ADA027419

AVIATION RESEARCH LABORATORY

INSTITUTE OF AVIATION
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

TECHNICAL REPORT

A VERSATILE COMPUTER-GENERATED DYNAMIC FLIGHT DISPLAY

BRUCE A. ARTWICK

ARL-76-5/ONR-76-1
MAY 1976

DDC
RECEIVED
JUL 21 1976
B

Contract: N00014-76-C-0081
Work Unit Number: NR 196-133

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

PREPARED FOR

ENGINEERING PSYCHOLOGY PROGRAMS
OFFICE OF NAVAL RESEARCH

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ARL-76-5/ONR-76-1

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): A VERSATILE COMPUTER-GENERATED DYNAMIC FLIGHT DISPLAY,

5. TYPE OF REPORT & PERIOD COVERED: Technical Report

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): Bruce Arthur Artwick

8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0081

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Aviation Research Laboratory
Institute of Aviation
University of Illinois, Savoy, Illinois 61874

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Office of Naval Research
Engineering Psychology Programs

12. REPORT DATE: May 1976

13. NUMBER OF PAGES: 63

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): NR-196-133

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

Flight displays	Aviation Instrumentation	Landing Displays
Contact Analog Displays	Predictor Displays	Image Synthesis
Attitude Displays	Three Dimensional Graphics	Navigational Aids
Display Programs	Pursuit Displays	Attitude Displays
Computer Graphics	Cockpit Instrumentation	Flight Simulation

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

This report describes a real-time, dynamic, computer-driven visual display program which is written in the Fortran programming language. Versatility, efficiency and ease of use are stressed in the development, resulting in an easy to interface to dynamic display which can be implemented economically with a bare minimum of graphics hardware and a sixteen bit mini-computer which has Fortran capabilities. Modular structure is stressed and speedup methods are discussed including the

NEXT PAGE

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

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

406171

use of a matrix multiplier. A unique frame synthesizing feature is described in detail. Sample data base structures and display images conclude the report.

ACCESSION for
NTIS White Section ☑
DDC Buff Section ☐
UNANNOUNCED ☐
JUSTIFICATION.....
.....
BY.....
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

AVIATION RESEARCH LABORATORY

Institute of Aviation

University of Illinois at Urbana-Champaign
Willard Airport
Savoy, Illinois
61874

Technical Report

ARL-76-5/ONR-76-1

May 1976

A VERSATILE COMPUTER-GENERATED DYNAMIC FLIGHT DISPLAY

Bruce Arthur Artwick

Prepared for

ENGINEERING PSYCHOLOGY PROGRAMS

OFFICE OF NAVAL RESEARCH

Contract: N00014-76-C-0081
Work Unit Number: NR 196-133

DISTRIBUTION STATEMENT: APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

CONTEXT

The Aviation Research Laboratory of the University of Illinois is investigating synthetic imaging displays and computer-augmented flight control for the Office of Naval Research. Mr. Gerald Malecki, Assistant Director of Engineering Psychology Programs, is the technical monitor of the research. Professor Stanley N. Roscoe was the principal investigator during the initial phase of study and experimental apparatus development covered by this report. Professor Robert C. Williges is serving as principal investigator for the continuing effort while Professor Roscoe is on academic leave during 1975-76.

The research is directed toward (1) the isolation of minimum sets of visual image cues sufficient for spatial and geographic orientation in the various ground-referenced phases of representative flight missions, (2) the generation and spatially integrated presentation of computed guidance commands and fast-time flight path predictors, and (3) the matching of the dynamic temporal relationships among these display indications for compatibility with computer-augmented flight performance control dynamics, both within each ground-referenced mission phase and during transitions between phases. The investigative program draws selectively upon past work done principally under ONR sponsorship or partial sponsorship, including the ANIP and JANAIR programs.

Program Progress and Plans

During Phase I of the current contract, the Aviation Research Laboratory systematically investigated the relationships between the movement of the controls and the response of the airplane and demonstrated substantial improvement in pilot performance as a consequence of their reorganization. By the completion of Phase I, all planned control modifications, specifically the digital control system have been incorporated into the GAT-2 simulator. No additional work on this task is contemplated for the initial year of Phase II.

To study experimentally the effectiveness of alternate sets of visual cues, the Aviation Research Laboratory has developed a highly versatile computer-generated display system to present dynamic pictorial images either on a head-down, panel-mounted CRT or on a head-up television projection to a large screen mounted in front of the pilot's windshield on the Link GAT-2 simulator. Due to the great flexibility of the pictorial display, visual cues and flight status information can be manipulated experimentally. Experimentation to isolate the visual cues sufficient for approach and landing is in progress.

The incorporation of predictive indications of successive future states is currently under investigation during Phase II. Experiments will be conducted to determine the number and temporal spacing of flight path predictors to be integrated into the forward-looking flight view. Determination and software implementation of command guidance symbology compatible with the synthetic forward-looking contact

analog and predictive flight path presentations will also be undertaken. It is the ultimate objective of this program to develop, during the second year of Phase II, a reconfigured cockpit with integrated sensor and computer-generated imaging displays and computer-augmented controls.

TABLE OF CONTENTS

	Page
INTRODUCTION	1
APPROACH	3
Language Considerations	3
Modularity in Programmming	3
Organized Buffer Structure	4
Display Algorithm	4
Periodic Updating Considerations	5
PROGRAM FEATURES	6
Plug-in Replacement Modules	6
Communication Through Common Blocks	6
Universal Interface Capabilities	6
External Program Interaction	6
Navigational Map Overlay Feature	7
Nonclipped Projection	7
Coordinate System	8
Variable Field of Vision	8
Display Filtering	10
PROGRAM STRUCTURE	12
Buffer Structure	12
Subroutine Calling Conventions	15
Numerical Structure	15
Control Program	16

GRAPHICS SUBROUTINES 19
IMX Subroutine 19
IMATRX Subroutine 21
ISYNTH Subroutine 24
IGEN Subroutine 28
ICODE Subroutine 35
PUSHER Subroutine 35
IPROJ Subroutine 38
NAVIGATIONAL SUBROUTINES 40
IGAT Subroutine 41
INAV Subroutine 41
COMPUTATIONAL SUBROUTINES 45
IPYRA Subroutine 45
IMUL Subroutine 45
INITIALIZATION 47
INIT Subroutine 47
Block Data Subroutine 47
DISPLAY PERFORMANCE 52
CONCLUSION 54
REFERENCES 56
APPENDIX A 57
APPENDIX B 60

LIST OF TABLES

Table Page

1 Program performance 53

LIST OF FIGURES

Figure		Page
1	Coordinate system	9
2	Windowing	9
3	Buffer formats	14
4	Buffer structure	14
5	Display loop sequencing	17
6	Main program	18
7	IMX subroutine	20
8	Rotation matrix	22
9	Translation and windowing matricies	22
10	IMATRX subroutine	23
11	ISYNTH subroutine part a	26
12	ISYNTH subroutine part b	27
13	IGEN flow chart part a	29
14	IGEN flow chart part b	30
15	IGEN flow chart part c	31
16	IGEN subroutine part a	32
17	IGEN subroutine part b	33
18	IGEN subroutine part c	34
19	ICODE subroutine	36
20	PUSHER subroutine	37
21	IPROJ subroutine	39
22	IGAT subroutine part a	42
23	IGAT subroutine part b	43

24 INAV subroutine 44

25 IPYRA subroutine 46

26 IMUL subroutine 46

27 INIT subroutine part a 48

28 INIT subroutine part b 49

29 Block data subroutine 49

INTRODUCTION

During flight simulation experiments and pilot training it is often desirable to project an accurate representation of the outside world over which a simulator is theoretically flying. In the past, such methods as prefilmed movies and "flying" television cameras have been used and in recent times computer generated dynamic graphics have been attempted. At the present time, real time computer graphics can not compete with some of the earlier methods on the basis of complexity and realism of display but with the performance of digital hardware increasing and the price decreasing, the computer may soon equal or surpass other display methods. There exists a need for fast three dimensional graphics projection programs to be used with computer hardware to create these dynamic displays. To be effective, this software must not only project images quickly and accurately but must also be versatile, transportable, and well organized so others can understand, use, and experiment with it. Reliance on expensive graphics hardware must be avoided to keep the simulation cost effective.

The display program described herein is an attempt to meet the above criteria and to form a program which can be used in full or in part. A number of old and proven graphics techniques as well as a synthesizing procedure newly developed for this program are employed. Modularity concepts are used throughout allowing execution time speed-ups from the replacement of Fortran modules with assembly language equivalents. This modularity also permits easy interfacing to auxiliary

graphics hardware and software such as matrix multipliers and predictor displays.

Having been developed on a Digital Equipment Corporation PDP 11, the interfacing of this program to the Aviation Research Laboratory's Raytheon 704 computer and Singer Link Gat 2 simulator will be used as an application example and evaluation of software transportability. Development and application were performed on small inexpensive computers but the program is designed to be upward compatible to larger machines where dramatic performance increases and display realism should result.

APPROACH

LANGUAGE CONSIDERATIONS. Since high speed execution is of prime importance in a real time simulation, assembly language seemed to be the ideal language for this program. Previous display programs, however, have shown that the resulting assembly language code is not very transportable or flexible and is nearly impossible for anyone but the original programmer to understand. The most common scientific programming language in use today is Fortran. A good Fortran compiler can produce relatively time efficient code thus making Fortran the choice of language for this program.

A number of steps were taken to offset the slow execution time of the Fortran display program, the first being integerization. Floating point arithmetic uses twice the memory and requires about three times the execution time of integer mode arithmetic. Integer arithmetic is also compatible with integer array processors such as hardware matrix multipliers which are only capable of operating in integer mode.

MODULARITY IN PROGRAMMING. A very modular program structure was chosen in order to simplify experimentation and application and to make the program easy to understand. A short main program consisting mostly of calls to symbolically named subroutines allows a person unfamiliar with the program to easily understand the display sequence.

Many array processing subroutines are used to transform, clip, and project images therefore a common calling structure was adopted to provide simple interchangeability of array processing functions if desired. Experimenting with various versions of the main program was,

for this very reason, greatly simplified. Substituting one type of projection routine for another, for example, was simply a matter of changing the subroutine call but leaving the string of dummy variables unchanged.

ORGANIZED BUFFER STRUCTURE. The many buffers needed to process large quantities of graphics information can make a graphics program rigid or extremely flexible depending on the buffer structure. The buffers in this program were arranged about the following guide lines.

1) Buffers must be easily concatenated by the array processors.

2) Buffers with the same type of data must always have the same data structure.

3) Index data and display data should be in separate buffers whenever possible.

This buffer structure, along with the common calling conventions, make this program easy to work with.

DISPLAY ALGORITHM. The graphics techniques used to convert a three dimensional data base into a two dimensional screen projection have been well developed and established. Start and end points which define lines are multiplied by a four by four transformation matrix to translate and rotate them into the proper reference frame relative to the observer's eye. A modified, highly efficient version of the Cohen and Sutherland clipping algorithm is then employed to eliminate any off screen lines and to clip lines which are partially on the screen. Finally, a pyramid projection is performed projecting the start and end points and thus the lines onto a display screen. To increase projection

speed a frame synthesizing method is used. Interpolations are made between display points in consecutive frames. Since only addition is required to add the small delta values (see the synthesizer subroutine explanation), frame synthesis time is very low. A much smoother display results. The subroutine explanations further describe graphic projection and transformation functions.

PERIODIC UPDATING CONSIDERATIONS. In past display programs it has been said that far away objects seem to move slower and thus need to be updated less often than close objects. This may be true for translational movement but the argument does not hold when rotation is employed. When an aircraft is banking, for example, the most distant object (the horizon perhaps) is rotating at the same rate as the closest object. If far away points are only updated periodicically, geometric distortions can result as far away lines lag behind in the rotation.

The easily concatenated buffers used in this display program permit periodic updating of points through proper arrangement of the main program. This feature may have some use in translation and very slow rotation applications but in general, and in the main program shown, periodic transformations are avoided in order to preserve realism.

PROGRAM FEATURES

PLUG-IN REPLACEMENT MODULES. In order to retain the high execution speed of assembly language, many of the complex but repetitive arithmetic functions are performed by calling very small subroutines. These subroutines are referred to as plug-in replacement modules. Great execution time speedups result from replacing the modules with their Fortran callable assembly language equivalents which can be written especially for the computer being used.

COMMUNICATION THROUGH COMMON BLOCKS. Subroutines and the main program only use data transmission through dummy variables when absolutely necessary. Symbolically named common blocks are instead used to simplify the main program and make it easy to understand.

UNIVERSAL INTERFACE CAPABILITIES. A program such as this can find many applications in aviation and other fields making it imperative to have some sort of simple modification which can be made to interface the display program to various forms of simulators and controls. The IGAT subroutine performs this task by establishing a common convention for passing display control information to the display program. Rewriting this short subroutine for the simulator being used allows the rest of the software to become compatable with that simulation.

EXTERNAL PROGRAM INTERACTION. Simultaneously running programs, such as predictor displays, can place objects to be displayed into the display data base using the auxiliary display buffers provided for this purpose. The program using these buffers can specify which type of transfromations are desired on the presented data. Lines to appear

stationary in space require translation and rotation transformations while only a rotational transformation is needed to put objects in a reference frame relative to the aircraft's position. Any combination of pitch, bank, heading, x, y, or z can be specified as a transform making sixty-four reference frames available to the external user.

NAVIGATIONAL MAP OVERLAY FEATURE. Integer mode arithmetic limits data base size to plus or minus 32767 units. Through the use of variable scale factors these units can represent any dimension desired. The scale of one foot per unit may be good for flight at pattern altitude while a scale of three inches per unit may be more appropriate for final approach where a finely detailed and precise picture is desired. Before the display program is run, maps can be set up which have these scales. At run time a new map can be overlayed into the data base and the projection scale factors changed simultaneously resulting in a smooth map transition. This feature combines the large range of a coarse map with the precision of a fine map.

Maps of the same scale can be used to extend the display data base's effective size. Using appropriate x, y and z offsets, maps can be strung together to provide an almost limitless world size. Far away objects can be eliminated or simplified on appropriate maps to give desired fade out effects and eliminate horizon brightness due to section line crowding at low altitudes.

NONCLIPPED PROJECTION. The process of clipping lines is very time consuming. In order to increase picture complexity and maintain high execution speed a nonclipping projection method has been devised.

Small objects, such as runway markers and vertical poles, which tend to leave the screen all at once, benefit from this routine which simply eliminates the whole line from the screen if any part of that line falls off the screen. This method is not used on large lines which must be clipped.

COORDINATE SYSTEM. In order to keep the program well organized, a universal coordinate system has been adopted. Figure 1 shows this convention.

VARIABLE FIELD OF VISION. Projection geometry is affected by how far away from the viewing window (projection screen) the observer is and by the size of the window (screen size). A variable field of view permits this program to be used with many observer and display screen configurations. The field of view can effectively be changed by widening the data base and retaining a forty-five degree viewing pyramid. This data base distortion takes place after the rotation and translation transformations have been completed. The IMATRIX subroutine concatenates the window transform to the rotation and translation transform resulting in the need for only one data base transformation.

Although the standard procedure for field of vision control is data base multiplication along the x and y axes for viewing angles of less than forty-five degrees to each side, z axis division was chosen. Integer overflow of the data base is thus avoided. This limits the view to a square window. Through program modification, however, an appropriate amount of x axis or y axis division can accommodate a rectangular window projection. Figure 2 shows windowing.

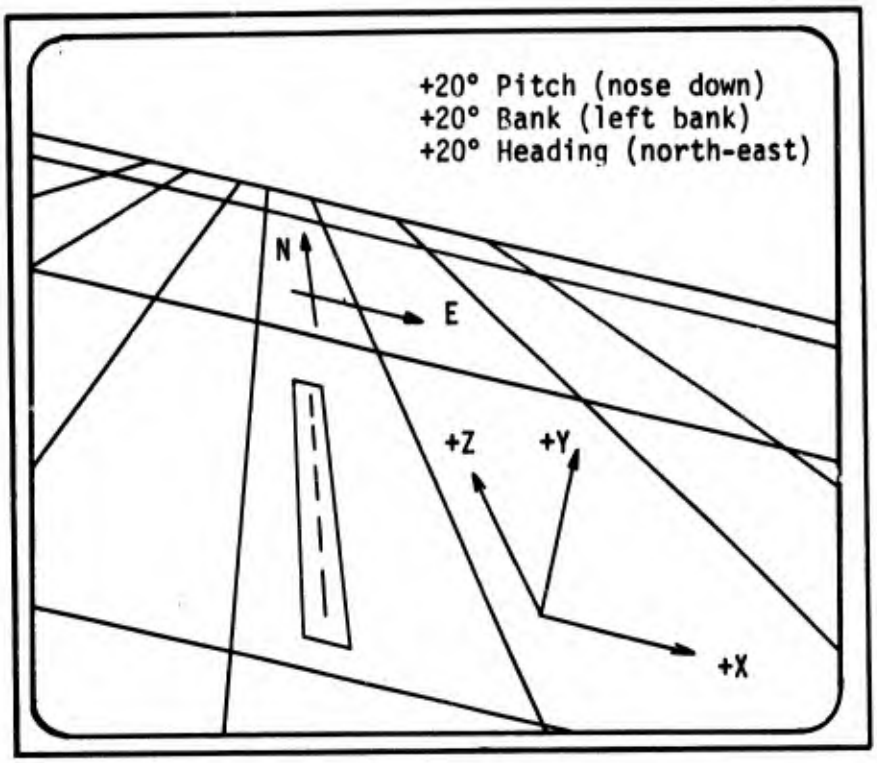
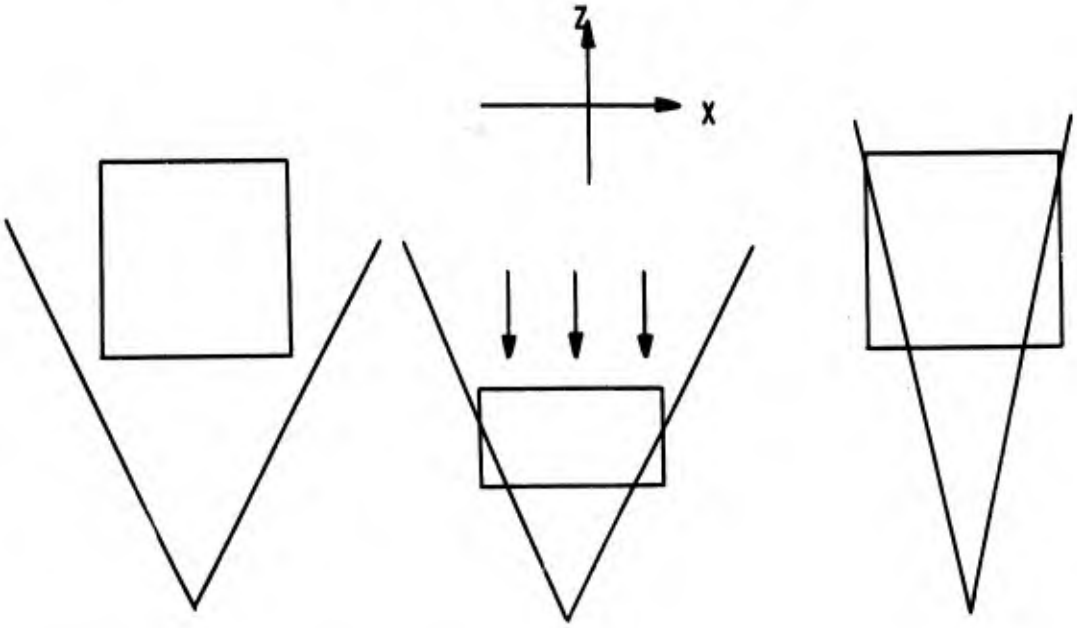


Figure 1. Coordinate system.



a) Data base image within the viewing pyramid.

b) Windowing performs Z dimensional compression.

c) Effective field of view.

Figure 2. Windowing.

IVIEW is a fractional constant used in the windowing transform to control z axis scaling. Window geometry defines this value as:

IVIEW = Tangent (Field of View)

where field of view is the angle from the z axis. This angle can be submitted during program initialization. IVIEW is calculated by the initialization subroutine.

DISPLAY FILTERING. Start and end points of lines tend to move in jerky, discrete steps due to the integer data base format. At great distances from points this effect is not noticeable but when lines fall near the base of the viewing pyramid, annoying distortion results. The IGEN and ISYNTH subroutines have methods which reduce this bad effect.

The IGEN subroutine must convert points' coordinates into floating point mode in order to clip the lines they represent. If it is found that an end point is near the base of the viewing pyramid, the x, y and z coordinates are multiplied by a scaling constant thus reducing truncation error upon integer conversion and projection. This action is referred to as accuracy scaling.

Screen point hysteresis is used in the ISYNTH subroutine. A projected line's start and end points are given threshold values (hysteresis constants) in the plus and minus x and y directions which they must cross in order to produce screen point movements. Movements in one continuous direction will be smooth as one threshold will always be exceeded and the screen line will move accordingly. Small jerky movements in both directions, however, will not be reflected on the screen if the projected line errors are smaller than the hysteresis

constant. At high altitudes, hysteresis has no beneficial effect and can be turned off to save computation time. In cases where jerky motion is desired (air turbulence effects for example), hysteresis can be turned off. The hysteresis screen point filter, which is a threshold type filter, was chosen over screen point averaging due to its superior rapid response characteristics.

PROGRAM STRUCTURE

The display program consists of three major sections; the display driver, navigator, and the control program.

Graphic subroutines comprise the driver. Upon the receipt of a data base and space coordinates (x,y,z,pitch,bank,heading), the driver performs all necessary transformations and synthesizing to produce an accurate projection on a display screen.

The navigator keeps track of the simulator's position, handles data base overlays and offsets, and is used to interface a simulator to the display driver.

The control program is the main program of the Fortran display program. It consists of many calls to navigation and display driver subroutines and controls their order of execution.

BUFFER STRUCTURE. A three dimensional data base consisting of lines represented by start and end points is submitted to the display driver. A two dimensional screen projection consisting of start and end points results. Many transformations occur to produce the results and buffers are used to store intermediate results.

Since four by four matrix multiplication is used to transform 3D points into the proper reference frame, a four by one vector is used to specify a 3D point. Another buffer is used as an index to these points and the index code tells whether the point is a start or continue point. The index therefore applies to the points even after the transformation has been completed. A positive index value represents a start point. The first element of the index buffer will always be a start point and

will thus always be positive. The first index element serves a dual purpose. The 3D buffer length (number of 3D points) is submitted as the first index element. The index can be used in two ways. The end of the data base can be sensed by an index code of zero or the first index element can be used as a process length. Hardware which requires a transfer address and length is easily accommodated with this form of indexing.

A combined data and index buffer is used to define 2D lines. A display code, a start point (x and y screen value), and an end point produce five array entries. The code indicates whether the line is to be displayed or not.

A delta value corresponding to each 2D point is stored in the same type of buffer format as 2D points. The code, however, has no significance and is simply used as a filler value so 2D points and corresponding delta values have the same array subscript numbers.

The total number of buffers used is determined largely by the control program and the program application. Figure 4 shows the buffer structure for the control program used for the aircraft simulation. Figure 3 illustrates the buffer formats just described.

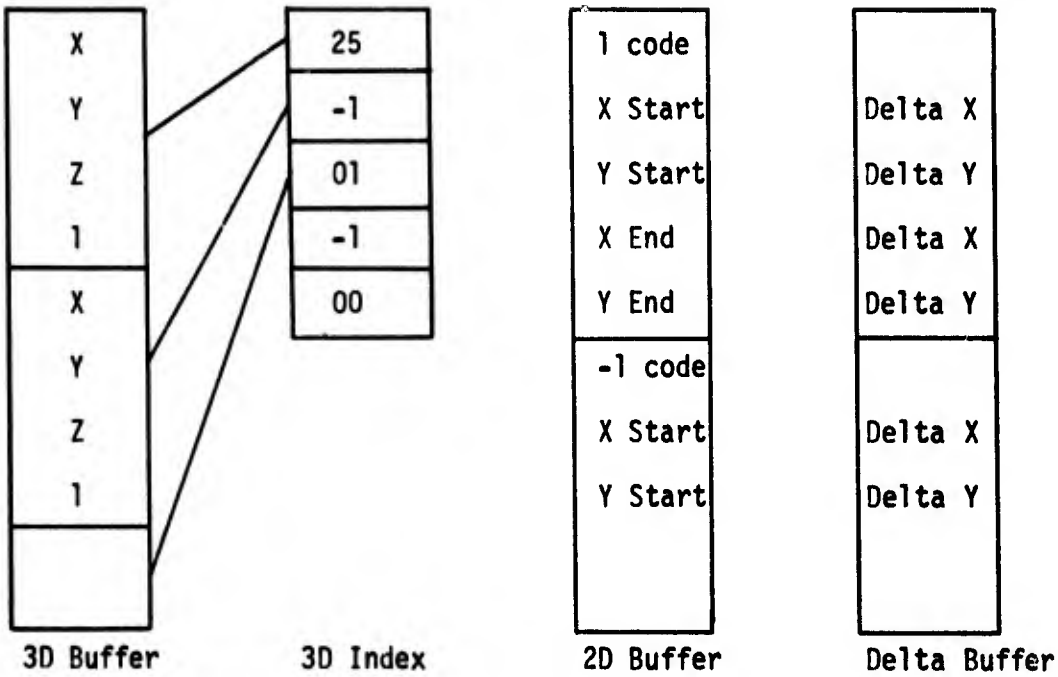


Figure 3. Buffer formats.

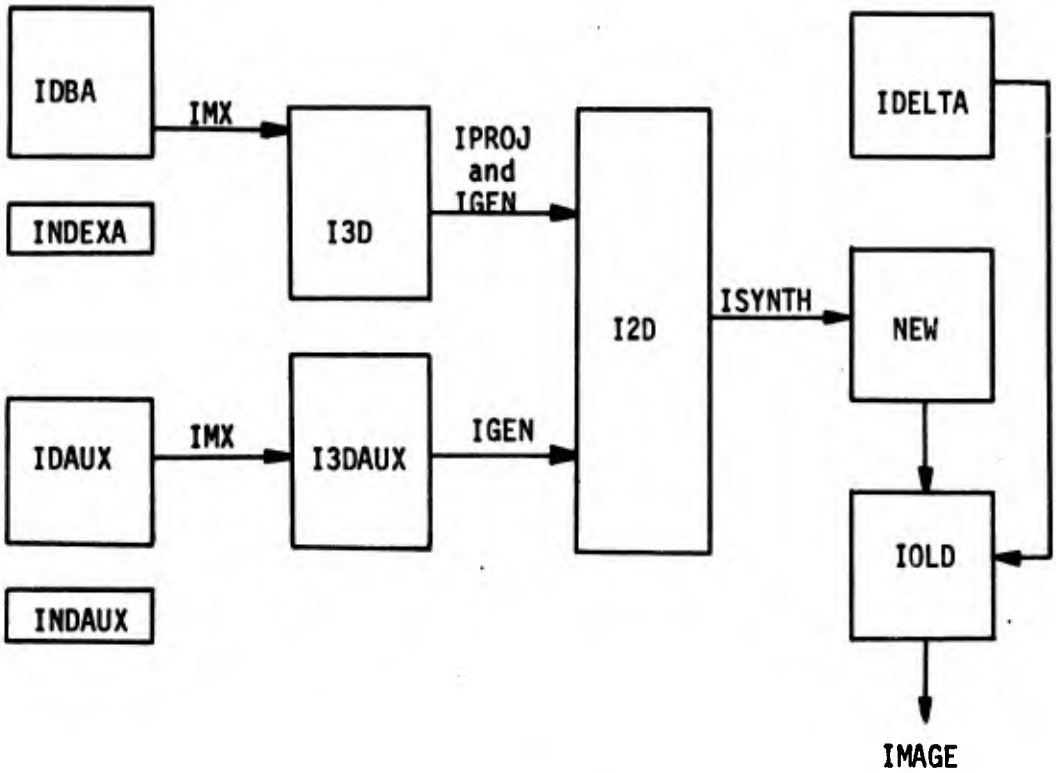


Figure 4. Buffer structure.

SUBROUTINE CALLING CONVENTIONS. In order to make array processing subroutines easily usable, array processing subroutines have been given similar calling structures which follow the following conventions.

```
CALL SUBROUTINE (in,index,out,k,m,n)
```

where:

```
in    = input array
index= index to input array and possibly output array
out   = output array of subroutine results
k     = input array pointer
m     = index array pointer
n     = output array pointer
```

The called subroutine starts operating on arrays at the pointer locations and leaves the pointers pointing at the last element processed plus one upon return.

NUMERICAL STRUCTURE. For versatility and execution time speedups sixteen bit integer, two's complement arithmetic is used. To retain precision, multiplication is accomplished by calling subroutines which have thirty-two bit product capabilities. The Fortran versions of these routines convert data to floating point to do this while their assembly language equivalents generally use the thirty-two bit capabilities of the computer's multiplication hardware.

Fractions are expressed by positioning the binary point to give fifteen bits of fractional precision and one sign bit.

Many calculations involve the multiplication of a trig function by a nonfractional unit. A subroutine has been provided to perform the operation by taking the sixteen most significant bits of the thirty-two bit product of the fraction multiplied by the integer.

Sine and cosine lookup tables are generated upon program initialization resulting in the speedup of trigonometric operations.

CONTROL PROGRAM. The control program shown in figure 6 can be used as-is but is meant to be more of a guideline as to what can be done with the display driver and navigation subroutines. The circular flow chart which accompanies it (figure 5) provides a time domain representation of the processes being performed in the display program.

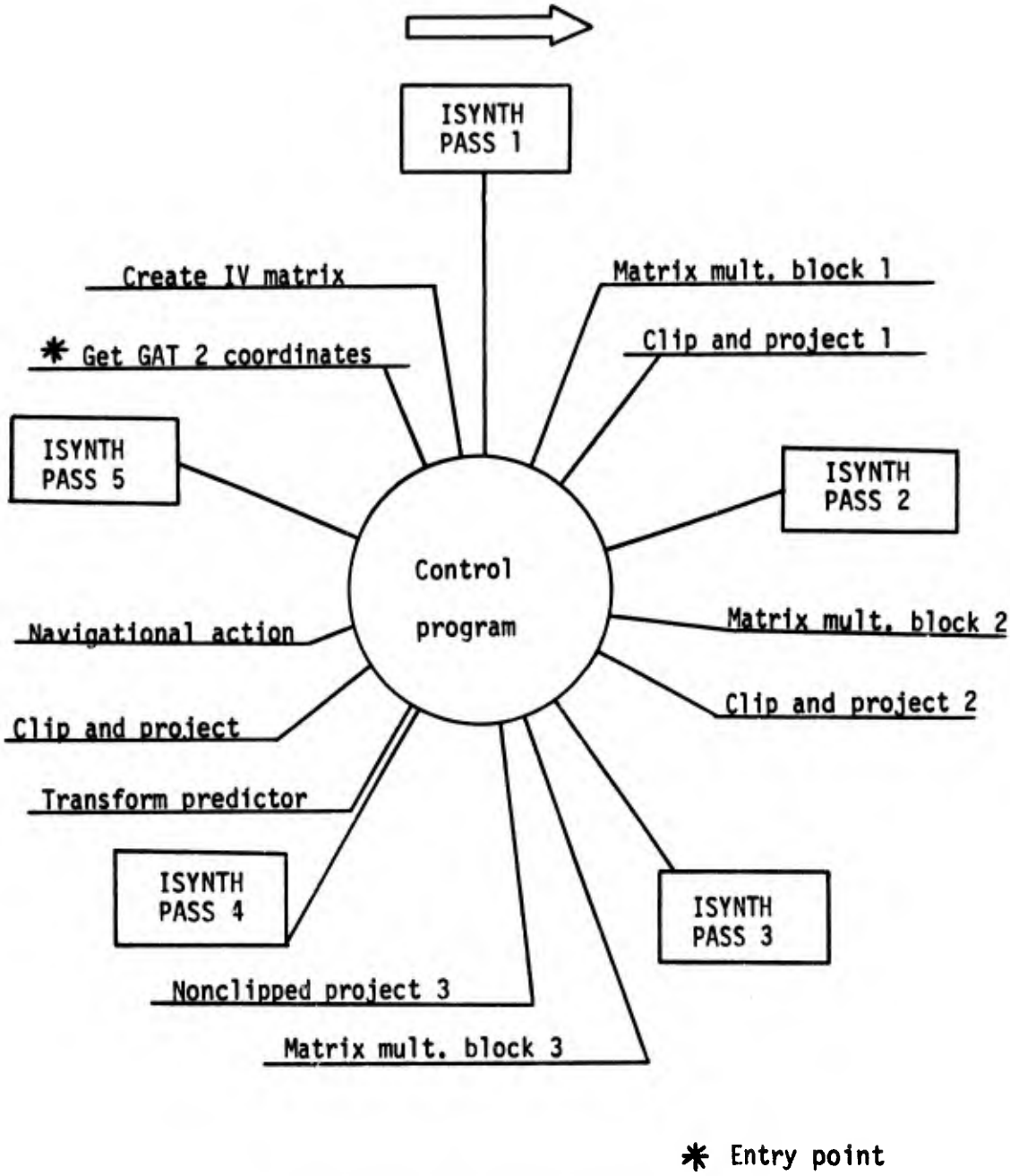


Figure 5. Display loop sequencing.

```
c     main program*****************************************
c     display loop sequencing is controlled by this program.
c     main program*****************************************
c
c     data assignments.
      common /base/  idba(400),indexa(100),idaux(80),indaux(20)
      common /base3/ i3d(400),i3daux(80)
      common /mtx/   iv(16),iview
      common /proj/  i2d(400)
      common /synth/ new(400),iold(400),idelta(400),nframe,ihyst
      data ipass1,ipass2,ipass3,ipass4,ipass5/-1,0,0,0,1/
      data model,mode2/1,2/
c
c     control program, initialization section.
      call init
c
c     control program, display loop section.
1     call igat
      call imatrx
      k=1
      m=1
      n=1
      kk=1
      mm=1
      nn=1
      call isynth (ipass1)
      call imx (idba,indexa,i3d,k,m,n,model,iv)
      call igen (i3d,indexa,i2d,kk,mm,nn)
      call isynth (ipass2)
      call imx (idba,indexa,i3d,k,m,n,model,iv)
      call igen (i3d,indexa,i2d,kk,mm,nn)
      call isynth (ipass3)
      call imx (idba,indexa,i3d,k,m,n,model,iv)
      call iproj (i3d,indexa,i2d,kk,mm,nn)
      call isynth (ipass4)
      k=1
      m=1
      n=1
      kk=1
      mm=1
      call imx (idaux,indaux,i3daux,k,m,n,mode2,iv)
      call igen (i3daux,indaux,i2d,kk,mm,nn)
c     inav is not used in this version of the display program.
c     this is where it would appear if it was.
c     call inav
      call isynth (ipass5)
      go to 1
      end
```

Figure 6. Main program.

GRAPHICS SUBROUTINES

The graphics subroutines are now presented. A more detailed description of many of the graphics techniques can be found in reference 5.

IMX SUBROUTINE. This array processing subroutine multiplies the input array which consists of four by one vectors, each representing a point in space, by a four by four transformation matrix (IV) and stores the resulting four by one vectors in the output array. Standard pointer conventions are followed in the subroutine call and an index code of zero is recognized as the last vector to be multiplied. The resulting output is the transformed data base which represents points in the viewer's reference frame. Figure 7 shows the IMX subroutine.

Rapid matrix multiplication can be accomplished by replacing this subroutine with a hardware matrix multiplier setup and initiate routine if such hardware is available. In this event, transfer address information can be taken from the dummy variable string and the first index array value can be used as a length parameter.

The IMODE variable determines whether a full or partial transform will be made.

MODE1 OPERATION- The transformations specified by the complete IV matrix are performed. Pitch, bank, heading, x, y, and z are considered.

MODE2 OPERATION- Only the rotational transformations are performed. This corresponds to the aircraft's reference frame as opposed to the ground reference frame in the MODE 1 case.

```
      subroutine imx (idb,index,i3d,k,m,n,mode,iv)
c     imx-----------------------------------------------------
c     the display points are multiplied by the transformation
c     matrix.
c     imx-----------------------------------------------------
c
c     data assignments.
      dimension idb(400),index(100),i3d(400),ivsave(3),iv(16)
c
c     mode 2 decision and matrix modification.
      if (mode.eq.1) go to 4
      do 2 i=1,3
      ivsave(i)=iv(i+12)
2     iv(i+12)=0
c
c     matrix multiplication.
4     jx=idb(k)
      jy=idb(k+1)
      jz=idb(k+2)
      do 5 j=1,3
      j1=iv(j)
      j2=iv(j+4)
      j3=iv(j+8)
      j4=iv(j+12)
      ja=imul(jx,j1)
      jb=imul(jy,j2)
      jc=imul(jz,j3)
5     i3d(n+j-1)=ja+jb+jc+j4
      k=k+4
      m=m+1
      n=n+4
      if (index(m-1).ne.0) go to 4
6     if (mode.eq.1) return
c
c     mode 2 matrix restore.
      do 7 i=1,3
7     iv(i+12)=ivsave(2)
      return
      end
```

Figure 7. IMX subroutine.

IMATRX SUBROUTINE. IMATRX creates an integer transformation matrix from the coordinates provided in the REF common block (IX,IY,IZ,IPIT,IBNK,IHDG). Fast generation of sines and cosines of angles are accomplished through lookup tables. The tables range from 1 to 360 degrees in one degree steps.

A rotation matrix is created using standard graphics equations derived from geometric principles. Pitch, bank and heading are considered. All trigonometric calculations use fifteen bit fractional arithmetic. The nine rotational elements of the transformation matrix are also expressed in this form.

A translation matrix is created by effectively adding the reference position (IX, IY, IZ) to the data base elements. The translation matrix is concatenated with the rotation matrix resulting in a complete transformation matrix. In the transformational sense, the translation is performed first (positioning the aircraft in the proper place), then the rotation is performed (rotating the world about the aircraft giving the impression of pitch, bank, and heading). Figures 8 and 9 show the rotation and translation matricies.

Field of view is corrected for by multiplying the translated and rotated data base by the window transform. The IVIEW constant is submitted by the program user through a data base entry or by default upon initialization. The transformation matrix is appropriately modified to perform field of vision corrections through dimensional compression upon data base multiplication. The IMATRX subroutine is shown in figure 10.

Cos H Cos B + Sin H Sin P Sin B	-Cos H Sin B + Sin H Sin P Cos B	Sin H Cos P	0
Cos P Sin B	Cos P Cos B	-Sin P	0
-Sin H Cos B + Cos H Sin P Sin B	Sin H Sin B + Cos H Sin P	Cos H Cos P	0
0	0	0	1

P= Pitch B= Bank H= Heading

Figure 8. Rotation matrix.

1	0	0	0
0	1	0	0
0	0	1	0
X	Y	Z	1

a) Translation matrix.

1	0	0	0
0	1	0	0
0	0	IVIEW	0
0	0	0	1

b) Windowing matrix.

Figure 9. Translation and windowing matricies.

```
      subroutine imatrx
c
c     imatrx-----------------------------------------------
c     the transformation matrix, iv, is created.
c     imatrx-----------------------------------------------
      common /ref/ ix,iy,iz,ipit,ibnk,ihdg
      common /mtx/ iv(16),iview
      common /trig/ isin(360),icos(360)
c     matrix calculations rotation section.
      ia=isin(ipit)
      ib=isin(ibnk)
      ic=isin(ihdg)
      id=icos(ipit)
      ie=icos(ibnk)
      ip=icos(ihdg)
      ig=imul(ip,ie)
      ih=imul(ic,ib)
      ii=imul(ip,ib)
      ij=imul(ic,ie)
      ik=imul(ih,ia)
      il=imul(ij,ia)
      im=imul(ii,ia)
      in=imul(ig,ia)
      iv(1)=ig+ik
      iv(2)=-ii+il
      iv(3)=imul(ic,id)
      iv(5)=imul(id,ib)
      iv(6)=imul(id,ie)
      iv(7)=-ia
      iv(9)=-ij+im
      iv(10)=ih+in
      iv(11)=imul(ip,id)
c     translational calculations.
      do 3 i=1,3
      ia=iv(i)
      ib=iv(i+4)
      ic=iv(i+8)
      id=imul(ix,ia)
      ie=imul(iy,ib)
      ip=imul(iz,ic)
3     iv(i+12)=-id-ie-ip
c     windowing calculations.
      id=iv(15)
      iv(3)=imul(ia,iview)
      iv(7)=imul(ib,iview)
      iv(11)=imul(ic,iview)
      iv(15)=imul(id,iview)
      return
      end
```

Figure 10. IMATRX subroutine.

ISYNTH SUBROUTINE. This subroutine uses screen coordinate interpolation to synthesize many images from just two real frames. Since end points of lines move in nearly straight lines and at an almost constant velocity over a short period of time, the synthesized frames are very good approximations to the real projection. Only addition of interpolation constants (delta values) is needed to synthesize an image thus avoiding clipping, transforming, and projecting. The synthesizer was found to produce no noticable geometric distortion and resulted in a higher projection rate. The frame synthesizer works out of four buffers in the following way.

PASS 1- Delta values are calculated for each projected line's screen end points.

delta x= (new x-old x)/number of frames

delta y= (new y-old y)/number of frames

If an old frame line's code indicates that the line is turned off (not to be projected), no delta values are calculated for it. Only lines which are turned on in the new and old frame will be projected. Since PASS 2 only uses the old frame code, this code is changed to indicate the proper condition if necessary (a line leaves the screen in this way). The delta values are added to the old screen end points creating an updated old frame. The updated frame is then projected.

PASS 2 THROUGH N- During the intermediate frames, between the first and last pass, the delta values are added to the updated old frame and it is projected.

PASS N- The new frame's points and corresponding codes are

transfered to the old frame buffer and new display points and codes are transferred from the projection buffer to the new frame buffer. The old frame is then projected.

If screen point hysteresis is being used, projection buffer points are checked to determine whether the hysteresis value has been exceeded. If it has, the projection buffer point gets transferred to the new frame buffer. If not, the new frame buffer retains its previous point. Figures 11 and 12 show the details of the frame synthesis.

```
      subroutine isynth (ipass)
c     isynth*************************************************
c     nframe frames are synthesized out of just 2 base frames.
c        pass 1:  delta values are calculateded.
c        pass 2 to nframe-1:  delta values are added.
c        pass nframe:  a new frame is transferred.
c     projection occurs on every pass.
c     isynth*************************************************
c
c     data assignments
      common /synth/ new(400),iold(400),idelta(400),nframe,ihyst
      common /proj/  i2d(400)
      common /crt/   dbuf(800)
      i=1
c     showit transfers a new image to the display generator.
      call showit
c
c     pass number decision.
      if (ipass) 1,2,3
3     if (ihyst) 33,33,53
c
c     pass 1 action.
11    iold(i)=-1
      go to 15
13    if (iold(i).lt.0) go to 15
      do 14 j=1,4
      idelta(i+j)=(new(i+j)-iold(i+j))/nframe
14    iold(i+j)=iold(i+j)+idelta(i+j)
      j=iold(i+1)
      k=iold(i+2)
      l=iold(i+3)
      m=iold(i+4)
c     atline (x1,y1,x2,y2) draws a line from points 1 to 2.
      call atline (j,k,l,m)
15    i=i+5
1     if (new(i)) 11,40,13
c
c     pass2 to nframe-1 action.
21    do 22 j=1,4
22    iold(i+j)=iold(i+j)+idelta(i+j)
      j=iold(i+1)
```

Figure 11. ISYNTH subroutine part a.

```
      k=iold(i+2)
      l=iold(i+3)
      m=iold(i+4)
      call atline (j,k,l,m)
23    i=i+5
2     if (iold(i)) 23,40,21
c
c     pass nframe action  (last pass).
30    i=i+5
33    do 31 j=0,4
      iold(i+j)=new(i+j)
31    new(i+j)=i2d(i+j)
      if (iold(i).le.0) go to 92
      j=iold(i+1)
      k=iold(i+2)
      l=iold(i+3)
      m=iold(i+4)
      call atline (j,k,l,m)
92    if (i2d(i)) 30,40,30
c
c     pass nframe action with hysteresis.
50    i=i+5
53    do 51 j=1,4
      iold(i+j)=new(i+j)
      ihl=i2d(i+j)-3
      ih2=i2d(i+j)+3
      if (ihl.gt.new(i+j)) new(i+j)=ihl
51    if (ih2.lt.new(i+j)) new(i+j)=ih2
      iold(i)=new(i)
      new(i)=i2d(i)
      if (i2d(i).le.0) go to 52
      j=iold(i+1)
      k=iold(i+2)
      l=iold(i+3)
      m=iold(i+4)
      call atline (j,k,l,m)
52    if (i2d(i)) 50,40,50
c
40    return
      end
```

Figure 12. ISYNTH subroutine part b.

IGEN SUBROUTINE. When a clipped two dimensional projection of an array of transformed data base points is needed, the IGEN subroutine is called. This is a time efficient version of the Cohen and Sutherland projection and clipping algorithm. The following features are incorporated to effect the speedup.

1. Point swapping during clipping is eliminated.
2. Old screen point values are used for continue point projection values if they are found to be valid.
3. Redundant on or off the screen checking is eliminated.
4. A five element code is used instead of a four element code.

Standard array processing subroutine calling and pointer conventions are used.

The ICODE and PUSHER subroutines are used exclusively by IGEN. ICODE creates codes which indicate whether a point is off the screen and to which side of the screen it lies. PUSHER does the actual clipping by pushing a line's end point to the viewing pyramid boundry it intersects.

Due to this complexity of this subroutine, flow charts are presented in addition to the subroutine listings in figures 13 through 18.

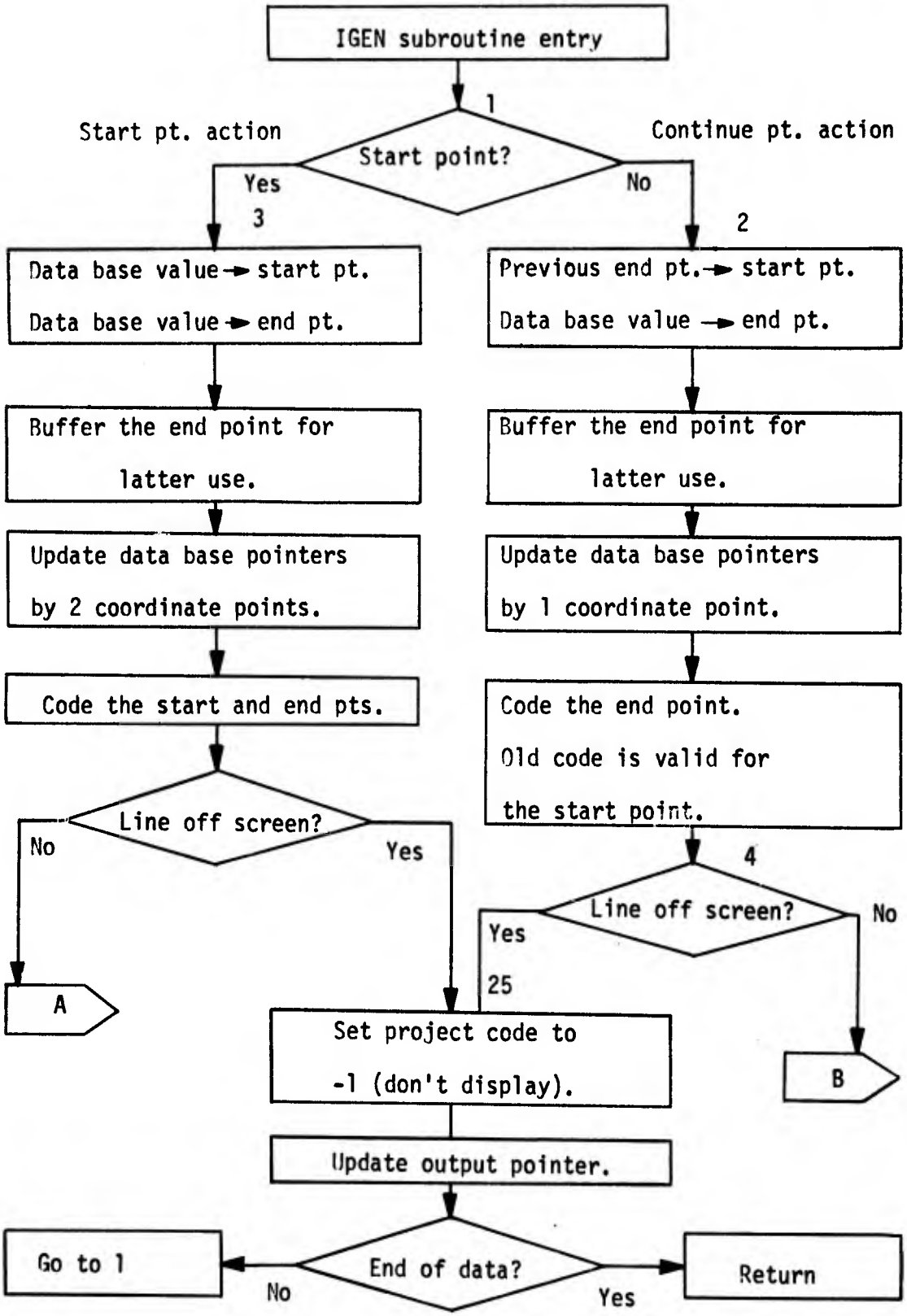
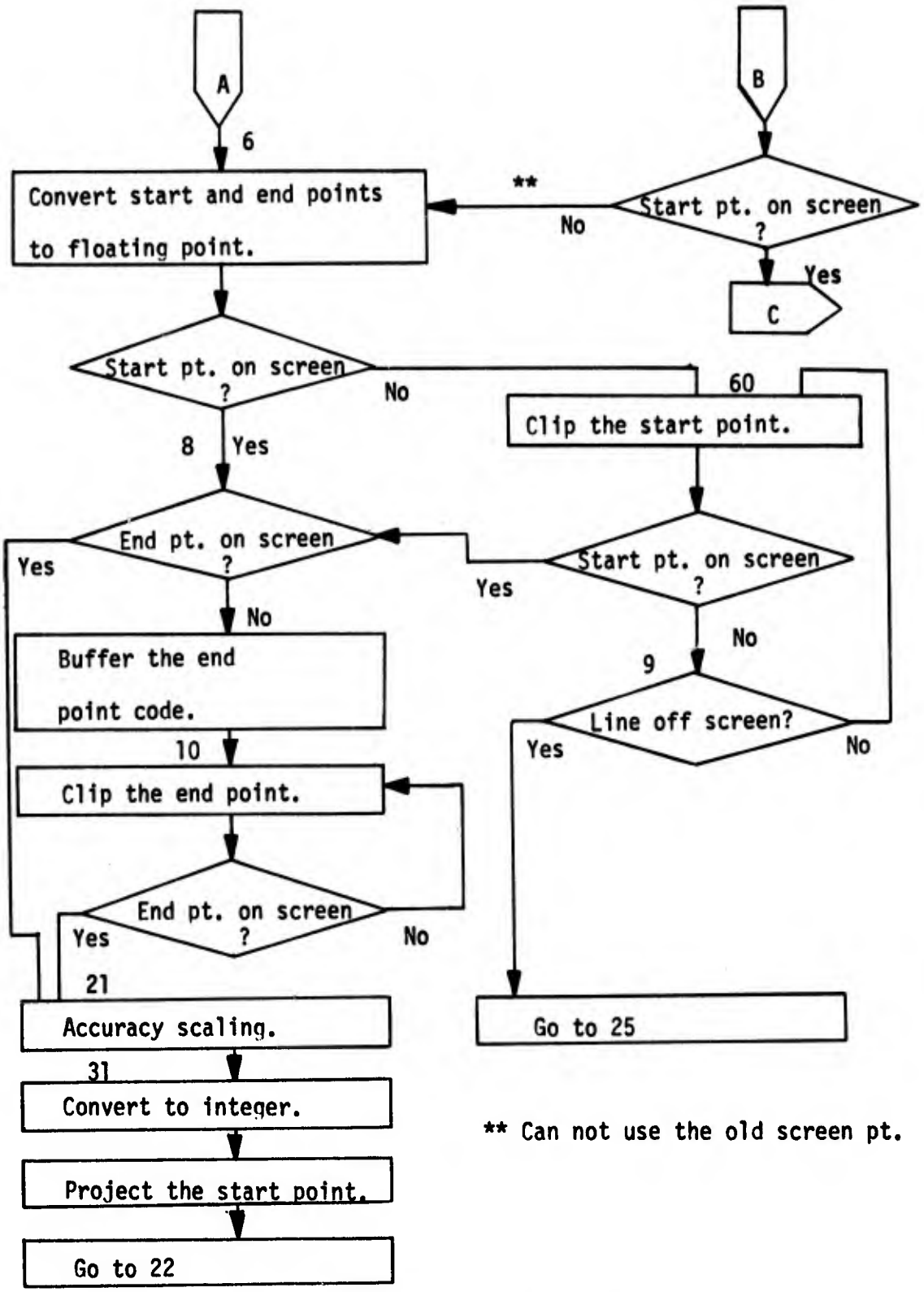


Figure 13. IGEN flow chart part a.



** Can not use the old screen pt.

Figure 14. IGEN flow chart part b.

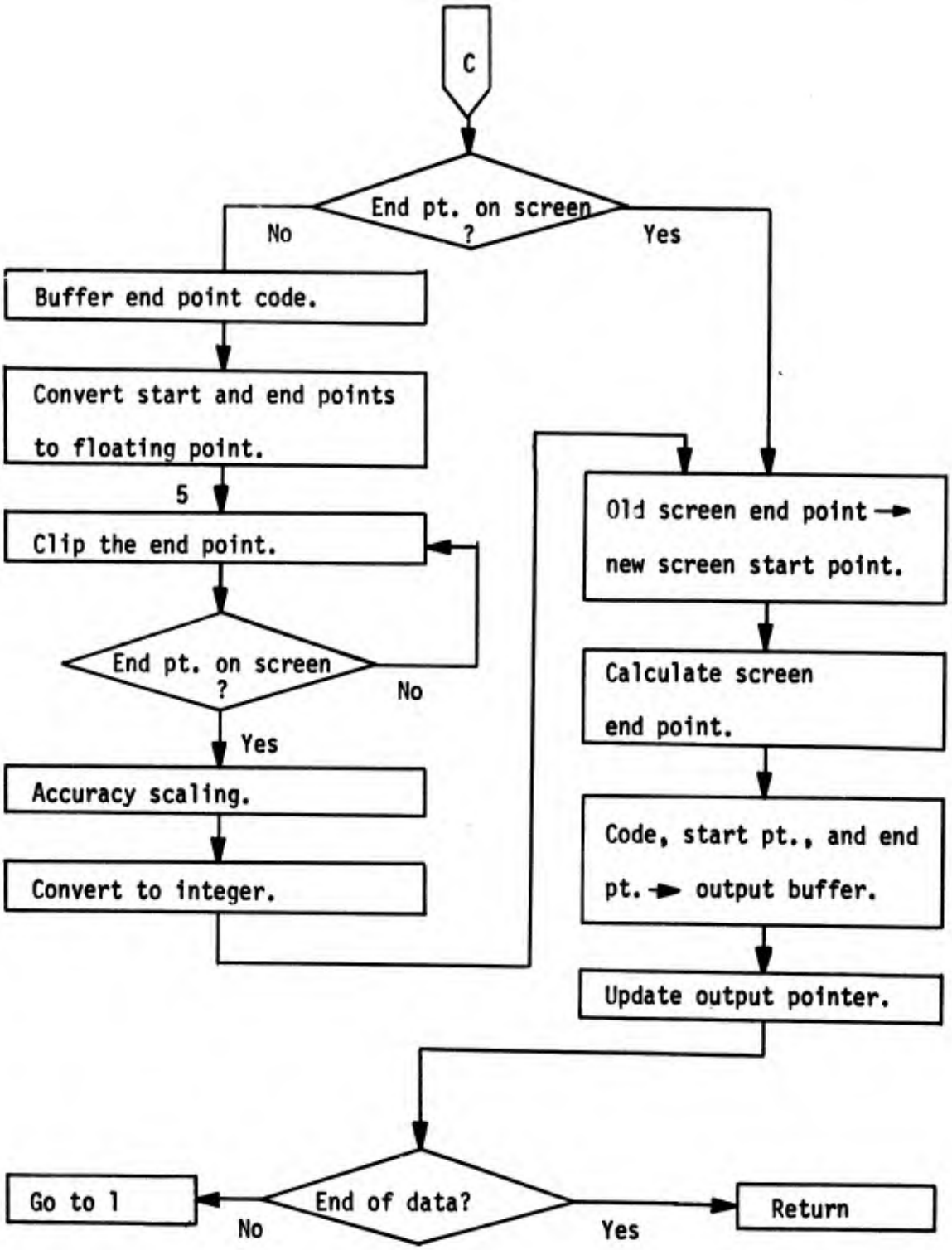


Figure 15. IGEN flow chart part c.

```
      subroutine igen (jd,index,j2d,k,m,n)
c     igen-----------------------------------------------
c     igen controls the clipping and projection of clipped lines.
c     refer to the flow chart and lables for operation.
c     igen-----------------------------------------------
c
c     data assignments.

      dimension j2d(400),jd(400),index(100)
      logical c1(5),c2(5),cb2(5)
c
1     if (index(m)) 2,2,3
c
c     continue point action.
2     jx1=jxb2
      jy1=jyb2
      jz1=jzb2
      jx2=jd(k)
      jy2=jd(k+1)
      jz2=jd(k+2)
      jxb2=jx2
      jyb2=jy2
      jzb2=jz2
      k=k+4
      m=m+1
      do 11 i=1,5
11    c1(i)=c2(i)
      call icode (c2,jx2,jy2,jz2)
      do 4 i=1,4
4     if (c1(i).and.c2(i)) go to 25
      if (.not. c1(5)) go to 6
      if (c2(5)) go to 20
      do 12 i=1,5
12    cb2(i)=c2(i)
      x1=jx1
      y1=jy1
      z1=jz1
      x2=jx2
      y2=jy2
      z2=jz2
```

Figure 16. IGEN subroutine part a.

```
5       call pusher (x2,y2,z2,x1,y1,z1,cb2)
        if (.not.cb2(5)) go to 5
        if (z2.gt.500.) go to 55
        x2=x2*50.
        y2=y2*50.
        z2=z2*50.
55      jx2=x2
        jy2=y2
        jz2=z2+1.
20      j2d(n+1)=j2d(n-2)
        j2d(n+2)=j2d(n-1)
22      call ipyra (jx2,jz2,jp)
        j2d(n+3)=jp
        call ipyra (jy2,jz2,jp)
        j2d(n+4)=jp
        j2d(n)=1
        n=n+5
        if (index(m-1).eq.0) return
        go to 1
c
c       start point and following continue point action.
3       jx1=jd(k)
        jy1=jd(k+1)
        jz1=jd(k+2)
        jx2=jd(k+4)
        jy2=jd(k+5)
        jz2=jd(k+6)
        jxb2=jx2
        jyb2=jy2
        jzb2=jz2
        k=k+8
        m=m+2
        call icode (c1,jx1,jy1,jz1)
        call icode (c2,jx2,jy2,jz2)
        do 7 i=1,4
7       if (c1(i).and.c2(i)) go to 25
6       x1=jx1
        y1=jy1
        z1=jz1
        x2=jx2
```

Figure 17. IGEN subroutine part b.

```
      y2=jy2
      z2=jz2
      if (c1(5)) go to 8
60    call pusher (x1,y1,z1,x2,y2,z2,c1)
      if (c1(5)) go to 8
      do 9 i=1,4
9     if (c1(i).and.c2(i)) go to 25
      go to 60
8     if (c2(5)) go to 21
      do 13 i=1,5
13    cb2(i)=c2(i)
10    call pusher (x2,y2,z2,x1,y1,z1,cb2)
      if (.not.cb2(5)) go to 10
21    if (z1.gt.500.) go to 30
      x1=x1*50.
      y1=y1*50.
      z1=z1*50.
30    if (z2.gt.500.) go to 31
      x2=x2*50.
      y2=y2*50.
      z2=z2*50.
31    jx1=x1
      jy1=y1
      jz1=z1+1.
      jx2=x2
      jy2=y2
      jz2=z2+1.
      call  ipyra (jx1,jz1,jp)
      j2d(n+1)=jp
      call  ipyra (jy1,jz1,jp)
      j2d(n+2)=jp
      go to 22
c
c     nonvisible line elimination.
25    j2d(n)=-1
      n=n+5
      if (index(m-1).eq.0) return
      go to 1
      end
```

Figure 18. IGEN subroutine part c.

ICODE SUBROUTINE. A five element code is generated to indicate where in space a 3D point lies in relation to a viewing pyramid. The first four elements of the code are the same as the Cohen Sutherland code. The fifth code element tells whether a point is on or off the screen. This eliminates the need to test the other four code elements to determine if a point is on the screen.

Code Element

1. The point is to the left of the x=-z plane
2. The point is to the right of the x=z plane
3. The point is below the y=-z plane
4. The point is above the y=z plane
5. The point is within the viewing pyramid

The planes used in this code describe the viewing pyramid within which visible points lie. The ICODE subroutine shown in figure 19 describes the coding algorithm.

PUSHER SUBROUTINE. The mathematics used to clip a line are performed here. First, code elements are checked to determine towards which screen boundary a line's end point must be pushed. The mathematics (see figure 20) are then performed and the line is recoded. Floating point arithmetic is used for pushing the points to the screen boundry due to the need for high precision when clipping long lines which intersect the viewing pyramid near its base.

```
      subroutine icode (code,jx,jy,jz)
c     icode-----------------------------------------------
c     a 5 element code is assigned to the point jx,jy,jz
c     based on which side of the planes the point falls.
c     code (1,2,3,4,5) is the code.
c     1= point is to the left of the jx=-jz plane
c     2= point is to the right of the jx=jz plane
c     3= point is below the jy=-jz plane
c     4= point is above the jy=jz plane
c     5= point is within the viewing pyramid
c     icode-----------------------------------------------
c
      logical code(5)
      code(5)=.true.
      if (jx.lt.-jz) go to 1
      code(1)=.false.
11    if (jx.gt.jz) go to 2
      code(2)=.false.
22    if (jy.lt.-jz) go to 3
      code(3)=.false.
33    if (jy.gt.jz) go to 4
      code(4)=.false.
      return
1     code(1)=.true.
      code(5)=.false.
      go to 11
2     code(2)=.true.
      code(5)=.false.
      go to 22
3     code(3)=.true.
      code(5)=.false.
      go to 33
4     code(4)=.true.
      code(5)=.false.
      return
      end
```

Figure 19. ICODE subroutine.

```
      subroutine pusher (x1,y1,z1,x2,y2,z2,code)
c     pusher-----------------------------------------------
c     line clipping is performed.  point x1,y1,z1 is pushed
c     toward point x2,y2,z2 until a pyramid intersection occurs.
c     the pushed point is then recoded.
c     pusher-----------------------------------------------
c
      logical code(5)
      if (code(3)) go to 1
      if (code(4)) go to 2
      if (code(1)) go to 3
c
c     push left.
      t=(z1-x1)/((x2-x1)-(z2-z1))
       z1=t*(z2-z1)+z1
       x1=z1
       y1=t*(y2-y1)+y1
      go to 4
c
c     push up.
1     t=(z1+y1)/((y1-y2)-(z2-z1))
       z1=t*(z2-z1)+z1
       x1=t*(x2-x1)+x1
       y1=-z1
      go to 4
c
c     push down.
2     t=(z1-y1)/((y2-y1)-(z2-z1))
       z1=t*(z2-z1)+z1
       x1=t*(x2-x1)+x1
       y1=z1
      go to 4
c
c     push right.
3     t=(z1+x1)/((x1-x2)-(z2-z1))
       z1=t*(z2-z1)+z1
       x1=-z1
       y1=t*(y2-y1)+y1
c
c     recode.
4     i=x1
      j=y1
      k=z1
      call icode (code,i,j,k)
      return
      end
```

Figure 20. PUSHER subroutine.

IPROJ SUBROUTINE. In many cases it is desirable to project many small objects without the clipping restriction, that is, to not project the line if any part of it falls off the screen. Projecting lines in this manner not only eliminates the need for clipping but for coding of points as well. IPROJ treats a set of points as a list of start and end point pairs and converts them into screen coordinates. If a line is off the screen, it sets the project/no project code to no project and moves on to the next line. This is a very small subroutine, intended for assembly language replacement.

Divide instructions on most computers have an overflow on divide warning feature which can very efficiently be used to eliminate most of the off screen point checking. The projection functions are:

$$\text{screen } x = 28000*(x/z)$$

$$\text{screen } y = 28000*(y/z)$$

If x/z or y/z are greater than one, the point is off the screen. Due to the same two conditions, accumulator or register overflow will also result and a warning flag will be set. Thus, this flag can be used as a project/no project indicator. An added bonus is the fact that divide instructions usually run about three times faster when overflow abort occurs, wasting less time on unprojected points. Note that any point with a negative z must still be eliminated by the software as it falls behind the viewing pyramid. The Fortran version of this subroutine is shown in figure 21.

```
      subroutine iproj (in,index,iout,k,m,n)
c     iproj-----------------------------------------------
c     nonclipped lines are projected.  if any part of a
c     line falls outside the viewing pyramid , it is eliminated.
c     iproj-----------------------------------------------
c
c     data assignments
      dimension in(400),iout(400),index(100)
1     if (in(k+2).le.0.or.in(k+6).le.0) go to 2
c
      x=in(k)
      y=in(k+1)
      z=in(k+2)
      out=x/z*25000.
      if (out.gt.25000..or.out.lt.-25000.) go to 2
      iout(n+1)=out
      out=y/z*25000.
      if (out.gt.25000..or.out.lt.-25000.) go to 2
      iout(n+2)=out
c
      x=in(k+4)
      y=in(k+5)
      z=in(k+6)
      out=x/z*25000.
      if (out.gt.25000..or.out.lt.-25000.) go to 2
      iout(n+3)=out
      out=y/z*25000.
      if (out.gt.25000..or.out.lt.-25000.) go to 2
      iout(n+4)=out
      iout(n)=1
      go to 3
c
2     iout(n)=-1
c
3     k=k+8
      m=m+2
      n=n+5
      if (index(m-1).ne.0) go to 1
      return
      end
```

Figure 21. IPROJ subroutine.

NAVIGATIONAL SUBROUTINES

The navigational subroutines, IGAT and INAV, provide the display driver with a data base and reference and scale information. The display subroutines then produce an accurate projection. The form which navigational subroutines take is largely dependent on program application and what equipment is being used. Two types of reference information are used.

NAV - Navigational simulator reference parameters.

Simulator position information is passed between program units in floating point mode allowing a large range of simulator movement. Since the display driver operates in integer mode, map overlays must be used to extend display range if overflow is to be avoided. The XFSET and ZFSET are the ground plane offsets of the map being used. The NAV positions (XPOS and ZPOS) minus the offsets put the reference parameters within the integer range of plus or minus 32767.

REF - Graphical reference parameters.

Display driver integer mode references of x, y, z, pitch, bank, heading intended for direct use by the display driver are included in the REF common block. The field of vision parameter is also passed to other program units through REF.

IGAT SUBROUTINE. Simulator coordinates are read, filtered and scaled to provide REF information. Navigational data is passed to the other program units through NAV. In the case of the GAT 2 simulation for which this version of IGAT is written (figures 22 and 23), positional information is obtained by calling subroutines which examine analog to digital converter outputs which represent the simulators position.

INAV SUBROUTINE. Map overlays and their appropriate offsets and scaling are handled here. Map decisions are based on simulator position. The actual data base swapping is performed in this subroutine. This subroutine has not been used extensively and is not used in the example control program. A simple version of INAV is shown in figure 24.

```
      subroutine igat
c     igat*************************************************
c     singer link gat2 simulator interface program
c     simulator coordinates are read and filtered.
c     this is a good example of a simulator interface
c     program but it was found that better filtering is
c     needed for practical applications.
c     igat*************************************************
c
c     data assignments
      integer da2,dr2,de2,pi,ya,ba,qapi,paro,raya,rc,vp,sinh,cosh,al
      integer th1,th2,xramp,yramp,da,dr,de
      common /nav/xpos,ypos,zpos,xfset,zfset,iscale
      common /ref/ix,iy,iz,ipit,ibnk,ihdg
      data ixn,ixo,izn,izo,iyo,istart/5*0,1/
c
c     ramp calculation decision.
      if (istart) 2,2,1
1     write (13,1000)
1000  format (' type 1 for ramp calculations,0 for no ramp')
      read (13,1002) irmp
1002  format (i1)
      istart=0
c
c     get gat parameters
c     dalyad2 and gatxy subroutines provide positional data
2     call dalyad2(da,dr,de,da2,dr2,de2,ba,ya,pi,paro,
     c raya,qapi,rc,vp,sinh,cosh,al,th1,th2,xramp,yramp)
      call gatxy(igatx,igaty)
c
c     calculate heading angle
      sine=sinh
      if (cosh.eq.0) cosh=1
      cosine=cosh
      angle=sine/cosine
      ihdg=atan(angle)*57.29578
      if (cosh.lt.0) ihdg=ihdg+180
      if (ihdg.lt.1) ihdg=ihdg+360
c
c     create x and y coordinates with offset values.
c     scale the values.
      z=igaty
      x=igatx
      iz=z*16.-(zfset/4.)
      ix=x*16.-(xfset/4.)
```

Figure 22. IGAT subroutine part a.

```
c
c     adding the ramp value.
      if (irmp)10,10,6
6     iz=iz+(yramp/128)
      ix=ix+(xramp/128)
c
c     boundary correction: part1, position estimation
      ixest=ixn+(ixn-ixo)
      izest=izn+(izn-izo)
c     boundary correction: part2, correction
      idiffx=ix-ixest
      idiffz=iz-izest
      if (idiffx.lt.-20.or.idiffx.gt.20) go to 9
      if (idiffz.lt.-20.or.idiffz.gt.20) go to 9
      if (idiffx.gt.8) ix=ix-16
      if (idiffx.lt.-8) ix=ix+16
      if (idiffz.gt.8) iz=iz-16
      if (idiffz.lt.-8) iz=iz+16
      go to 10
c
c
c     positional filtering.  three trial averaging.
9     continue
10     i=(ix+ixn+ixo)/3
      j=(iz+izn+izo)/3
      ixo=ixn
      ixn=ix
      ix=i
      izo=izn
      izn=iz
      iz=j
c
c     altitude filtering. two trial averaging.
      iy=al-120
      i=(iy+iyo)/2
      iyo=iy
      iy=i
c
c     pitch and bank calculations.
      ipit=pi/12
      if (ipit.lt.1) ipit=ipit+360
      ibnk=ba/12
      if(ibnk.lt.1) ibnk=ibnk+360
      return
      end
```

Figure 23. IGAT subroutine part b.

```
      subroutine inav
c     inav-------------------------------------------------
c     this navigational subroutine does two things:
c     1. switches to the low altitude data base if altitude
c     drops below 500 feet and turns on screen point hysteresis.
c     2.  switches to the high altitude data base if altitude
c     goes above 750 feet.
c     inav-------------------------------------------------
c
      common /base/  idba(400),indexa(100),idaux(80),indaux(20)
      common /base2/ idbb(400),indexb(100)
      common /ref/   ix,iy,iz,ipit,ibnk,ihdg
      common /synth/ new(400),iold(400),idelta(400),nframe,ihyst
c
      if (iflag.eq.1.and.iy.lt.500) go to 1
      if (iflag.eq.0.and.iy.gt.750) go to 2
      return
c
c     switch to low data base and turn hysteresis on.
1     do 10 i=1,400
      itemp=idba(i)
      idba(i)=idbb(i)
10    idbb(i)=itemp
      do 11 i=1,100
      itemp=indexa(i)
      indexa(i)=indexb(i)
11    indexb(i)=itemp
      ihyst=1
      iflag=0
      return
c
c     switch to high data base and turn hysteresis off.
2     do 20 i=1,400
      itemp=idba(i)
      idba(i)=idbb(i)
20    idbb(i)=itemp
      do 21 i=1,100
      itemp=indexa(i)
      indexa(i)=indexb(i)
21    indexb(i)=itemp
      ihyst=0
      iflag=1
      return
      end
```

Figure 24. INAV subroutine.

COMPUTATIONAL SUBROUTINES

IPYRA SUBROUTINE. An equation often used in point projection is the pyramid projection equation.

$$\text{screen } x = 3D\ x / 3D\ z * (k/2)$$

The variable k is the screen width. Division by z gives the projection depth perspecive. IPYRA is a small assembly language plug-in replacable module which performs this function. The following Fortran version of the IPYRA module (figure 25) can be used on any computer and converts data to floating point mode to perform the division and retain accuracy. Great execution time speedups result from assembly language replacement of this module.

The thirty-two bit product capabilities of a sixteen bit computer's multiplication hardware are utilized to retain accuracy in the division and multiplication in the assembly language version shown.

IMUL SUBROUTINE. As stated previously, multiplication of a 16 bit fractional constant by an integer is a common occurance. Accurate fractional multiplication is performed by IMUL.

The fraction and integer are multiplied and the top sixteen bits of the thirty-two bit product are taken as the result. Binary point placement for the fraction is accomplished in this way. The Fortran module shown in figure 26 converts data to floating point mode to retain high accuracy while the assembly language version uses thirty-two bit hardware product capabilities as IPYRA does. Overall display program speedups of up to four hundred percent have been obtained by assembly language replacement of IMUL.

```
      subroutine ipyra (i,j,jp)
c     ipyra-----------------------------------------------
c     the pyramid projection function is accurately performed.
c     jp=(i/j)*25000.
c     this subroutine is assembly language replacable.
c     ipyra-----------------------------------------------
      ri=i
      rj=j
      jp=(ri/rj*25000.)
      return
      end
```

Figure 25. IPYRA subroutine.

```
      function imul(i,j)
c     imul fortran module----------------------------------
c     imul is an assembly language replacable module.
c     imul performs   i*j/32767
c     imul fortran module----------------------------------
      ri=i
      rj=j
      imul=ri*rj/32767.0
      return
      end
```

Figure 26. IMUL subroutine.

INITIALIZATION

INIT SUBROUTINE. Lookup table generation and data base read-in are performed by the initialization subroutine. Variable program parameters are assigned default values which are redefined by parameter control statements in the users data base if default is not specified (see data base format section). Figures 27 and 28 show the INIT subroutine.

BLOCK DATA SUBROUTINE. All common blocks are allocated by the block data subroutine. Initialization of synthesizer buffers and other display buffers and variables are also performed. The block data subroutine shown in figure 29 is used with the sample control program.

```
      subroutine init
c     init*************************************************
c     init reads in and generates parameters and data base blocks.
c     init*************************************************
c
c     data assignments.
      common /base/  idba(400),indexa(100),idaux(80),indaux(20)
      common /base3/ i3d(400),i3daux(80)
      common /crt/   dbuf(500)
      common /mtx/   iv(16),iview
      common /nav/   xpos,ypos,zpos,xfset,zfset,iscale
      common /trig/  isin(360),icos(360)
c
c     default value assignments.
      iview=32767
c
c     initialize the display buffer and screen
      call getup
      call opnfil (dbuf,740)
c
c     the initial display start point is now chosen.
c     find the immediate position of the simulator.
      call gatxy(igatx,igaty)
      xfset=igatx
      xfset=xfset*64.
      zfset=igaty
      zfset=zfset*64.+3000.
      iscale=1
c
c     read the default/setup card.
      read (21,1001) isetup
1001  format (i1)
      if (isetup.eq.0) go to 12
c
c     read the field of view parameter.
      read (21,1002) field
1002  format (f10.0)
      field=field/360.*6.28308
      iview=sin(field)/cos(field)*32767.
c
c     read in the 4 data base blocks.
12    i=1
      nbuf=0
1     read (21,1000) indexa(i),(idba(i*4-4+j),j=1,3)
1000  format (i2,3i6)
      idba(i*4)=1
      i=i+1
      if (indexa(i-1).ne.0) go to 1
      nbuf=nbuf+1
```

Figure 27. INIT subroutine part a.

```
      if (nbuf.ne.3) go to 1
      indexa(1)=i-1
      i=1
2     read (21,1000) indaux(i),(idaux(i*4-4+j),j=1,3)
      idaux(i*4)=1
      i=i+1
      if (indaux(i-1).ne.0) go to 2
      indaux(1)=i-1
c
c     generate the sine and cosine tables.
      do 11 i=1,360
      a=i
      a=a*6.28318/360.0
      isin(i)=32767.*sin(a)
11    icos(i)=32767.*cos(a)
      return
      end
```

Figure 28. INIT subroutine part b.

```
c
c     block data*******************************************
c     the display's common blocks are set up and initialized.
c     block data*******************************************
c
      common /base/  idba(400),indexa(100),idaux(80),indaux(20)
      common /base3/ i3d(400),i3daux(80)
      common /crt/   dbuf(800)
      common /mtx/   iv(16),iview
      common /nav/   xpos,ypos,zpos,xfset,zfset,iscale
      common /proj/  i2d(400)
      common /ref/   ix,iy,iz,ipit,ibnk,ihdg
      common /synth/ new(400),iold(400),idelta(400),nframe,ihyst
      common /trig/  isin(360),icos(360)
c
      data i2d/400*0/,new/400*0/,nframe/5/
      data iv/15*0,32767/
      data ihyst/0/
      end
```

Figure 29. Block data subroutine.

DATA BASE FORMAT

A simple data base read-in format was incorporated into the display program to allow for simple data base manipulation. A data base may be on cards, tape or disc as dictated by the data read-in section of the initialization subroutine.

The first card (or card image) indicates whether or not set-up parameters will be included in the card deck. A zero in column one is used to specify that default values should be used. A one in column one indicates that set-up parameters will be given. If the first card has a one in column one, the following cards will contain set-up information such as field of view and initial position.

After the set up has been completed (by default or definition), the actual data base can be read in. The data base consists of three dimensional start and continue points. Integer mode is used and it is recommended that values be kept between plus and minus 20000. Each card contains a code and a coordinate in the following format.

I2	I6	I6	I6
Code	x	y	z

Code	Meaning
01 =	start point
-1 =	continue point
00 =	continue point and end of data base block

The display program described expects three data base blocks to be read in. The first two will be processed by the IGEN subroutine and the third will be processed by IPROJ. The predictor symbol buffer is read as the fourth block. This block is matrix multiplied (mode 2) and is projected by IGEN. A sample data base is shown in Appendix A. No

clipping is performed by IPROJ, therefore small objects should be in block three. Only lines that must be clipped are included in block one and two. Higher projection rates result from making block one and two as small as possible.

DISPLAY PERFORMANCE

The speed at which images are projected is determined by program configuration, equipment used, and data base complexity. Higher display speeds are obtained when assembly language replacement modules are used. Table 1 gives the performance of the display program in various configurations.

Although no thorough error analysis was made of the image quality, the following can be said. The picture geometry is good due to the strict mathematics used in projection. Objects keep their proper shape and perspective and don't come apart as they often do when approximation methods are used for projection. Motion quality can be described as fair. Due to the sixteen bit integer calculations, motion is not as smooth as may be desired but screen filtering has helped this situation considerably. Frame synthesizing results in a very natural smoothing of movement.

A computer with more precision would greatly increase projection speed, eliminate the need for any floating point calculations, and increase image accuracy and motion quality.

TABLE 1

PROGRAM PERFORMANCE

PROGRAM	COMPUTER	OPERATING SYSTEM	COMPILER	DISPLAY RATE (50 LINES)
FLY V03	PDP 11/40	RT 11	FORTRAN IV	2 FRAMES/SEC.
A.FOR	PDP 11/40	RT 11	FORTRAN IV	6 FPS
B.FOR	PDP 11/40	RT 11	FORTRAN IV	9 FPS
DISP.F	PDP 11/40	UNIX	FORTRAN IV	1.5 FPS
A.RAY	RAYTHEON 704	XRAY	INLINE *	4 FPS
A.RAY	RAYTHEON 704	XRAY	INLINE	7 FPS
B.RAY	RAYTHEON 704	XRAY	INLINE	11 FPS
C.RAY	RAYTHEON 704	XRAY	INLINE	20+ FPS **

* RAYTHEON 704 FORTRAN IV INLINE COMPILER
**ESTIMATED DISPLAY RATE.

	FRAME SYNTHESIS	IPYRA AND IMUL ASSEMBLY	IMX AND IPROJ ASSEMBLY	MATRIX MULTIPLIER	DISPLAY RATE
FLY V03					2 FPS
A.FOR	X				6 FPS
B.FOR	X	X			9 FPS
DISP.F	X				1.5 FPS
A.RAY	X				4 FPS
A.RAY	X	X			7 FPS
B.RAY	X	X	X		11 FPS
C.RAY	X	X	X	X	20+ FPS **

CONCLUSION

That which was set out to be accomplished in this project was, for the most part, successfully accomplished. The resulting program is transportable, easy to work with, and relatively time efficient. It was unfortunate that assembly language replacement of subroutines had to be used to obtain high projection rates, but the fact remains that a complete Fortran version of the program exists and can be run on any system which handles Fortran if speed is not of prime importance. In the writing of many of the assembly language modules it was found that translating the Fortran program into assembly language was very easy due to the program structure already being available.

The Fortran display program is already finding application in a predictor display simulation and in the future, more ground based simulations will be incorporating this program. One of the most interesting applications may be the installation of a small computer and display system inside an actual aircraft, using radio navigation signals to obtain references for the display. Very efficient, integerized software is essential for a small computer acting in this capacity therefore the basic structure of this program can be used here also.

Unlike old display programs (typically called landing displays), this program, with its map overlay feature, can be used as an overall flying display. Cross-country trips as well as simple landings can be performed. Another interesting application of this program is space flight. With the essentially limitless data base sizes provided by map overlays, a complete takeoff, docking, and reentry can be performed.

The map overlay feature is one of the most powerful parts of the display program and has not, up to this time, been developed or used extensively.

Basically, this display program has added easy to work with visual capabilities to many simulations where none were available before, without the use of expensive graphics hardware.

REFERENCES

1. Bell, J. W., Bottlik, I. P., Lucero, A. B. Simulation techniques for airborne electro-optical imaging systems. Wright-Patterson Air Force Base, Ohio: Air Force Human Resources Laboratory, Technical Report, January 1975.
2. Hoolko, R. L. Design of a digital computer-driven cathode-ray-tube display system. Savoy, Ill.: University of Illinois at Urbana-Champaign, Institute of Aviation, Aviation Research Laboratory, Technical Report ARL-73-11/AFOSR-73-7, 1973.
3. Hummel, T. L. A digital computer-generated contact analog landing display. Savoy, Ill.: University of Illinois at Urbana-Champaign, Institute of Aviation, Aviation Research Laboratory, Technical Report ARL-73-9/AFOSR-73-5, 1973.
4. Kelley, K. C. A computer graphics program for the generation of half-tone images with shadows. Urbana, Ill.: University of Illinois Thesis, 1970.
5. Newman, W. M., Sproull, R. F. Principles of interactive computer graphics. New York: McGraw-Hill, 1973.
6. Schaumacher, R., Brand, B., Gilliland, M., Sharp, W. Applying computer-generated images to visual simulation. Detroit: General Electric Company Report, 1970.

APPENDIX A

SAMPLE DATA BASE AND PROJECTIONS

SAMPLE DATA BASE

This data base consists of the runway, center lines, and approach bar shown below.

A-1. Data base form.

The following lines show the data input format for this data base.

```
1 setup data base
45.0 field of view parameter
01    25     0  1800
-1-00025     0  1800      data base block 1
-1-00025     0     0      runway
-1    25     0     0
00    25     0  1800
01     0     0   200
-1     0     0   400
01     0     0   600      data base block 2
-1     0     0   800      center lines
01     0     0  1000
-1     0     0  1200
01     0     0  1400
00     0     0  1600
01     0     0-00200
-1     0    25-00200      data base block 3
01-00015    25-00200      approach bar
00    15    25-00200
```

The initialization shown in subroutine INIT reads four data base blocks, the fourth being the predictor symbols.

Care is taken to assure that the program's buffers are large enough to accomodate the whole data base.

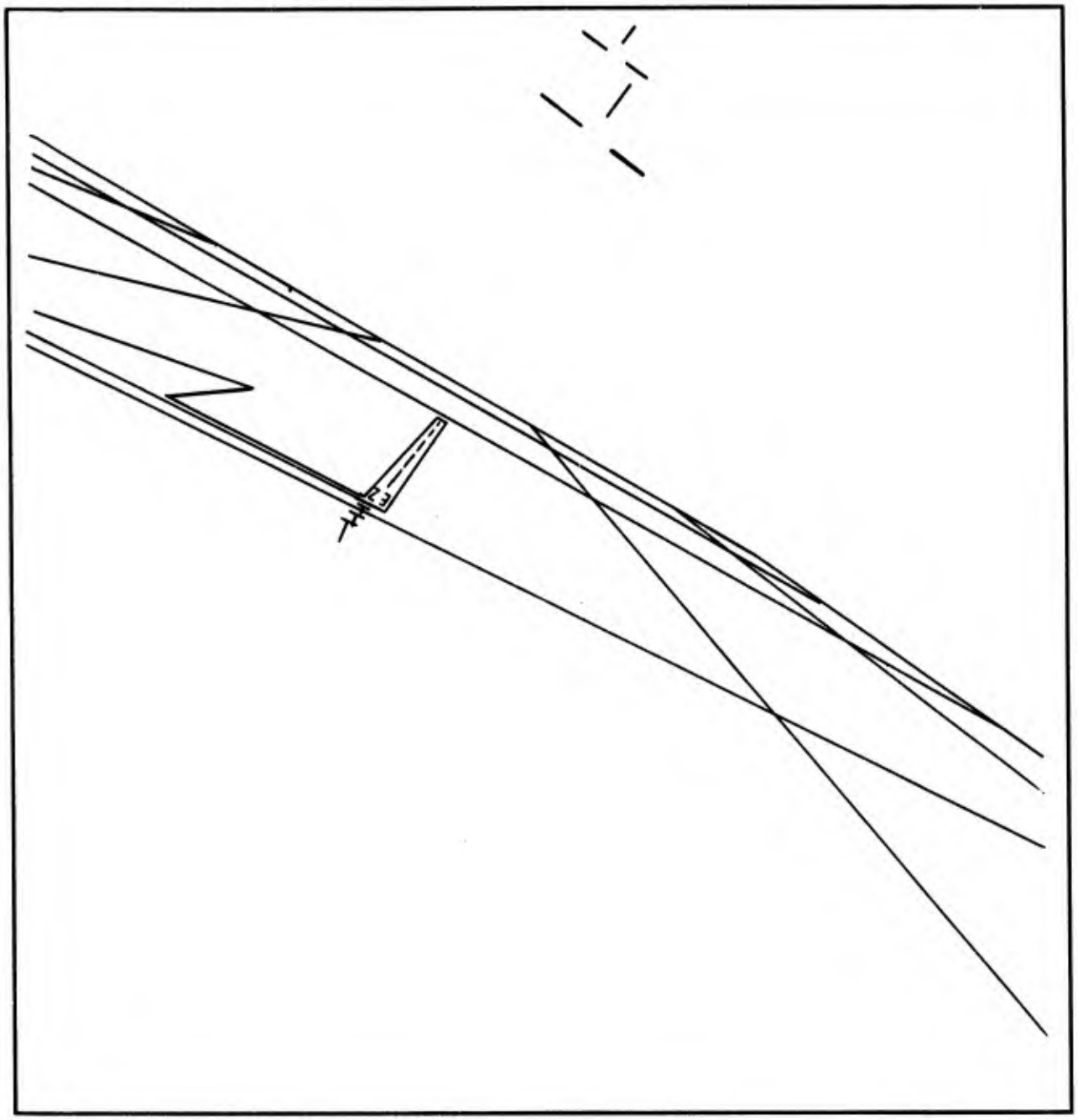


Image complexity: 50 lines Display rate: 11 frames per second

Display contents: Runway, center lines, numbers (both ends), taxiway, ramp area, 3 predictor symbols, 3 approach bars, 10 landing lights, ground texture.

A-2. Sample projection.

APPENDIX B

ASSEMBLY LANGUAGE MODULES

```
*  ipyra-----------------------------------------------------
*  ipyra subroutine module for raytheon 704
*  the call is:  call ipyra (i,j,jp)
*  the result is:  jp=i*28000/j
*  ipyra-----------------------------------------------------
         ntry  ipyra
ipyra    d     pool
         ldx   jadr
         ldw*  0
         stw   j
         ldx   iadr
         ldw*  0
         mpy   cl
         div   j
         ldx   jpadr
         stw*  0
         smb   r.ret
         jsx   r.ret
         d     pool
j        d     0
cl       d     20000
pool     d     5
         d     0,0
iadr     d     0
jadr     d     0
jpadr    d     0
         end
```

B-1. Raytheon 704 IPYRA assembly language module.

This assembly language module increases the projection rate and utilizes the Raytheon 704 multiply hardware to the fullest extent. The intermediate thirty-two bit product is stored in the accumulator (least significant bits) and in the index register.

```
*  imul-------------------------------------------------
*  raytheon 704 imul assembly function module
*  the call is:    iout=imul(i,j)
*  the result is:  iout=i*j/32768
*  imul-------------------------------------------------
         ntry  imul
calad    d     0
i        d     0
imul     equ   $
         d     0
         stw   calad
         cax
         ixs   3
         nop
         ldx*  0
         sxp
         jmp   $-2
         ldw*  0
         stw   i
         ldx   calad
         ixs   4
         nop
         ldx*  0
         sxp
         jmp   $-2
         ldw*  0
         mpy   i
         cxa
         ldx   calad
         ixs   2
         nop
         ldx*  0
         sxp
         jmp   $-2
         stw*  0
         ldx   calad
         ixs   5
         nop
         smb   r.exec
         jmp   r.exec
         end
```

B-2. Raytheon 704 IMUL assembly language module.

The IMUL function module can be replaced with this assembly language version decreasing matrix multiplication time.

```
        ;  ipyra assembly language module
        ;ipyra-----------------------------------------------
        ;  this pyramid projection subroutine performs the
        ;  function: ipyra(i,j)= (i*254/j)+256.
        ;  this function is written for the pdp 11/40 computer.
        ;ipyra-----------------------------------------------
        .globl ipyra
        .globl imul
        .radix 10
        r0=%0
        r1=%1
        r5=%5
        pc=%7
ipyra:  tst (r5)+
        mov °(r5)+,r0            note: @ is represented by the ° symbol.
        mul #254,r0
        div °(r5),r0
        add #256,r0
        rts pc
        ;
        ;
        ;  imul-------------------------------------------------
        ;  this is the fractional multiply subroutine assembly
        ;  language module, written for the pdp 11/40 computer.
        ;  imul(i,j) is the call
        ;  i*j/32767 is the result
        ;  imul-------------------------------------------------
imul:   tst (r5)+
        mov °(r5)+,r0
        mul °(r5),r0
        div #32767,r0
        rts pc
        .end
```

B-3. PDP 11 assembly language modules.

These two assembly language modules can increase display rate on a PDP 11 computer running under an RT-11 fortran compiler. The IPYRA version shown above not only multiplies by a scale factor (254) but adds a constant to the result as well. This scales and shifts the image into plasma panel coordinates, as this was the output device for the application where this subroutine was used.

DISTRIBUTION LIST

Director, Engineering Psychology (5 cys)
Programs, Code 455
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217

Defense Documentation Center (6 cys)
Cameron Station
Alexandria, Virginia 22314

Lt. Col. Henry L. Taylor, USAF
OAD (E&LS) ODDR & E
Pentagon, Rm 3D129
Washington, DC 20301

Dr. Robert Young
Director Human Resources Research
Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Human Factors Plans, OP987P7
Office of the Chief of Naval
Operations
Department of the Navy
Washington, DC 20350

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, DC 20380

Assistant Chief for Technology, Code 200
Office of Naval Research
800 N. Quincy Street
Arlington, Virginia 22217

Office of Naval Research (6 cys)
International Programs
Code 1021P
800 North Quincy Street
Arlington, Virginia 22217

Aircraft Instrumentation Program,
Code 211
Office of Naval Research
800 N. Quincy Street
Arlington, Virginia 22217

Director, ONR Branch Office
ATTN: Dr. C. E. Davis
536 South Clark Street
Chicago, IL 60605

Director, ONR Branch Office
ATTN: Mr. R. Lawson
1030 East Green Street
Pasadena, CA 91106

Dir., Naval Research Laboratory (6 cys)
Technical Information Division
Code 2627
Washington, DC 20375

Mr. John Hill
Naval Research Laboratory
Code 5707.40
Washington, DC 20375

Dr. Andreas B. Rechnitzer
Office of the Oceanographer
of the Navy
Hoffman Building II
200 Stoval Street
Alexandria, VA 22332

Dr. Heber G. Moore
Hqs., Naval Material Command
Code 0331
Department of the Navy
Washington, DC 20360

Mr. Arnold Rubinstein
Naval Material Command, NAVMAT 0344
Department of the Navy
Washington, DC 20360

Commander, Naval Air Systems Command
Crew Station Design, AIR 5313
Department of the Navy
Washington, DC 20360

Commander, Naval Air System Command
Human Factors Programs
AIR 340F
Washington, DC 20360

Commander, Naval Air Systems Command
ATTN: Mr. T. Momiyama
Advance Concepts Division, AIR03P34
Washington, DC 20360

Dr. Herbert J. Mueller
Naval Air Systems Command
AIR-310, Research Admin.
Washington, DC 20360

Mr. George Tsaparas
Naval Air Systems Command
NAVAIR 340D
Washington, DC 20360

CDR Thomas Gallagher
Bureau of Medicine & Surgery
Operational Psychology Branch
Code 513
Washington, DC 20372

Dr. George Moeller
Head, Human Factors Engineering Branch
Submarine Medical Research Laboratory
Naval Submarine Base
Groton, CT 06340

CDR James Goodson
Chief, Aerospace Psychology Division
Naval Aerospace Medical Institute
Pensacola, FL 32512

Bureau of Naval Personnel
Special Assistant for Research Liaison
PERS-OR
Washington, DC 20370

Dr. Fred Muckler
Manned Systems Design, Code 311
Navy Personnel Research and
Development Center
San Diego, CA 92152

CDR Robert Wherry
Human Factors Engineering Branch
Crew Systems Department
Naval Air Development Center
Johnsville
Warminster, PA 18974

CDR Robert Kennedy
Human Factors Engineering Branch
Code 5342
Pacific Missile Test Center
Point Mugu, CA 93042

Mr. Ronald A. Erickson
Head, Human Factors Branch,
Code 4075
Naval Weapons Center
China Lake, CA 93555

Systems Engineering Test
Directorate
ATTN: Mr. F. Hoerner
U. S. Naval Air Test Center
Patuxent River, MD 20670

Mr. Richard Coburn
Head, Human Factors Division
Naval Electornics Laboratory Center
San Diego, CA 92152

Mr. James L. Long
Weapons Systems Research (N-332)
Naval Education & Training Command
Naval Air Station
Pensacola, FL 32508

Human Factors Dept., Code N215
Naval Training Equipment Center
Orlando, FL 32813

Dr. Alfred F. Smode
Training Analysis & Evaluation Group
Naval Training Equipment Center
Code N-00T
Orlando, FL 32813

Dr. Gary Poock
Operations Research Department
Naval Postgraduate School
Monterey, CA 93940

Technical Director
U.S. Army Human Engineering Labs
Aberdeen Proving Ground
Aberdeen, MD 21005

U.S. Air Force Office of
Scientific Research
Life Sciences Directorate, NL
1400 Wilson Boulevard
Arlington, VA 22209

Chief, Human Engineering Division
Aerospace Medical Research Lab
Wright-Patterson AFB, OH 45433

Lt. Col. Joseph A. Birt
Human Engineering Division
Aerospace Medical Research Laboratory
Wright Patterson, AFB, OH 45433

Dr. Stanley Deutsch
Office of Life Sciences
Headquarters, NASA
600 Independence Avenue
Washington, DC 20546

Dr. Clyde Brictson
Dunlap and Associates, Inc.
115 South Oak Street
Inglewood, CA 90301

Mr. Edward Connelly
OMNEMII, Inc.
Tyson's International Bldg.
8150 Leesburg Pike, Suite 600
Vienna, VA 22180

Dr. E. Jones, Life Sciences Dept.
McDonnell Douglas Astronautics Co.-East
St. Louis, MO 63166

Mr. Gail J. Borden
Human Factors Research, Inc.
Santa Barbara Research Park
6780 Cortona Drive
Goleta, CA 93017

Dr. Gordon H. Robinson
University of Wisconsin - Madison
Department of Industrial Engineering
1513 University Avenue
Madison, WI 53706

Dr. W. S. Vaughan
Oceanautics, Inc.
3308 Dodge Park Road
Landover, MD 20785

Director, Human Factors Wing
Defense & Civil Institute of
Environmental Medicine
Post Office Box 2000
Downsville, Toronto, Ontario
CANADA

Dr. Malcolm L. Ritchie
Professor, Human Factors Engineering
Wright State Universtiy
Dayton, OH 45431

Dr. Lloyd Hitchcock
Human Factors Engineering Branch
Crew Systems Department
Naval Air Development Center
Warminster, PA 18974