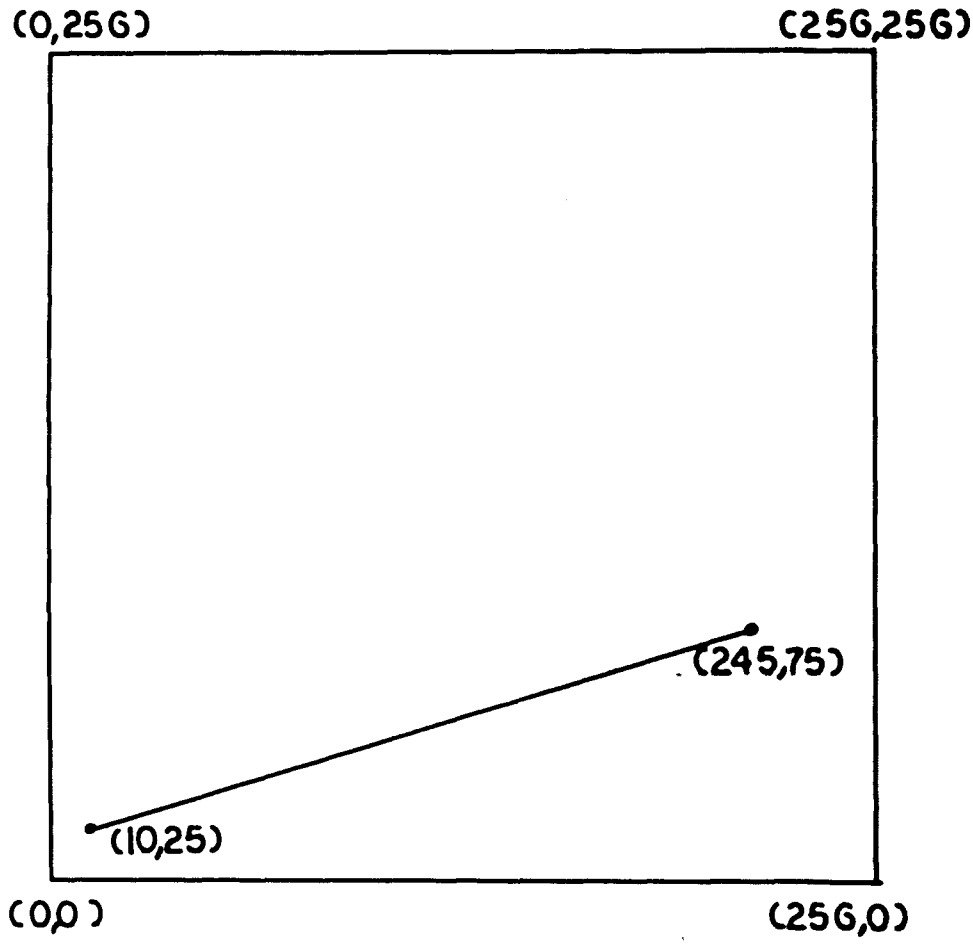
This subroutine is being discussed here because it is utilized by both the CCPS and PLOTT packages. What this subroutine does is to draw a line between any two coordinates on the H. P.'s video display. The video display is defined by a 256 x 256 point matrix, with the lower left hand corner defined as (0,0) and the upper right hand corner defined as (256, 256). The programmer's call for the subroutine is:

```
CALL DRAW (IX, IY, LX, LY)
```

where (IX, IY) is the initial X-Y coordinate and (LX, LY) is the terminating X-Y coordinate. Thus, a line is drawn between (IX, IY) and (LX, LY).

The line is drawn by lighting the discrete points closest to the actual line. To illustrate this refer to Figure IV-A-1. In this figure a line has been drawn between the points (10, 25) and 245, 75) via the CALL DRAW statement given. If the lower end of the line were enlarged, the line in Figure IV-A-2 would result. The subroutine is lighting the point closest in the Y direction for each X value along the line.

The equation for the line in the example is, y= 0.213 X + 22.872. For, X = 15, along the line, the value for Y is 26.067. Therefore the point lit up must be (15, 26), the point closest to the actual line.




CALL DRAW(10,25,245,75)

FIGURE IV-A-1

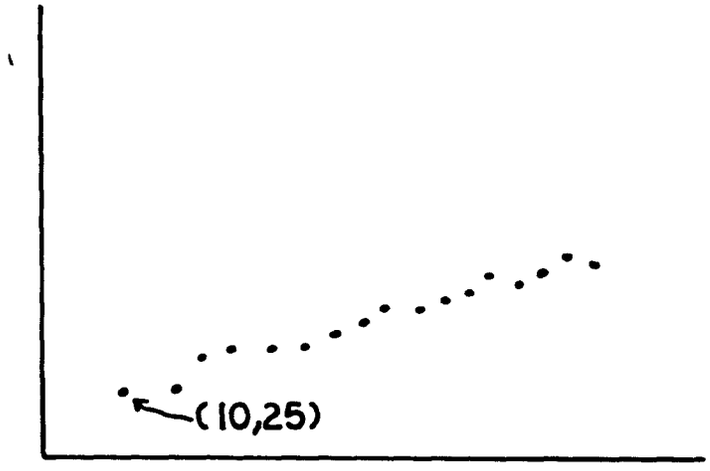



FIGURE IV-A-2

It is apparent that for slopes less than a Y value is determined for each X value and for slopes greater than one a X value is calculated for every Y value. This is to make as sure as is possible that all lines appear as solidly drawn lines with no breaks or irregularities.

Note that all defined coordinates must be integer variables to be used with the subroutine DRAW. No program flow is supplied for this subroutine since the subroutine is used within the internal structures of the CCPS and the PLOTT package, and is not seen by the general user.

The subroutine is stored under the file name, &LINE, and its object file name is %LINE.

## V. The Three-Dimensional Vector-Matrix Package

### A) Introduction

This section discusses a package of subroutines developed as an aid in three-dimensional vector and matrix problems. Although developed primarily as an aid to the CCPS matrix manipulations it can be employed in a wide spectrum of applications. These applications could include static and dynamic mechanics, electromagnetic waves and fields, partial differential equations, etc. The subroutines discussed in this section perform the following mathematics:

1.) |A| -finds the absolute value of a vector

2.) A + B -sums two vectors

3.) A - B -subtracts two vectors

4.) A = B - equates two vectors

5.) A · B -dots two vectors

6.) $PROJ_A B$ -projects one vector on another

7.) C x A -multiplies a scalar by a vector

8.) A x B -cross multiplies two vectors

9.) A · i -finds the i-th component of a vector

10.) A · j -finds the j-th component of a vector

11.) A · k -finds the k-th component of a vector

12.) $[M]^{-1}$ -inverts a matrix

13.) [A] * [B] -multiplies two matrices

14.) [A] + [B] -sums two matrices

15.) [A] - [B] -subtracts two matrices

16.) C x [A] -multiplies a matrix by a scalar

17.) [A] x B -multiplies a matrix by a vector

B) The TDPKG Subroutines

The discussions that ensue will analyze each subroutine as to how it is used, the input-output parameters, and what results or quantities it generates.

1.) Subroutine VABS (X, Y)

This subroutine calculates the absolute value of a vector. X is a three-dimensional input vector

and Y is equal to |X|. X must be dimensioned properly in the main program.

2.) Subroutine VSUM (X, Y, Z)

This subroutine sums two vectors together. X and Y are two three-dimensional input vectors and Z is equal to X + Y. X, Y, and Z must be properly dimensioned.

3.) Subroutine VDIFF (X, Y, Z)

This subroutine subtracts two vectors. X and Y are two three-dimensional input vectors, and Z is equal to X - Y. X, Y, and Z must be dimensioned properly in the calling program.

4.) Subroutine VSAME (X, Y)

This subroutine equates two vectors. X is the three-dimensional input vector and Y is equal to X. X and Y must be dimensioned properly in the main program.

5.) Subroutine VDOT (X, Y, A)

This subroutine determines the dot product (inner product) of two vectors. X and Y are two three-dimensional input vectors and A is equal to X · Y. X and Y must be dimensioned correctly in the calling program.

6.) Subroutine VPROJ (X, Y, A)

This subroutine projects one vector on to another vector. This is a scalar projection

not a vector projection. X and Y are two three-dimensional input vectors and A is equal to $Proj_X Y$. X and Y must be properly dimensioned in the calling program.

7.) Subroutine VSMUL (X, A, Y)

This subroutine multiplies a vector by a scalar. X is the three-dimensional input vector. A is the scalar, and Y is C x X. X and Y must be properly dimensioned in the calling program.

8.) Subroutine VCROSS (X, Y, Z)

This subroutine takes the cross-product (outer product) of two vectors. X and Y are the two three-dimensional input vectors, and Z is equal to X x Y. X, Y, and Z must be properly dimensioned in the calling program.

9.) Subroutine VXCOR (X, A)

This subroutine determines the i-th component of a vector. X is the three-dimensional input vector and A is equal to X • i. X must be properly dimensioned in the calling program.

10.) Subroutine VYCOR (X, A)

This subroutine determines the j-th component of a vector. X is the three-dimensional input

vector and A is equal to X · j. X must be correctly dimensioned in the main program.

11.) Subroutine VZCOR (X, A)

This subroutine determines the k-th component of a vector. X is the three-dimensional input vector and A is equal to X · k. X must be correctly dimensioned in the calling program.

12.) Subroutine MINV (D, DI)

This subroutine determines the inverse of a three-dimensional matrix. D is the three-dimensional input matrix and DI is equal to $[D]^{-1}$. If the inverse is not found, |D| equals zero, a warning is displayed on the system console. This warning is:

WARNING!! DETERMINENT = 0!! INVERSE MATRIX NOT FOUND!!

D and DI must be properly dimensioned in the main program.

13.) Subroutine MMULT (A, B, C)

This subroutine multiplies two matrices together. A and B are two three-dimensional matrices and C is equal to [A] x [B]. A, B, and C must be properly dimensioned in the calling program.

14.) Subroutine MSUM (A, B, C)

This subroutine sums two matrices together. A and B are two three-dimensional matrices and C is equal to [A] + [B]. A, B, and C must be properly dimensioned in the calling program.

15.) Subroutine MDIF (A, B, C)

This subroutine takes the difference of two matrices. A and B are the two three-dimensional input matrices and C is equal to [A] - [B]. A, B, and C must be correctly dimensioned in the calling program.

16.) Subroutine MSMUL (A, C, B)

This subroutine multiplies a matrix by a scalar. A is the three-dimensional input matrix, C is the scalar, and B is equal to C x [A]. A and B must be properly dimensioned in the main program.

17.) Subroutine VMULM (A, B, C)

This subroutine multiplies a matrix by a vector. A is the three-dimensional input matrix, B is the three-dimensional input vector, and C is equal to [A] x B (Note the order of the multiplication).

In this case, B is considered a column vector, so:

$$\begin{matrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{matrix} = \begin{matrix} A_{11} & A_{12} & A_{13} & B_1 \\ A_{21} & A_{22} & A_{23} & B_2 \\ A_{31} & A_{32} & A_{33} & B_3 \end{matrix}$$

Each of these subroutines are stored in separate source files and have individual object files. A list of the subroutines and their source and object file names is given below to aid the programmer in using each subroutine individually.

| Subroutine Name | Source File Name | Object File Name |
|---|---|---|
| VABS | &ABS | %ABS |
| VSUM | &SUM | %SUM |
| VDIFF | &DIFF | %DIFF |
| VSAME | &SAME | %SAME |
| VDOT | &DOT | %DOT |
| VPROJ | &PROJ | %PROJ |
| VSMUL | &SMUL | %SMUL |
| VCROSS | &CROSS | %CROSS |
| VXCOR | &XCOR | %XCOR |
| VZCOR | &ZCOR | %ZCOR |
| MINV | &MINV | %MINV |
| MMULT | &MMULT | %MMULT |
| MSUM | &MSUM | %MSUM |
| MDIF | &MDIF | %MDIF |
| MSMUL | &MSMUL | %MSMUL |
| VMULM | &VMULM | %VMULM |

C) The TDPKG Procedural File

There is a file named TDPKG which will load from disc and attach to the user's program in core memory all of the subroutines discussed above. To use this program the user types the following:

```
TDPKG
```

while in the FMGR system program. To attach all of the subroutines to a main program the user would type the following:

```
LG, 10
MR, 8  main program file name
TDPKG
```

the computer will respond by listing all the programs in the TDPKG system and:

```
/LOADR :   main program name   READY
/LOADR :  $END
```

indicating the user's program is ready to run. The user may then run the program with the usual RTE RUN command.

The user may be selective in what object files he desires to attach to his program. If he desires to load and attach only the subroutines VCROSS, MINV, and VMULM to his main program in core memory he could type;

```
: LG,  4
: MR,  %  main program file name
: MR,  %CROSS
: MR,  %MINV
: MR,  %VMULM
: RU,  LOADR, 99,,,,3
```

and then only these three routines would be attached and loaded.

This discussion assumes that the user has a good working knowledge of the FMGR and RTE system commands.

## VI. Summary and Conclusions

The two principal objectives of this thesis have been accomplished. One, the conceptual camera system has been developed to simulate a real camera thru a series of software programs that may be easily implemented by a programmer. The theoretical mathematical foundation for this system has been clearly analyzed and defined. The internal data base and programming structure have been logically set forth and completely discussed for the CCPS. Finally, the practical implementation and programming considerations need by the general user have been explicitly illustrated as an aid in program design.

Two, the data plotting capability of the Hewlett-Packard 1000 computer system has been expanded. So that

the general system user now has greater flexiblity in plotting data in a general programming situation. This increase in plotting flexibility has come with an easy to use plot package. This software plotting package has been fully discussed and structurally analyzed. Practical user implementations and system programming considerations have been illustrated to help initiate the programmer with using the package.

As an additional by-product of these two objectives a vector-matrix programming package has been developed. Originally this package was intended as an aid in the internal programming of the CCPS. Now this package can be used by a programmer in a versatile spectrum of applications. The subroutines in this package have also been set forth in a clear usable manner to ease in their use. A discussion and analysis also accompanies them.

Thus, the programming objectives of this thesis have developed a strong vantage point upon which more complex problems and analyses may be made in two pertinent areas of computer graphics for the Hewlett-Package 1000 computer system.

To sum up, it is appropriate to state what work is left to be done in the immediate future for the CCPS and and the PLOTT package.

For the CCPS the following considerations should command immediate attention:

1.) The integration of complex camera movements. Storage for the camera's movements in COORDS, ANGLE, and TRIG data arrays has already been allocated, so this consideration only requires writing subroutines to implement the following camera actions;

| | | |
|---|---|---|
| TILT | TRUCK | ZOOM |
| PAN | CRANE | |
| ROLL | DOLLY | |

Then the programmers must store the camera movements in the internal data structure, and integrate the camera action subroutines into the film image processing subroutine, IMAGE, immediately before the body motion subroutines. A detailed description of these complex camera actions can be found in bibliography reference number (4).

2.) Perspective Views. In the immediate future eight levels of gray scale will be available on the H.P. 1000's video display. This will allow the CCPS's perspective projection algorithm to incorporate a third variable to give the effect of depth perception. This

third variable, Zc, is an indicator of each body's depth displacement from the film plane (see Section II-D). Thus, a body which is further from the film plane than another body should appear less intense (dimmer) than the body which is closer. Thus, eight depth thresholds could be determined to produce this effect ranging from full intensity to no intensity (zero displacement) to infinite displacement from the film plane).

3.) Hidden line elimination. The complex problem of removing hidden lines is a difficult one to resolve. This problem could be attacked by applying one of two algorithms designed to eliminate hidden lines. These two algorithms, with trade offs in speed and memory usage, are:

a.) Robert's Solution* - very slow

b.) Warnock Algorithm* - faster but more complexity required.

For the PLOTT package the following considerations warrant immediate attention:

*Detailed discussion appears in reference number (6).

1.) Plotting more than one dependent variable in the GRID subroutine.

2.) Histogram plotting.

3.) Log - log and semi-log plotting.

# BIBLIOGRAPHY


1.) Bedford, F. and Dwivdei, T. Vector Calculus. 1st ed. New York: McGraw-Hill Book Co., Inc., 1970.

2.) Gottfried, B. Programming with Fortran IV. New York: Quantum Publishers, Inc., 1972.

3.) Hewlett-Packard Co. RTE Fortran IV. A Reference Manual Prepared by the Hewlett-Packard Co., 1977.

4.) Katzen, Joel E. "A Conceptual Three-Dimensional Camera for Computer Animation," Unpublished Master's Thesis, University of Pennsylvania, 1969.

5.) Kreyszig, E. Advanced Engineering Mathematics. 3rd ed. New York: John Wiley and Sons, Inc., 1972.

6.) Newman, W. and Sproull, R. Principles of Interactive Computer Graphics. 1st ed. New York: McGraw-Hill Book Co., Inc., 1973.

7.) Posdamer, Jeffrey L. "The Mathematics of Computer Craphics", Byte Magazine, vol. 3, no. 9 (1978), pp. 22-39.

8.) Stromberg - Carlson Corp. Programmers' Reference Manual, S-C 4020, Computer Recorder. Document No. 9500056, 1964.

# Appendix A

Since no source program listings may be incorporated into a thesis, the reader is referred, for all programs in Appendices A, B, and C, and in the main body of the thesis, to the complied computer printouts of all the source lists of graphics programs written by this author. These complied listings are available in the H. P. 1000 computer room or from the H. P. 1000 system manager.

A sample main program for the CCPS may be found in the complied source lists under the program name MAIN and the source file name &MAIN. This program illustrates what is required in a typical CCPS main program (see CCPS Programming Techniques - Segmented Programs). As can be seen, this program jumps to the first example CCPS segmented program SGMTO.

SGMTO demonstrates what steps the programmer must incorporate into his first CCPS segmentation. As illustrated, this program is a type 5 program. It first defines the line connection scheme for three bodies via the MSEQ (4, 15, 5) array. These bodies are a pyramid and two cubes. The line connection scheme is then stored in the internal data structure via the CALL SEQ (MSEQ) statement. Then the various body motions are defined for

each body using any one of the six body motion subroutines. As shown for body 1, a combination of three motions are described, two translational and one rotational. The body is to move in the Z direction, of its body co-ordinate system, 40 feet starting at frame 2 and ending at frame 6. It must simultaneously move in the Y direction - 40 feet from, again, frame 2 to frame 6. Finally, the body must rotate from the X to the Z axis, a total of 45 degrees beginning at frame 2 and stopping at frame 3. Like motions have been described for bodies 2 and 3. SGMTO then jumps to the second segment, SGMTT, thru the CALL EXEC statement.

SGMTT is a program demonstration of what must be included in every second segmentation of the CCPS. In this type 5 program, the three bodies are first defined thru DATA statements. Body 1 (X, Y, Z) and body 3 (XXX, YYY, ZZZ) are cubes identical in size and body 2 (XX, YY, ZZ) is a pyramid. Then the filming loop is set up to draw the bodies on the film plane. Note that two film loop processes are used in SGMTT, separated by a PAUSE statement. The first loop films the motions of bodies 1 and 2. The effect is that, body 1 rotates slightly and moves across the screen from the upper left hand corner to the lower right hand corner. Body 2

rotates and quickly moves directly back into the film plane almost to infinity along the Zc axis. The program then pauses to view the results. The programmer resumes execution via a GO statement. The screen is erased and the filming of loop 2 occurs. In this loop, body 3 is rotated about its center with simultaneous $\phi$ and $\theta$ angular displacements. The CALL ERASE statement causes each successive frame to be erased once drawn. Thus, the total effect is a cube rotating in two directions about its center.

## Appendix B

The reader is referred to the compiled source listings of graphics programs for the following discussion.

To demonstrate the use of the PLOTT package an example program has been written called DEMO. It is stored in the source file, &DEMO. The program produces two plots of an inverted (sin X)/X curve, between -3 and 3 . The first plot is a point plot and the second is a line plot. The plots are produced by the use of the GRID and TITLE subroutines. All parameter definitions for both subroutines are discussed in the main body of this thesis. A PAUSE statement appears between the first and second plots to give the user a chance to view the first plot before the second is drawn. Program execution may be resumed via a GO statement.

# Appendix C

The reader is referred to the compiled source listings of graphics programs for the following discussion. As a demonstration of the vector-matrix package in the main body of this thesis an example program titled DEMO1 has been written. This program incorporates some of the vector manipulations possible with this package. The routine itself determines the distance from a plane to a point. The plane is defined by three points as illustrated and the point is also given. The routine displays the i-th, j-th, and k-th components of the normal to the plane, $\overline{N}$, and displays the distance of the point to the plane, both in formatted statements. The mathematical solution for this problem may be found in bibliography reference (1.), p. 194.

## Vita

James William Chamberlain was born on August 1, 1954 in Johnson City, New York. He is the first child of James Lesile and Audrey Chamberlain. He attended high school at Binghamton Central High School and graduated in June, 1972. Later, Jim attended Broome Community College in Binghamton, New York and graduated with an Associate of Applied Science degree in Electrical Technology on May 23, 1975. He then enrolled at the State University of New York at Binghamton and graduated from there with a Bachelor of Technology degree in Electrical Technology on May 23, 1976. He is currently pursuing a Master of Science degree at Lehigh University in Electrical Engineering while working for the school as a Teaching Assistant.

His extra-curricular activities include bicycling, collecting antique radios, and is an active Ham Radio operator (WA2HFO) with a General class license.