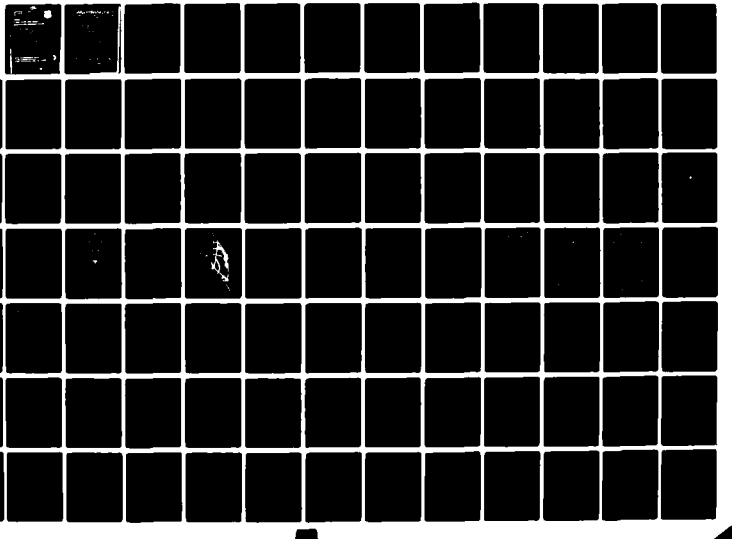
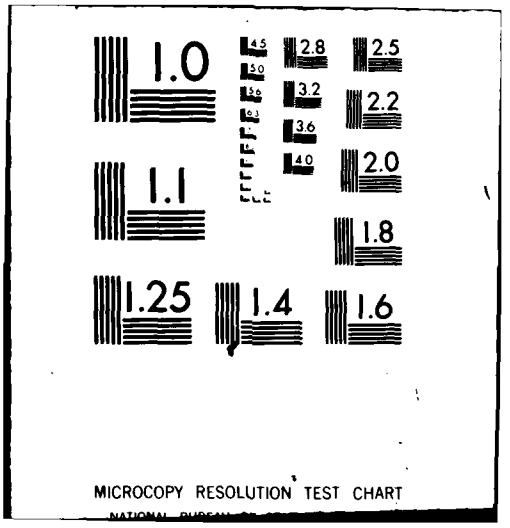
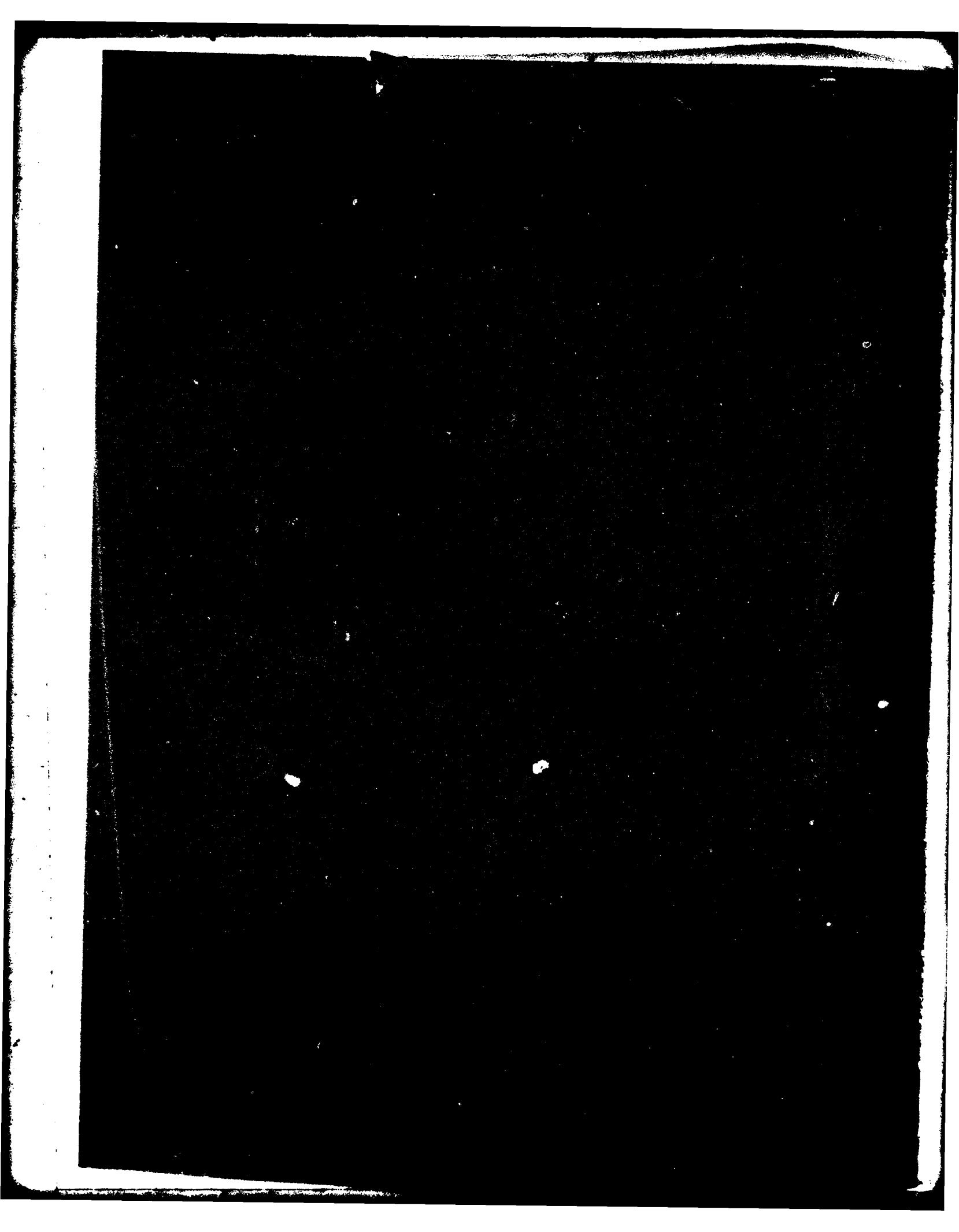
AD-A110 271 PAR TECHNOLOGY CORP ROME NY F/G 9/2
SABERS. STAND-ALONE ADIC BINARY EXPLOITATION RESOURCES SYSTEM. --ETC(U)
SEP 81 A J FRANKLIN, R L CALDWELL, S COLE F30602-78-C-0078
UNCLASSIFIED RADC-TR-81-250-VOL-1 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-81-250, Vol I (of three)	2. GOVT ACCESSION NO. AD-A110278
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): SABERS, STAND-ALONE ADIC BINARY EXPLOITATION RESOURCES SYSTEM, Volume I	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, April 78 - January 81
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Albert J. Franklin, Randy L. Caldwell, Scott Cole, Thomas L. McGibbon, Kathy H. Michel, James R. Wilson	8. CONTRACT OR GRANT NUMBER(s): F30602-78-C-0078
9. PERFORMING ORGANIZATION NAME AND ADDRESS: PAR TECHNOLOGY CORPORATION, 228 Liberty Plaza, Rome NY 13440	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 64750F, 19550114
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRDT), Griffiss AFB NY 13441	12. REPORT DATE: September 1981
	13. NUMBER OF PAGES: 124
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

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)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Garry W. Barringer (IRDT)

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

Data Base Management Systems
Transaction Processing
Graphic Systems
Sperry-Univac 1652 Terminal
ADCOM Applications
Orbital Mechanics

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

The SABERS development effort has been to design and implement a cohesive system for the Aerospace Defense Command (ADCOM) to provide an upgraded and improved analyst capability for the ADCOM Intelligence Center (ADIC) and its missions. In addition, SABERS has developed system software (such as a data base management system, a user interface, and a graphics package) to support current and future ADIC application needs. The SABERS application system provides an upgraded capability for the ADIC analyst, utilizes data currently available within the ADIC, is general and flexible in its

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

4/2 6.0

use, and is designed to minimize the amount of information the analyst has to enter into the system. The application functions implemented are built around a set of ten (10) data bases which are directly accessible by the analysts. The functions include a number of numerical and graphical applications. System software that is part of the current SABERS implementation includes a data base management system (DBMS), a user interface, and a graphics package. Goals reached in the DBMS development include the ideal that the application programmer's software interface to the SABERS DBMS should be at a high enough level such that the programmer can easily describe to the DBMS the information content of his data base, easily create the data base, and then easily access the information in the data base. Furthermore, powerful data base search and retrieval capabilities are part of the DBMS. Data base management applications provide a generalized capability for examining, updating, adding to or deleting information contained in the data bases. Goals realized by the user interface subsystem include the ideal that the application programmer's software interface to the SABERS user's terminal is to be at a high enough level that the programmer does not have to concern himself with the idiosyncrasies of the terminal. It should be easy for the programmer to describe to this interface the format of the display to be presented to the user. It should be easy for the interface to present the display to the user and to receive inputs from him. Finally, it should be easy for the programmer to access the inputs. The primary goal of the graphics package which is realized in SABERS is the ideal that an application programmer should be able to describe a picture to the graphics package using data values he understands. The graphics package performs all the necessary transformations to map a picture from the user's coordinate system into the terminal's coordinate system. The graphics package is also as terminal-independent as possible. A major part of the SABERS effort was the development of software for the Sperry-Univac 1652 terminal. This development involved designing and implementing code within the 1652 to interface it with the SABERS computer system (the VAX 11/780) as well as designing and implementing the code to control the terminal.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This report contains the final summary of the work accomplished under Contract F30602-78-C-0078, entitled the "Stand-Alone ADIC Binary Resource System", referred to as SABERS. This report is in compliance with Data Item 007 of contract Line Item 0002 of this contract.

The total documentation of the accomplishments realized in the performance of the SABERS contract is being provided in two separate documents:

o This final report, which provides:

1. The description of the design objectives of the software and of the numerical algorithms (the body of the report).

2. The information required by the users to employ the functional tools and to maintain the software (the User Manuals contained in Appendices A-F).

o The computer program documentation, which describes:

1. The particulars of the routines and subroutines.

2. The schema of each of the data bases.

1.1 BACKGROUND

The purpose of the SABERS effort was to develop a set of tools to demonstrate improvements to the Aerospace Defense Command (ADCOM) intelligence analyst at the ADCOM Intelligence Center (ADIC). SABERS requirements included the ability to review past and current events using a computerized data base,

mathematical analysis of space phenomena, graphical presentation of the results of the analysis, and uniform and efficient user interaction. Additional constraints were portability, terminal independence, structured programming, and documentation.

SABERS uses existing software, where possible. In addition, the newly developed software is portable within the I&W community (AN/GYQ-21(V) environment). The software was developed on the Digital Equipment Corporation (DEC) VAX 11/780 computer in compatibility mode. Compatibility mode emulates the DEC PDP 11/70 minicomputer and the RSX-11M operating system.

Terminal independence is accomplished in SABERS through the existence of interface utilities which translate between software and hardware protocol. This mechanism allows uniform software coding for input and output at the application level, and defers terminal considerations to the interface software.

SABERS software is designed to be modular. Although the programming languages are FORTRAN and assembler, the code produced is as structured as possible. Documentation is provided at the program level in the program listings, at the system level in the computer program documentation and at the user level in this report.

1.2 POTENTIAL USERS OF SABERS

SABERS documentation addresses three classes of users, the Space and Missile Analyst (SMA), the application programmer, and the program maintainer. The SMA is the ADIC analyst using the end product to research past events and to analyze present situations. The application programmer is the designer and coder of new applications to be included in SABERS. The program maintainer is the systems programmer modifying and updating the existing SABERS code.

The knowledge requirements of each user are distinct. The SMA must know how to cause SABERS to execute its applications, and how to interact with the application routines. The application programmer must know what the existing software accomplishes, and how to call the subroutines and functions he needs. The program maintainer must know how the routines perform their functions and how modification will affect the other software.

1.3 SUMMARY OF THE FUNCTIONAL AREAS

The result of the SABERS contract is a complementary structure of routines in six functional areas: applications, map drawing, user interaction, graphics, data base management, and terminal control. The software developed in each of these areas is designed to interact with and support all of the other areas. This complementary structure provides a testbed for the SMA to evaluate the effectiveness of computer aided research and analysis. In providing this service, the complementary structure functions as a preliminary system design; therefore, it is referred to as the SABERS system throughout the documentation.

Each of the six functional areas of the SABERS system is designed to provide capabilities which may be easily modified and expanded. With the exception of the applications and map drawing functional areas, the collection of software composing a functional area may be extracted as a unit from the complementary structure and integrated within some different software structure. The software composing each functional area provides a basis for the corresponding unit which is to be implemented in an operational SMA system.

1.3.1 Applications

SABERS applications software is composed of routines that are developed to solve problems of interest to the SMA. The problems are related to data organized in the following areas:

- summary information about all current and historical launch events
- description of characteristics and capabilities of all launch vehicles
- description of characteristics and capabilities of all launch pads
- description of characteristics and capabilities of Blue tracking radar systems
- description of characteristics and capabilities of Blue spaceborne systems
- description of the status of all objects in space
- description of characteristics and capabilities of Red support facilities
- collection of orbital element sets collected by Blue radar sensors
- collection of IR values collected by Blue IR sensors
- collection of polynomial coefficients reported by Blue IR sensors

The problems include reviewing and updating the data, performing analysis on the data to compute the parameters of space phenomena (such as the time and location of a space launch), and presenting the data and analytic results in a graphic format (such as the location of Red support facilities and the ground trace of an orbiting payload).

The application routines act as executive routines. These routines make calls to the routines in the remaining functional areas to draw maps, request data from the SABERS data bases, interact with the SMA, and plot graphical outputs.

1.3.2 Map Drawing

The map drawing routines provide the computation necessary to draw the different map projections required as background for some application plots. The routines rely upon the data base routines to access the map data and upon the graphics routines to plot the maps.

1.3.3 User Interaction

The SMA interacts with the applications through the Terminal Independent Transaction Processor (TITP). TITP presents a form to the analyst with the names of the data required and possible default values. Using the screen editing capabilities of TITP, the SMA enters the desired values and sends the data to the application. TITP performs error checking before sending the information back to the application.

The user interaction routines allow the application programmer to describe the format of an entire screen image; that is, to describe the location of the fields within the screen, the data types of each field and the legal values of each field. TITP also allows the application to address, modify, and read the fields of a screen image by name.

1.3.4 Graphics

Applications cause graphic output to be produced by calling routines in the Terminal Independent Graphics Processor (TIGP). TIGP is a direct implementation of the ACM-SIGGRAPH Core proposed graphic system standard.

1.3.5 Data Base Management

Applications manipulate the data through the SABERS Data Base Management System (DBMS). The data is organized into data bases managed by the DBMS. These data bases are easily defined and created by the application programmer

using the provided data description language. DBMS provides capabilities to add, update, and delete information within the data bases. DBMS supports complex assertions on multiple key fields in performing data base searches.

The SABERS DBMS is developed using DEC RMS-11, capable of searching an indexed sequential file based on a single key. The SABERS implementation includes a FORTRAN interface to RMS-11, complex multi-key queries, multiple indexed sequential files and multi-user operations.

1.3.6 Terminal Control

The terminal identified as the primary analyst terminal is the Sperry-Univac 1652 (S-U 1652) terminal. The first known interface between the S-U 1652 and the VAX was developed for SABERS using the DZ11-A asynchronous multiplexer for EIA RS-232 terminals, making the S-U 1652 look like any teletype device to the VAX, and allowing the use of the DEC supplied device driver for communication.

In addition to supporting the communication protocol, the developed software supports the programming of the variable function "soft keys." The soft key is programmed with a sequence of characters, which are transmitted to the VAX for interpretation whenever the analyst presses the soft key. Software is also provided to download predefined soft key definitions from the VAX.

1.4 SUMMARY OF THE DOCUMENT

The organization of this report is based upon the organization of SABERS. The documentation of each functional area is provided in an appendix which may be extracted from the report at the same time the software unit is extracted from the SABERS system. The information appropriate for each type of user is provided in the appendix for the functional area.

The remainder of this report is divided into 2 sections, 6 appendices, and one attachment:

- Section 2 describes the software architecture. It describes the interaction required by the functional areas in the complementary structure.

- Section 3 describes the mathematical algorithms used in the SABERS applications.

- Appendix A is the Space and Missile Analyst User Manual, describing the operation of the SABERS system and the user's interaction with the applications in the application functional area.

- Appendix B is the Applications Programmer User Manual and Program Maintenance Reference Manual for the SABERS map drawing functional area.

- Appendix C is the Applications Programmer User Manual and Program Maintenance Reference Manual for the Terminal Independent Transaction Processor (TITP), the SABERS user interaction functional area.

- Appendix D is the Applications Programmer User Manual and Program Maintenance Reference Manual for the Terminal Independent Graphics Processor (TIGP), the SABERS graphics functional area.

- Appendix E is the Applications Programmer User Manual and Program Maintenance Reference Manual for the Data Base Management System (DBMS), the SABERS data base management functional area.

- Appendix F is the Program Maintenance Reference Manual for the Sperry-Univac 1652 terminal, the SABERS terminal control functional area.

- Attachment 1 is a copy of the FLECS User Manual. This is provided for application programmers and program maintainers who may wish to write new applications in FLECS, or who may need to read the source code of SABERS.

2. SABERS SOFTWARE ARCHITECTURE

This section describes the method of interaction between the functional units composing the SABERS system. Section 2.1 presents the system overview. Section 2.2 discusses the interprocess communication mechanism, and Section 2.3 discusses the use of data bases and data files.

2.1 SABERS SYSTEM OVERVIEW

The process image size under the RSX-11M operating system is 32k words. Since the total storage requirements of all the SABERS software is much greater than this limit, the software is modularized into several independent utility processes which operate concurrently with the application processes in support of the applications. These utility processes communicate with the application processes through the use of the Macro-11 Assembler interface described in Section 2.2.

Two of these utility processes coordinate the multi-user access to the SABERS data bases. These utilities are the Data Base Manager (DBM) and the Memory Management Service (MMS). The DBM processes the data base access requests, and the MMS manages the tables necessary for DBM to work. These two utilities are invoked once to support all SABERS users, and their resources are shared among all applications. These utilities together form the Data Base Management System (DBMS).

The remaining two utilities are the Terminal Independent Transaction Processor (TITP) and the Terminal Independent Graphics Processor (TIGP). A unique local copy of each utility is invoked for each user. The local copy includes the Device Manager (DM) appropriate for the user's terminal. The DM is a Macro-11 Assembler interface for the type of terminal on which the user is logged in.

The SABERS account is maintained on the computer in the form of a directory tree. The root of the tree is the SABERS default directory. This is the directory in which the DBM and MMS utilities are initiated. A subdirectory (or branch) off the SABERS root is maintained for each recognized SABERS user. The user initiates the execution of applications routines from his subdirectory. The local copies of the user's TITP and TIGP utilities are also initiated in this subdirectory.

The user's SABERS environment is established at the time the user "logs-on" to the system (the log-on procedure is described in Appendix A, the Space and Missiles Analyst User Manual). If DBM and MMS are not currently running, they are initiated in the SABERS root directory as detached (ownerless) shareable processes. After the user is verified to SABERS through the use of a system password, the correct default user subdirectory is established. The versions of TITP and TIGP with the DM for the user's terminal type are copied from the SABERS root directory to the user's default subdirectory and invoked. SABERS applications may then be initiated by the user. The requested application is copied from the SABERS root directory to the user's default subdirectory and initiated. This guarantees that the application process is local and unique to the user.

An application process expires as soon as its task is completed. The local copy of the application is deleted from the user's default subdirectory to prevent the buildup of files in the user's area. The local copies of the TITP and TIGP processes expire and are deleted when the user logs off from the system. The DBMS utilities (DSM and MMS) are ownerless, and are halted by SABERS when the number of SABERS users reaches zero. Thus the first SABERS user may log off without affecting any other user.

The system environment for three users is pictured in Figure 2-1. In this figure,

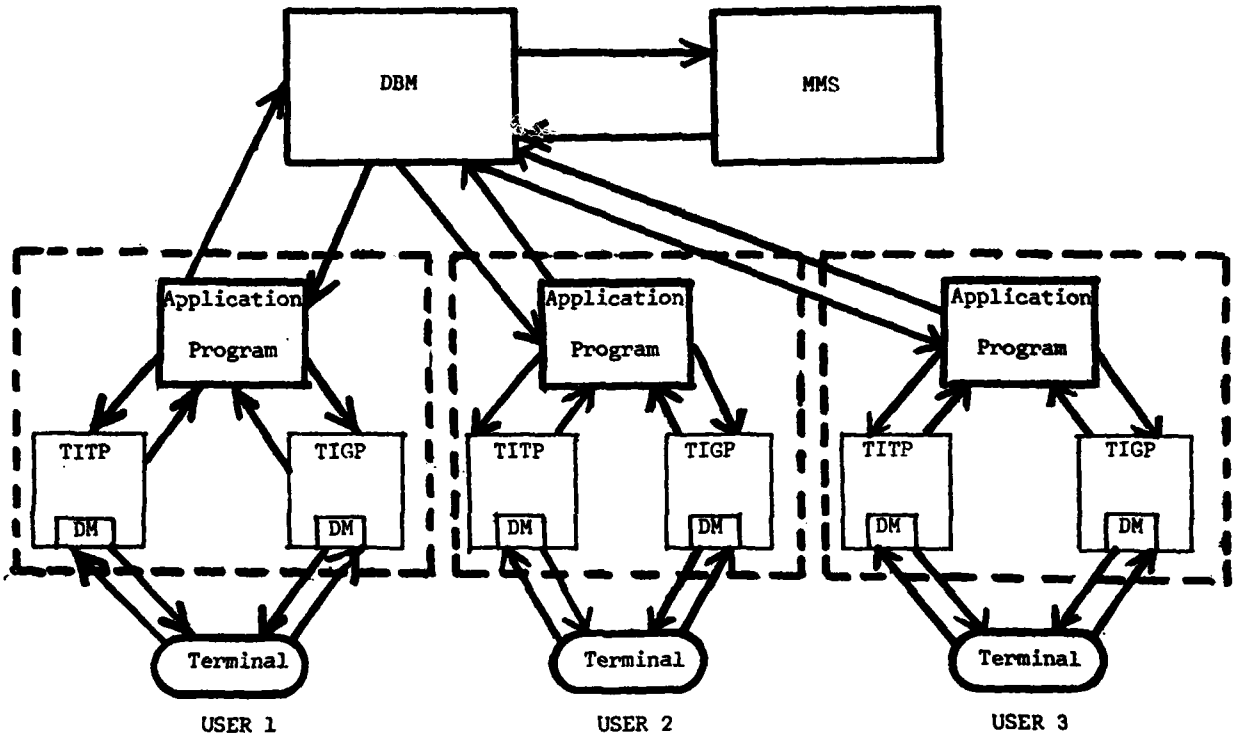


Figure 2-1 The Current SABERS System Organization

- o solid rectangles are independent processes
- o processes enclosed in dashed boxes are operating under the user's default subdirectory
- o DBM and MMS are operating under the SABERS root directory
- o DM is a terminal specific module required to interface the utility to the user's terminal
- o arrows show the lines of communication.

2.2 THE INTERPROCESS COMMUNICATION MECHANISM

The utility processes communicate with an application process through a Macro-11 Assembler interface. This interface makes use of the system service message sending capabilities and the global event flag capabilities of the RSX-11M operating system.

The interface modules are part of the application processes, and are constructed to make the communication mechanism transparent to the application programmer. The application program includes calls to the utilities just as if the utilities were physically part of the application process. The interface intercepts these subroutines calls and transforms them into messages directed to the appropriate utility process. The interface copies the parameters of the subroutine call into a message buffer and sends the message buffer in 13-word packets to the utility process, using the Send Data System Service. After the message is sent, the application process "hibernates" (takes up no system resources).

The operating system delivers the message to the utility by generating an Asynchronous System Trap which awakens the hibernating utility. (Note that placing inactive processes in a state of hibernation allows the operating system to manage the active executing processes more efficiently). The incoming message is decoded by the interface into a standard FORTRAN call which is then executed. Thus, the communication interface is also transparent

to the utility programmer. When the utility is finished executing, the results are made available to the calling application through the transmission of a message through the operating system.

One further refinement is required in interfacing with the DBM because of its global nature. Since requests for data base operations may be made by several applications, a synchronization mechanism to insure security of access is provided by the global event flag capability of the operating system. The application interface waits until the global flag shows that the DBM is available, sets it to show the DBM is not available, and sends the message. The application interface resets the global flag after receiving the data transfer from the DBM.

2.3 DATA BASES AND DATA FILES

Information necessary for the operation of SABERS is stored in data bases and data files. Global information available to all users is stored in the SABERS root directory. Global data bases contain the map data (coastlines and political boundaries) and the data organized to support the applications developed to solve problems of interest to the SMA (as presented in Section 1.3). Any changes to these global data bases affects all other SABERS users.

Local information, which is not accessible to any other SABERS users, is maintained in the individual user's default subdirectory. In addition to any local data base required by applications, local data bases exist to store the map plotting parameters, the current launch event identification number, and the last record reviewed. Local data files are used by the local copy of TIGP to store retained segments, and by the local copy of TITP to store screen display formats and screen responses.

Data bases and data files established and maintained by the utilities are transparent to the user and the application programmer. The information stored is used to perform calculations, to provide default responses, to aid

graphics, to format and display transaction screens, and to check user responses.

2.4 STRUCTURED PROGRAMMING

To support structured programming in a FORTRAN environment, the FORTRAN Language Extended Control Systems (FLECS) preprocessor has been employed where feasible. Such SABERS facilities as TIGP, TITP, the map drawing routines, the Data Base Management System, and some applications, have been programmed using FLECS. A copy of the FLECS User's Manual is included with this report.

FLECS expands the FORTRAN language by making the control structures recommended by modern programming practices available to the FORTRAN programmer. These structures include IF, UNLESS, WHEN . . . ELSE, CONDITIONAL, SELECT, DO, WHILE, REPEAT WHILE, UNTIL, and REPEAT UNTIL. In addition, editorial features, such as indenting, are provided to improve the appearance and readability of printed programs.

The FLECS preprocessor accepts programs which use these conventions and outputs code which conforms to FORTRAN 66 standards for compilation in a production compiler. FLECS was made available without charge to this contract and is available without charge to SABERS as implemented on the AN/GYQ-21(V).

3. SABERS APPLICATION ALGORITHMS

This section describes the mathematical algorithms developed for SABERS. The time algorithms presented in Section 3.1 and the coordinate systems discussed in Section 3.2 are used in almost all of the SABERS applications. Although the calculation of a satellite ground trace is important in many of the applications, the algorithms presented in Section 3.3 are directly used in the OVERLAY GROUND TRACE and OVERLAY TIME MARKS ON GROUND TRACE applications. The solution of the radar related problem discussed in Section 3.4 are the basis for the RADARS VS. ORBIT applications. The algorithms developed in Section 3.5 for the photo reconnaissance problems are used in the SATELLITE RECONNAISSANCE applications. The equations presented in Section 3.6 describe the algorithm used in the GENERATE THREAT WINDOWS application. The equations used to generate and plot points on the different map backgrounds for the application outputs are presented in Section 3.7.

3.1 TIME ALGORITHMS

Time is represented in SABERS by two values, the calendar day and the clock time since midnight. The user enters and receives the calendar day as the Gregorian date (year, month and day), or, alternatively, as (year, day number), in which the month and day information have been combined into the day number. The clock time is presented and received by the user as mean solar time (hour, minute, second) in 24-hour format.

The algorithms expect the calendar day to be encoded as a Julian date relative to January 0, 1900 at midnight, and the clock time to be the fractional part of the day since midnight. The benefits derived are simple time difference calculations, and the ability to express the exact time in one value (the complete Julian day = Julian date + fractional day).

3.1.1 Julian to Gregorian Date Conversion

The conversion between the user representations of time and the algorithm representations of time are based upon two algorithms presented by Henry F. Fliegel and Thomas C. Van Flandern in the Letters to the Editor section, page 657, of reference [4]. The algorithms were presented as a FORTRAN arithmetic statement function and as a FORTRAN subroutine.

3.1.1.1 Gregorian to Julian Date Conversion

The Julian date arithmetic statement function as presented by the authors returns an integer Julian date at noon valid for any Gregorian date producing a Julian date greater than zero. The algorithm makes use of the truncation feature of integer arithmetic in FORTRAN.

```
JD (I, J, K) = K - 32075 + 1461 * (I + 4800 + (J - 14) / 12) / 4
             + 367 * (J - 2 - (J - 14) / 12 * 12) / 12
             - 3 * ((I + 4900 + (J - 14) / 12 / 100) / 4
```

where I = year, J = month (number from 1 to 12) and K = day of month.

This algorithm, when evaluated for December 31, 1899, yields JD = 2415020. This implies that the Julian date of January 0, 1900 at midnight is 2415020.5. Changing the function to a real-valued function, and recognizing that the time interval between two Julian dates is the difference of the two Julian dates, the SABERS algorithm to convert from (year, month, day) to Julian date is:

```
DAY (IYEAR, MONTH, IDAY) = IDAY - 32075
    + 1461 * (IYEAR + 4800 + (MONTH - 14) / 12) / 4
    + 367 * (MONTH - 2 - (MONTH -14) / 12 * 12) / 12                (3-1)
    - 3 * ((IYEAR + 4900 + (MONTH -  14) / 12 / 100) / 4
    - 2515020.5
```

3.1.1.2 Julian to Gregorian Day Number Conversion

A closed form expression may be derived from the FORTRAN expression (3-1) to convert from (year, day number) to Julian date. Letting JUL be the real-valued Julian date required, and DNUM be the day number, we want

```
JUL = DNUM + DAY (IYEAR - 1, 12, 31)
```

Substituting IYEAR = IYEAR - 1, MONTH = 12 and IDAY = 31 in (3-1), we find

```
JUL = DNUM + 365 * IYEAR + (IYEAR - 1) / 4
    - 3 * ((IYEAR - 1) / 100 + 1) / 4
    - 693960.5
```

again making use of the integer truncation feature of FORTRAN.

3.1.1.3 Julian to Gregorian Date Conversion

The second algorithm presented by the authors converts from the Julian date JD to the year, month, and day. Written in the form of a FORTRAN subroutine, it also makes use of the integer truncation feature of FORTRAN.

```
SUBROUTINE DATE (JD, IYEAR, MONTH, IDAY)
L = JD + 68569
N = 4 * L / 146097
L = L - (146097 * N + 3) / 4
I = 4000 * (L + 1) / 1461001
L = L - 1461 * I / 4 + 31
J = 80 * L / 2447
IDAY = L - 2447 * J / 80
L = J / 11
MONTH = J + 2 - 12 * L
IYEAR = 100 * (N - 49) + I + L
RETURN
END
```

The SABERS subroutine differs only in replacing the input integer Julian date JD with a real Julian Date D and adding the Julian date of January 0, 1900 at midnight. The first line of the subroutine is thus replaced by

```
SUBROUTINE DATE (D, IYEAR, MONTH, IDAY)
JD = D + 2415020.5
```

Of course, the Gregorian date may be expressed as (year, day number) by solving for the day number DNUM with (2 - 1) by

```
DNUM = DAY (IYEAR, MONTH, DAY) - DAY (IYEAR - 1, 12, 31)
```

3.1.2 Mean Solar Time-Fractional Day Conversion

The conversion between mean solar time and fractional day since midnight is straightforward. Let F = fractional day, H = hour (24-hour clock), M = minute and S = second, then

$$F = (H * 3600 + M * 60 + S) / 86400$$

and

$$H = [24 * F]$$
$$M = [1440 * F - 60 * H]$$
$$S = 86400 * F - 60 * H - 3600 * M$$

where [X] means the greatest integer function of X.

3.2 COORDINATE SYSTEMS

There are three coordinate systems used to describe a point's location. These are the earth-centered inertial (ECI), the geocentric, and the local topographic (ENU) coordinate system. An orbital element set is used to define a satellite's position, and will also be described in this section.

3.2.1 Earth-Centered Inertial Coordinate System

The earth-centered inertial (ECI) coordinate system has its origin at the center of the earth. The principal axis points toward the vernal equinox (denoted by γ). The x-y plane is in the earth's equatorial plane, and the z axis points to the north pole, completing the right hand system (See Figure 3-1).

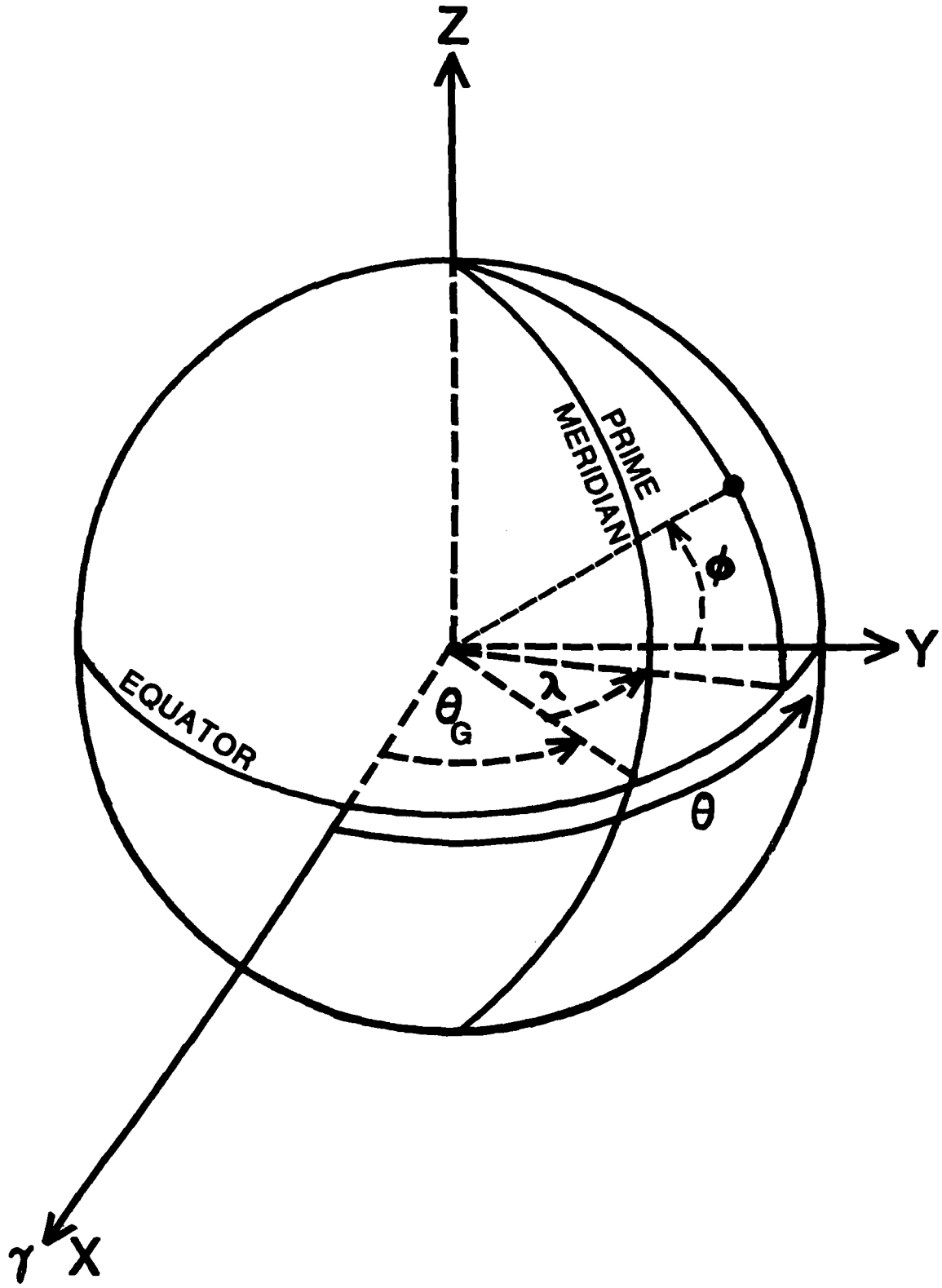
The ECI coordinate system is the reference system for the other systems in this section. This is a result of the characteristic of the ECI coordinate system that it does not change in its orientation in the time scales the SABERS algorithms deal with.

3.2.2 Geocentric Coordinate System

The geocentric coordinate system is a rotating frame of reference. The origin of the system is the center of the earth. The coordinates of a point are given as (λ, ϕ, h), where λ is the longitude, ϕ is the geodetic latitude, and h is the altitude of the point above the reference ellipsoid. As shown in Figure 3-1, the longitude is the angular deviation, measured at the equator, of the meridian passing through the point from the Prime Meridian (passing through Greenwich, England), $-180^{\circ} \leq \lambda \leq 180^{\circ}$. By convention, a positive longitude indicates that the point is to the east of Greenwich.

The earth is modeled as an ellipsoid generated by rotating an ellipse about the z axis. As shown in Figure 3-2, the line from a point P perpendicular to the reference ellipsoid at A will intersect the equatorial plane at B. The acute angle of intersection at B is defined as the geodetic latitude ϕ. The geocentric latitude ϕ' is defined as the angle between A and the equatorial plane measured at the center of the earth. These angles are related by

Figure 3-1 Earth-Centered Inertial and Geocentric Coordinate Systems



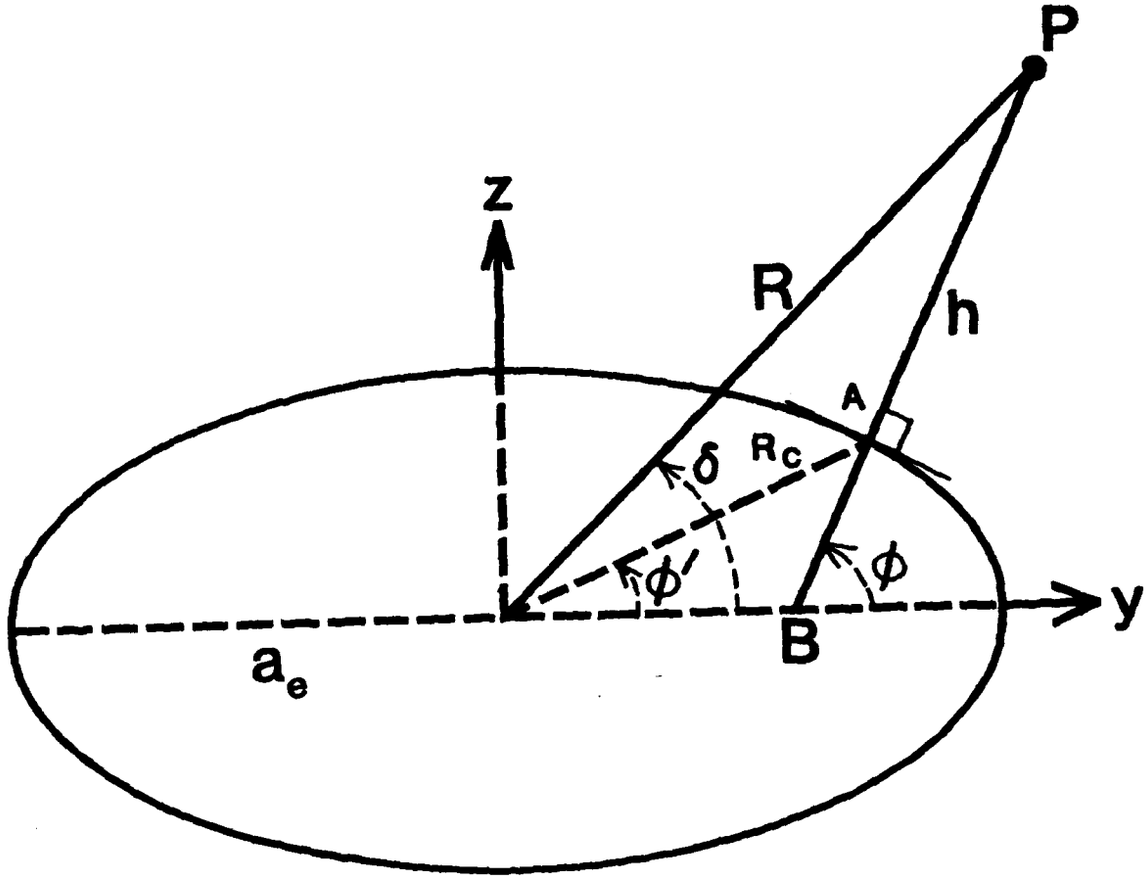


Figure 3-2 Ellipsoid Earth Model

$$\tan \phi = \frac{\tan \phi'}{(1 - f^2)} \quad , \quad 0^\circ \leq \phi \leq 90^\circ \quad , \quad 0^\circ \leq \phi' \leq 90^\circ$$

$$f = \frac{1}{298.25}$$

The distance along the perpendicular between P and A is defined as the altitude above the reference ellipsoid h.

3.2.2.1 Hour Angle

The geocentric coordinate system described above is a rotating system. The angular deviation of any point from the vernal equinox at time t is defined as the hour angle θ. As shown in Figure 3-1, $\theta = \theta_G + \lambda$, where θ_G is the hour angle of the Prime Meridian. As reported in reference [3], pages 20-21, the Greenwich hour angle may be closely approximated by

$$\theta_G = \theta_{G_o} + \Delta t \frac{d\theta}{dt}$$

where

$$\theta_{G_o} = 99.6909833^\circ + 36000.7689^\circ \, T_u + 0.00038708 \, T_u^2$$

with

$$T_u = \frac{\text{Julian date at midnight relative to January 0, 1900}}{36525}$$

Δt = fractional days since midnight

and

$$\frac{d\theta}{dt} = \frac{1}{365.24219879} \text{ revolutions/year}$$

$$= 360.9856473 \text{ degrees/day}$$

The hour angle θ, $0^o \leq \theta \leq 360^o$, defines the linkage between the rotating geocentric and non-rotating ECI coordinate systems.

3.2.2.2 Geocentric to ECI Transformation

The transformation from geocentric coordinates to ECI coordinates is taken directly from Transformation 4, pages 399-400, in reference [3] (See Figure 3-2).

$$\lambda, \phi, h, \theta_G \rightarrow x, y, z$$

$$\phi' = \tan^{-1} [(1 - f)^2 \tan \phi]$$

$$R_c^2 = \frac{a_e^2 [1 - (2f - f^2)]}{1 - (2f - f^2) \cos^2 \phi'}$$

$$R = \sqrt{R_c^2 + h^2 + 2R_c h \cos (\phi - \phi')}$$

$$\delta = \phi' + \sin^{-1} \left[\frac{h}{R} \sin (\phi - \phi') \right]$$

$$\theta = \theta_G + \lambda$$

$$x = R \cos \delta \cos \theta$$

$$y = R \cos \delta \sin \theta$$

$$z = R \sin \delta$$

where

a_e = semi-major axis of the earth

= 1 e.r. (earth radius)

= 6378.16 km.

3.2.2.3 ECI to Geocentric Transformation

The transformation from ECI to geocentric coordinates is taken directly from Transformation 3, pages 398-399, in reference [3] (See Figure 3-2).

$$x, y, z, \theta_G \rightarrow \lambda, \phi, h$$

$$R = \sqrt{x^2 + y^2 + z^2}$$

$$\alpha = \tan^{-1}\left(\frac{y}{x}\right)$$

$$\lambda = \alpha - \theta_G, \quad -180^\circ \leq \lambda \leq 180^\circ$$

$$\delta = \tan^{-1}\frac{z}{\sqrt{x^2 + y^2}}, \quad -90^\circ \leq \delta \leq 90^\circ$$

Starting with the estimate $\phi' = \delta$, the following equations are executed in an iterative manner until ϕ' is within the required tolerance.

$$R_c = a_e\sqrt{\frac{1 - (2f - f^2)}{1 - (2f - f^2)\cos^2\phi'}}$$

$$\phi = \tan^{-1}\left[\frac{\tan\phi'}{(1 - f)^2}\right]$$

$$h = \sqrt{R^2 - R_c^2 \sin^2(\phi - \phi')} - R_c \cos(\phi - \phi')$$

$$\phi' = \delta - \sin^{-1}\left[\frac{h}{R} \sin(\phi - \phi')\right]$$

3.2.3 Local Topocentric Coordinate System

The local topocentric coordinate system used by the SABERS algorithms is defined by pointing the principal e-axis towards local east, pointing the n-axis towards local north, and completing the right hand system by pointing the u-axis up (See Figure 3-3).

The transformation from ECI coordinates to local topocentric coordinates is given as follows:

$$\begin{pmatrix} e \\ n \\ u \end{pmatrix} = G\left[\begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} x \\ y \\ z \end{pmatrix}_0\right],$$

where $\begin{pmatrix} x \\ y \\ z \end{pmatrix}_0$ is the origin of the local topocentric system in ECI coordinates and G is the rotation matrix (see page 319 in reference [5] and page 3-5 in reference [8]),

$$G = \begin{vmatrix} -\sin\theta & \cos\theta & 0 \\ -\sin\phi' \cos\theta & -\sin\phi' \sin\theta & \cos\phi' \\ \cos\phi' \cos\theta & \cos\phi' \sin\theta & \sin\phi' \end{vmatrix}$$

where ϕ' is the geocentric latitude of the origin and θ is the hour angle of

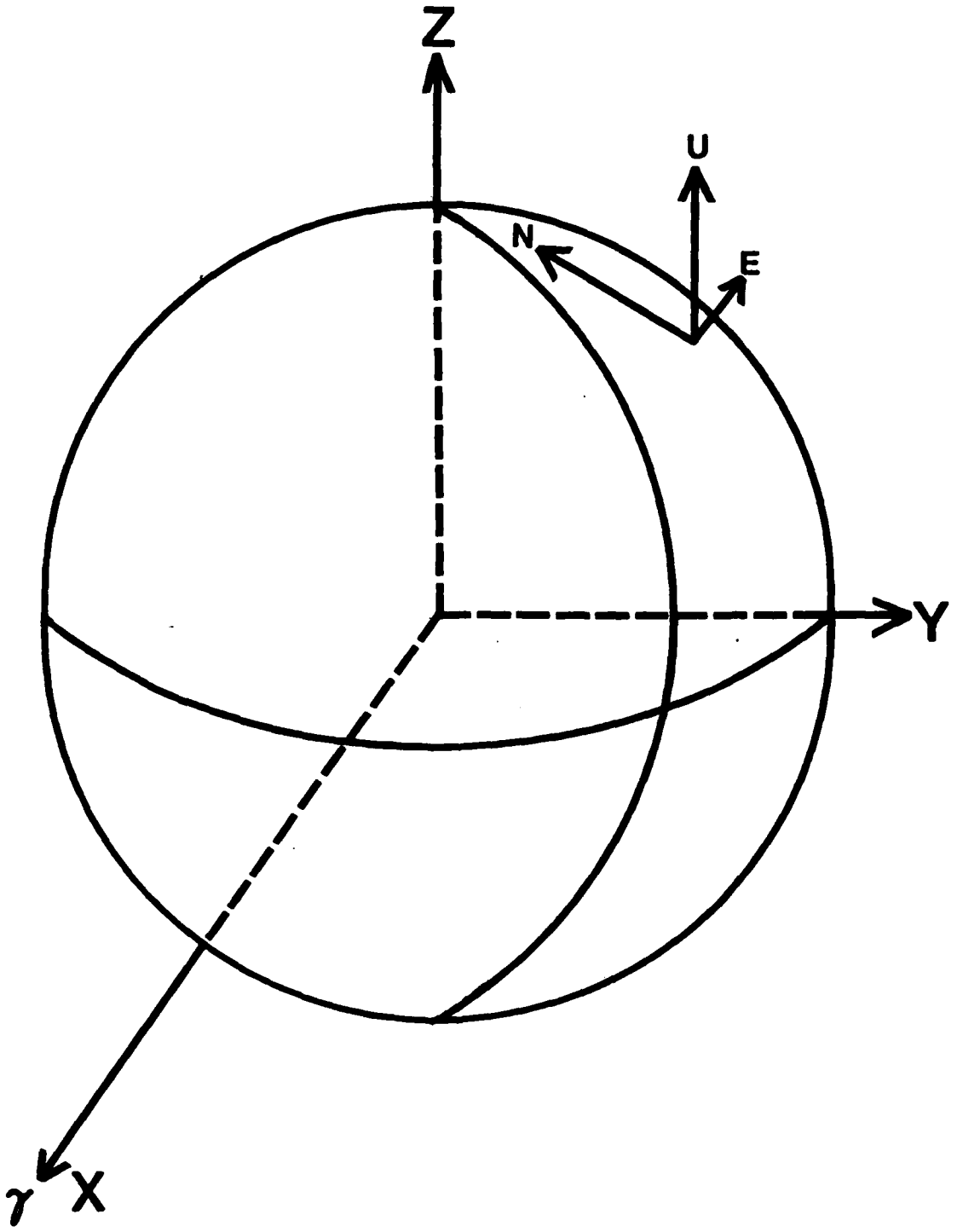


Figure 3-3 Local Topocentric Coordinate System

the origin. At time t, the location of the origin expressed in ECI coordinates determines the hour angle and the coordinates of the origin in the geocentric coordinate system via the algorithms presented in Sections 3.2.2.1, 3.2.2.2, and 3.2.2.3.

3.2.4 Orbital Element Set

The development presented in this section is due to Dr. Ranjan V. Sonalkar, as reported in reference [8], pages 3-1 to 3-21.

3.2.4.1 Description of the Orbital Elements

SABERS algorithms receive satellite orbit information in the set (T_E, Ω, e, i, ω, M, n, $\frac{\dot{n}}{2}$, $\frac{\ddot{n}}{6}$). The epoch time, T_E, is the time for which the orbital element set has been observed. As shown in Figure 3-4, the right ascension of the ascending node, Ω, is an angle measured in the earth's equatorial plane. It is defined by the vernal equinox and the ascending node (the intersection of the orbit plane and the equatorial plane for which the satellite is ascending from the southern hemisphere to the northern hemisphere). The eccentricity of the orbit, e, is assumed to be $0 \leq e < 1$, defining an elliptic orbit with the center of the earth at one focus. The inclination, i, is the angle between the equatorial plane and the orbit plane at the ascending node. The argument of perigee, ω, is the angle measured in the plane of the orbit defined by the ascending node and the periapsis (the point of the satellite's closest approach to the center of the earth). The mean anomaly, M, indicates the satellite's position in its orbit. The mean motion, n, measured in revolutions per day, contains information about the period of the orbit and the semi-major axis of the orbit ellipse. The quantities $\frac{\dot{n}}{2}$ and $\frac{\ddot{n}}{6}$ are the first and second time derivatives of the mean motion, respectively.

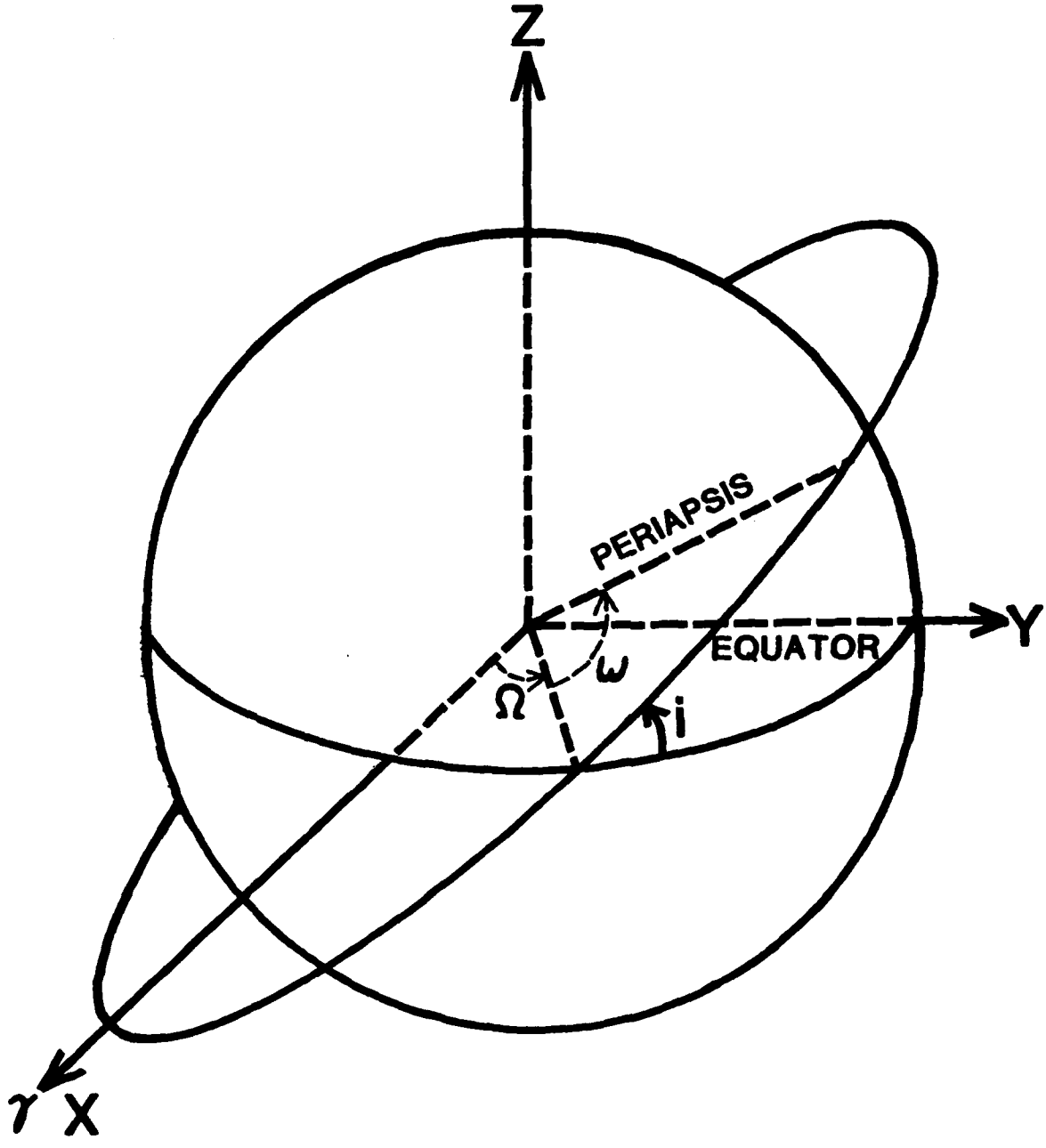


Figure 3-4 Orbital Element Set

3.2.4.2 Calculation of Satellite Position

Given a satellite orbital element set, the algorithm to calculate the satellite's ECI position and velocity vectors, X and $\dot{X}$, for Δt from epoch time, T_E, begins by calculating the value of the semi-major axis (in earth radii) at epoch.

$$a = \sqrt[3]{\frac{\mu}{n^2}} \left(1 + \frac{1}{3}\delta - \frac{1}{3}\delta^2\right)$$

where

$$\delta \triangleq \frac{-\frac{3}{2} J_2 \left(1 - \frac{3}{2}\sin^2 i\right)}{\left(\frac{\mu}{n^2}\right)^{\frac{2}{3}}(1 - e^2)^2}$$

μ = gravitational constant of the earth

= 398600.5 km^3/sec^2

J_2 = second zonal harmonic coefficient of the geopotential function

= 0.00108248

Next, the secular perturbations are introduced.

$$\Omega = \Omega_o - \frac{\frac{3}{2} J_2 n \cos i}{a^2 (1 - e^2)^2} \Delta t$$

$$\omega = \omega_o + \frac{\frac{3}{2} J_2 n (2 - \frac{5}{2} \sin^2 i)}{a^2 (1 - e^2)^2} \Delta t$$

where Ω_o and ω_o are the unperturbed values of the orbited element set, and Ω and ω are the perturbed values. Then the eccentric anomaly, E, is solved for, using the Newton-Raphson iteration method for Kepler's equation,

$$M = E - e \sin E$$

for

$$M = M_o + n \Delta t + \frac{\dot{n}}{2} \Delta t^2$$

where M_o is the mean anomaly at time $t = T_E$ and M is the mean anomaly at time $t = T_E + \Delta t$. The following are solved for:

$$L \text{ (true argument of latitude)} = 2 \tan^{-1} \sqrt{\frac{1 + e}{1 - e}} \tan \frac{E}{2} + \omega$$

$$R \text{ (range)} = a (1 - e \cos E)$$

$$\dot{R} \text{ (radial component of velocity vector)} = \sqrt{\mu a} \frac{e}{R} \sin E$$

$$R\dot{v} \text{ (transverse component of velocity vector)} = \frac{\sqrt{\mu a (1 - e^2)}}{R}$$

Defining unit vectors $\vec{U}$ in the radial direction and $\vec{V}$ perpendicular to the radial vector in the direction of the transverse component as

$$\underset{\rightarrow}{U} = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} \cos L \cos \Omega - \sin L \sin \Omega \cos i \\ \cos L \sin \Omega + \sin L \cos \Omega \cos i \\ \sin L \sin i \end{pmatrix}$$

$$\underset{\rightarrow}{V} = \begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix} = \begin{pmatrix} -\sin L \cos \Omega - \cos L \sin \Omega \cos i \\ -\sin L \sin \Omega + \cos L \cos \Omega \cos i \\ \cos L \sin i \end{pmatrix}$$

we have

$$\underset{\rightarrow}{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = R \cdot \underset{\rightarrow}{U}$$

$$\dot{\underset{\rightarrow}{X}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \dot{R} \cdot \underset{\rightarrow}{U} + R\dot{\Psi} \cdot \underset{\rightarrow}{V}$$

3.3 SPACE OBJECT GROUND TRACE

The orbital element set can be used to calculate space object ground traces. This section presents algorithms for calculating the ground traces of satellites and ballistic missiles.

3.3.1 Satellite Ground Trace

The satellite ground trace is the locus of points described by the satellite sub-orbital over a given time period $\Delta t = \sum_i \Delta t_i$. Given a satellite's position, the satellite sub-orbital is defined as the point of intersection of the line through the satellite position perpendicular to the

reference ellipsoid with the ellipsoid. Given a set of Δt_i from epoch T_E, and the orbital element set, (T_E, Ω e, i, ω M, n, $\frac{\dot{n}}{z}$, $\frac{\dot{n}}{6}$), the set of geocentric ground trace points to be plotted may be generated by the methods of Section 3.2.4.2 followed by the transformation of Section 3.2.2.3 for each Δt_i.

Dividing the time span desired into equal time segments is instructive in that the ground distances drawn indicate the proportion of time each area of the earth is under surveillance. The set of ground trace points generated this way do not produce smooth curves for non-circular orbits, however. The following algorithm generates a set of Δt_i such that the ground coverage distances plotted will be approximately equal in length.

The true anomaly, ν, is the central angle from periapsis to the satellite's position. Obviously, dividing 360^o into N equal true anomaly angles implies that equal ground distances will be covered on a spherical earth (See Figure 3-5). However, the expression which will generate the Δt_i is in terms of the eccentric anomaly E. The eccentric anomaly is defined as the angle (measured from the center of the earth) between the periapsis and the intersection of the circle circumscribing the orbit ellipse with the line through the satellite perpendicular to the semi-major axis (See Figure 3-6). The relationship between ν and E is given on page 39 in reference [2] by

$$\tan \frac{\nu}{2} = \sqrt{\frac{1 + e}{1 - e}} \tan \frac{E}{2}$$

and the Δt_i is given on page 1 of the same reference by

$$\Delta t = \frac{2 \left(\frac{E_2 - E_1}{2} - e \sin \frac{E_2 - E_1}{2} \cos \frac{E_2 - E_1}{2}\right)}{n}$$

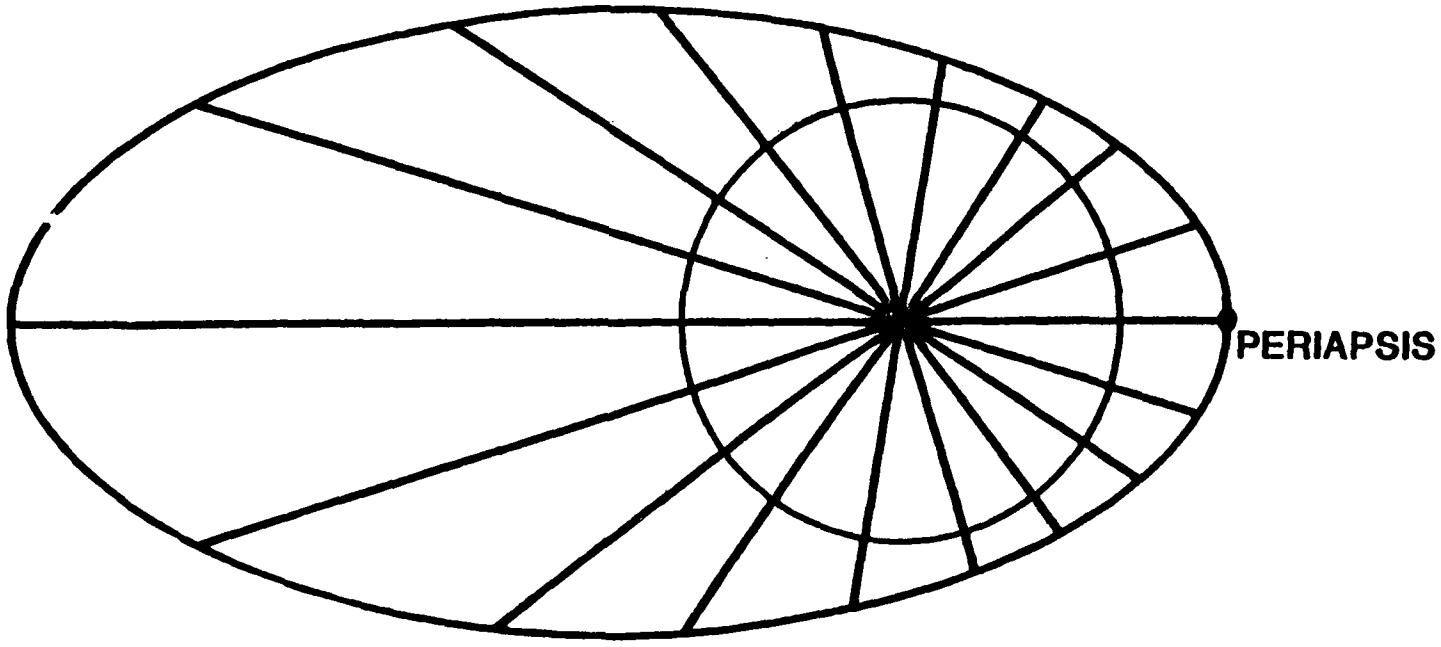


Figure 3-5 Using Equal True Anomalies to Produce Equal Ground Distances

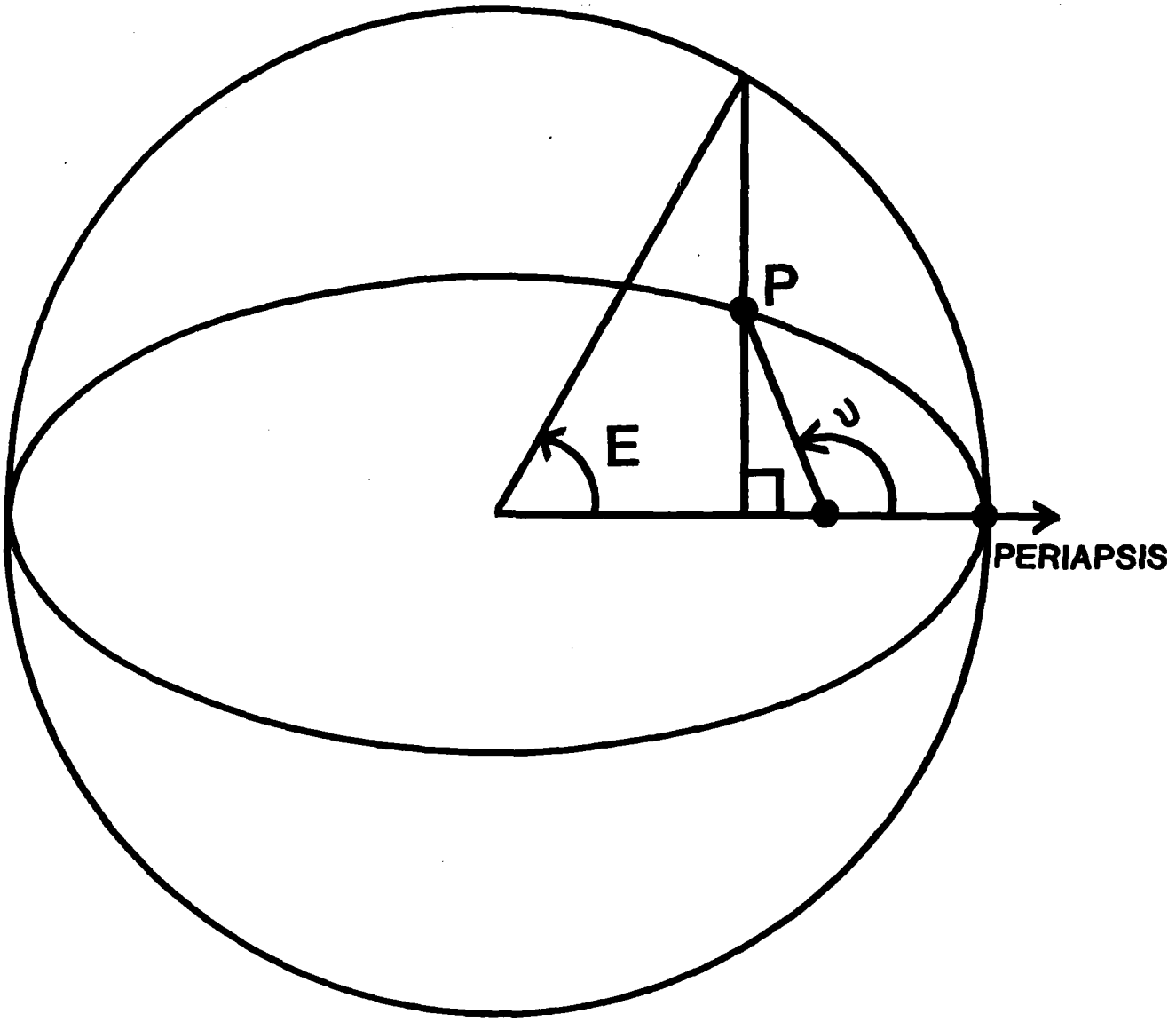


Figure 3-6 Definition of Eccentric Anomaly

The algorithm proceeds as follows:

1. Find the eccentric anomaly of the satellite at the start time by the method of Section 3.2.4.2, $0^\circ \leq E \leq 360^\circ$

2. Calculate the corresponding true anomaly as

$$\nu_o = 2 \tan^{-1}\left[\sqrt{\frac{1+e}{1-e}} \tan \frac{E}{2}\right]$$

3. Let $\Delta \nu$ be the increment of true anomaly. Then the table of true anomalies may be generated as

$$\nu_i = \nu_o + i \cdot \Delta \nu \ , \ i = 0 \text{ to } N.$$

4. The corresponding table of eccentric anomalies is generated as

$$E_i = 2 \tan^{-1}\left[\sqrt{\frac{1-e}{1+e}} \tan \frac{\nu_i}{2}\right] \ , \ i = 0 \text{ to } N$$

5. Finally, the tables of Δt_i may be calculated as

$$\Delta t_i = \frac{2\left(\frac{E_i - E_{i-1}}{2} - e \sin \frac{E_i - E_{i-1}}{2} \cos \frac{E_i - E_{i-1}}{2}\right)}{n}$$

for i = 1 to N.

3.3.2 Ballistic Missile Ground Trace

The ground trace of a ballistic missile defined by $(T_E, \Omega, e, i, \omega)$ launched at location L (λ, ϕ, h) requires the algorithms described in this section. If the predicted point of impact is known (say from TSATS), then the

beginning and end points are known. If the predicted point of impact is not known, then it must be estimated as follows.

As explained in reference [1] on pages 279 - 297, the orbit of a ballistic missile is an ellipse in its non-powered phase. Due to propulsion forces and atmospheric and gravitational effects, this is not true before burnout and after reentry (See Figure 3-7). However, the free-flight trajectory is symmetrical, and half of the free-flight range angle, Ψ, lies on each side of the semi-major axis. Since no time of burnout and time of reentry information is available, the assumption is made that the complete ballistic path is a part of an ellipse symmetric about the semi-major axis.

The true anomaly, ν_s, is calculated to the point of launch. The true anomaly, ν_T, is calculated to the predicted point of impact (if known), or to be $360^\circ - \nu_s$ otherwise. Values for the semi-major axis and mean motion are calculated, appropriate Δt_i are calculated, and then the method of Section 3.2.4.2 followed by the transformation of Section 3.2.2.3 is used to generate the ground trace points.

The launch point true anomaly ν_s is calculated as follows. Let $\overrightarrow{LP}$ be the vector in ECI coordinates of the launch point. $\overrightarrow{LP}$ is calculated by the transformation of Section 3.2.2.2. Let $\overrightarrow{P}$ be the vector in ECI coordinates of periapsis. As defined on page 77 in reference [3],

$$\overrightarrow{P} = \begin{pmatrix} \cos \omega \cos \Omega - \sin \Omega \sin \omega \cos i \\ \cos \omega \sin \Omega + \sin \Omega \cos \omega \cos i \\ \sin \omega \sin i \end{pmatrix}$$

Then,

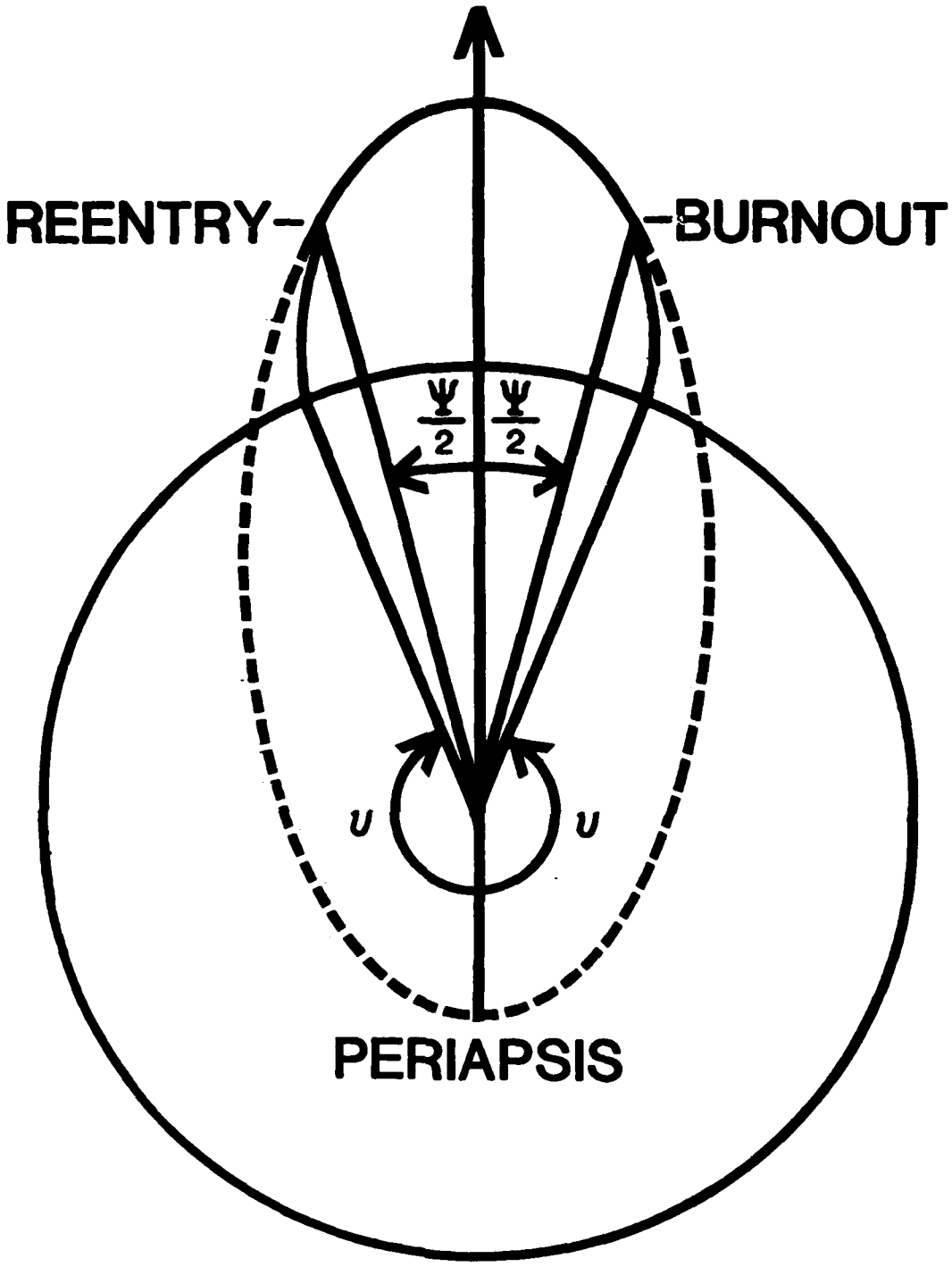


Figure 3-7 Ballistic Missile Trajectory

$$\nu_s = \cos^{-1} \frac{\left| \vec{P} \cdot \vec{LP} \right|}{\left| \vec{P} \right| \left| \vec{LP} \right|}$$

The equations on page 20, $r = \frac{a\,(1 - e^2)}{1 - e \cos \nu}$ and page 187, $\cos \nu = \frac{e - \cos E}{e \cos E - 1}$, in reference [1], may be solved for the semi-major axis a and the eccentric anomaly at the launch point E_S.

$$a = \frac{\left| \vec{LP} \right| (1 - e \cos \nu_s)}{1 - e^2}$$

$$E_S = \cos^{-1} \left(\frac{e + \cos \nu_s}{1 + e \cos \nu_s} \right)$$

The ballistic missile element set may be extended to an orbital element set by letting $M = E_s - e \sin E_s$ and $n = \sqrt{\frac{\mu}{a^3}}$, where μ is the gravitational constant of the earth, and by setting $\frac{\dot{n}}{2}$ and $\frac{\ddot{n}}{6}$ to zero.

Defining Δn to be $(n_s - n_t) / (N - 1)$ if the predicted impact point is known, and to be $(360^\circ - 2\nu_s) / (N - 1)$ otherwise, the Δt_i are calculated in the same manner as Section 3.3.1, steps 3 to 5.

3.4 RADAR RELATED PROBLEMS

The radar position (λ, ϕ, h) and coverage limits (R, A_m, A_M, E_m, E_M) are stored in the SABERS data base for each radar. The coverage limits are the range R, minimum and maximum azimuth A_m and A_M, and the minimum and maximum elevation, E_m and E_M. As shown in Figure 3-8, azimuth is measured clockwise from local north, and elevation is measured from the horizon, positive above

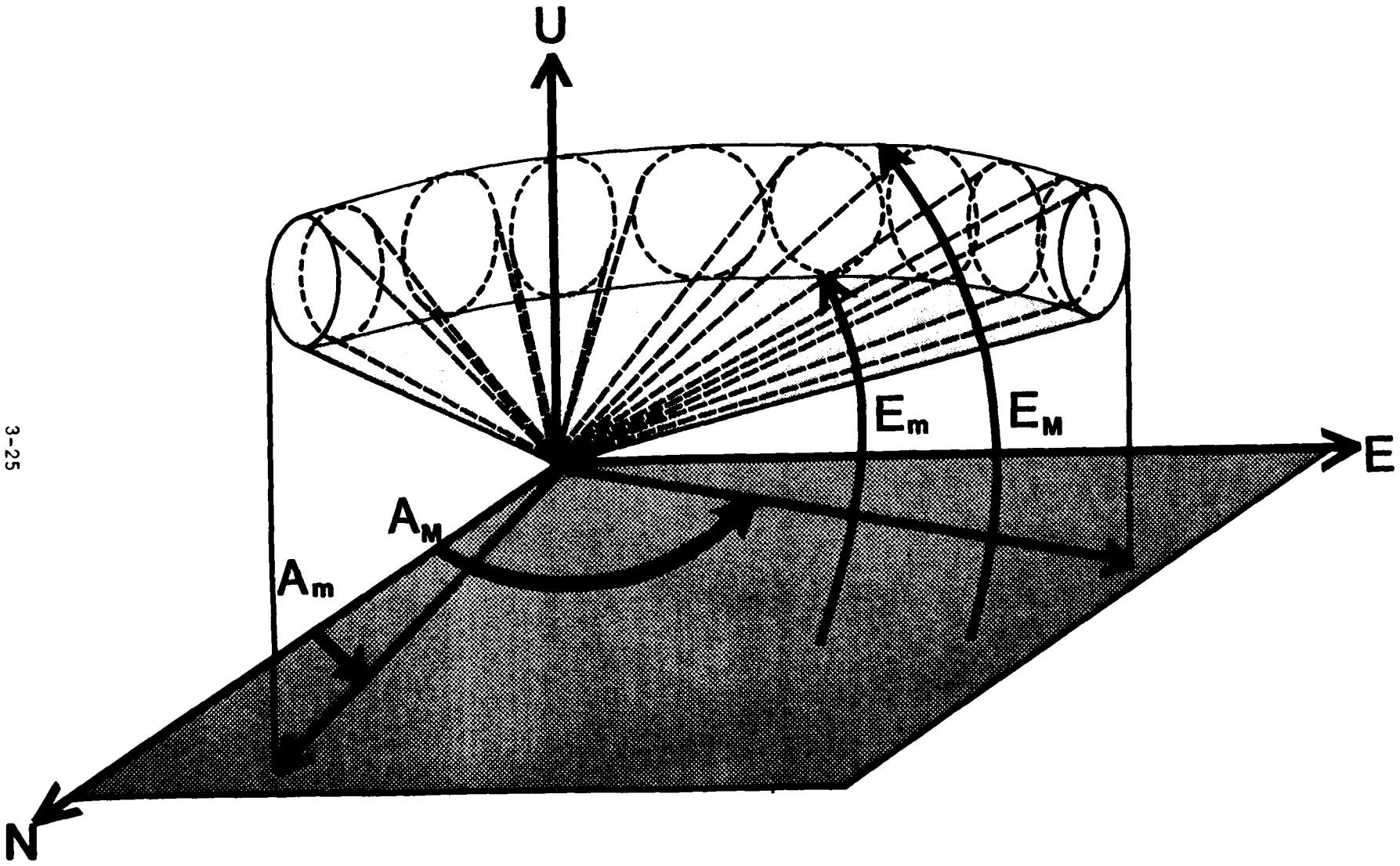


Figure 3-8 Radar Coverage Limits

the horizon and negative below the horizon. The azimuth limits are $0^{\circ} \leq A \leq 360^{\circ}$ and the elevation limits are $E_o \leq E \leq 90^{\circ}$. In practice, E_o is limited to values greater than five or ten degrees due to refraction effects.

3.4.1 Radar Coverage Plots

Elevation information is lost in graphically representing extents, and the range information is distorted. This is due to the method chosen to represent the radar coverages, which is to project points on the radar beam periphery onto the surface of the earth. The set of points to be plotted is projected from ECI coordinates by the transformation of Section 3.2.2.3. The set of ECI points is generated by the following algorithm.

In the local topocentric coordinate system, the unit vector defined by the azimuth is $\begin{pmatrix} e \\ n \\ u \end{pmatrix} = \begin{pmatrix} \sin A \\ \cos A \\ 0 \end{pmatrix}$, for any azimuth A. Using the transformation matrix G described in Section 3.2.3, this unit vector in the ECI coordinate system is $\underset{\rightarrow}{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = G^T \begin{pmatrix} e \\ n \\ u \end{pmatrix}$. The point on the radar beam at range $R + \Delta R$ for any azimuth in ECI coordinates is then $\underset{\rightarrow}{P} = \underset{\rightarrow}{L} + (R + \Delta R)\underset{\rightarrow}{X}$, where $\underset{\rightarrow}{L}$ is the radar site position in ECI coordinates obtained by the transformation of Section 3.2.2.2.

In order to show the effect of the curvature of the earth on the radar coverage picture, only 3/5 of the points to be plotted are calculated at each ΔA with range R. For $A = A_m$ and $A = A_M$, 1/5 of the points to be plotted are calculated by letting ΔR vary from -R to 0.

3.4.2 Radar Coverage of a Satellite

Given a particular geometry, it is straightforward to determine if a satellite is under radar coverage by comparing the actual range, azimuth, and elevation to the satellite with the radar limits. As shown in Figure 3-9, the

azimuth and elevation angles are defined in the local topocentric coordinate system by

$$El = \sin^{-1}(\delta u) \qquad -90^{\circ} \leq El \leq 90^{\circ}$$

$$Az = \tan^{-1}\left(\frac{\delta e}{\delta n}\right) \qquad 0^{\circ} \leq Az \leq 360^{\circ}$$

where $\begin{pmatrix} \delta e \\ \delta n \\ \delta u \end{pmatrix}$ is the direction cosine vector from the radar site to the satellite position.

In ECI coordinates, the range ρ to the satellite from the radar is

$$P = \left| \vec{X}_S - \vec{X}_R \right| = \sqrt{(X_S - X_R)^2 + (Y_S - Y_R)^2 + (Z_S - Z_R)^2}$$

where $\vec{X}_S$ is the satellite position calculated by the method of Section 3.2.4.2 and $\vec{X}_R$ is the radar position calculated by the transformation of Section 3.2.2.2. The direction cosine vector is then $\begin{pmatrix} \delta x \\ \delta y \\ \delta z \end{pmatrix} = \frac{\vec{X}_S - \vec{X}_R}{\rho}$ in ECI coordinates. By the transformation matrix G of Section 3.2.3, we may then calculate the local topocentric direction cosine vector as $\begin{pmatrix} \delta e \\ \delta n \\ \delta u \end{pmatrix} = G \begin{pmatrix} \delta x \\ \delta y \\ \delta z \end{pmatrix}$, and calculate El and Az.

To graphically represent the radar coverage for which the satellite is visible, the method of Section 3.4.1 may be used. The azimuth angles at t_{start} and t_{end} replace the radar limit angles A_m and A_M.

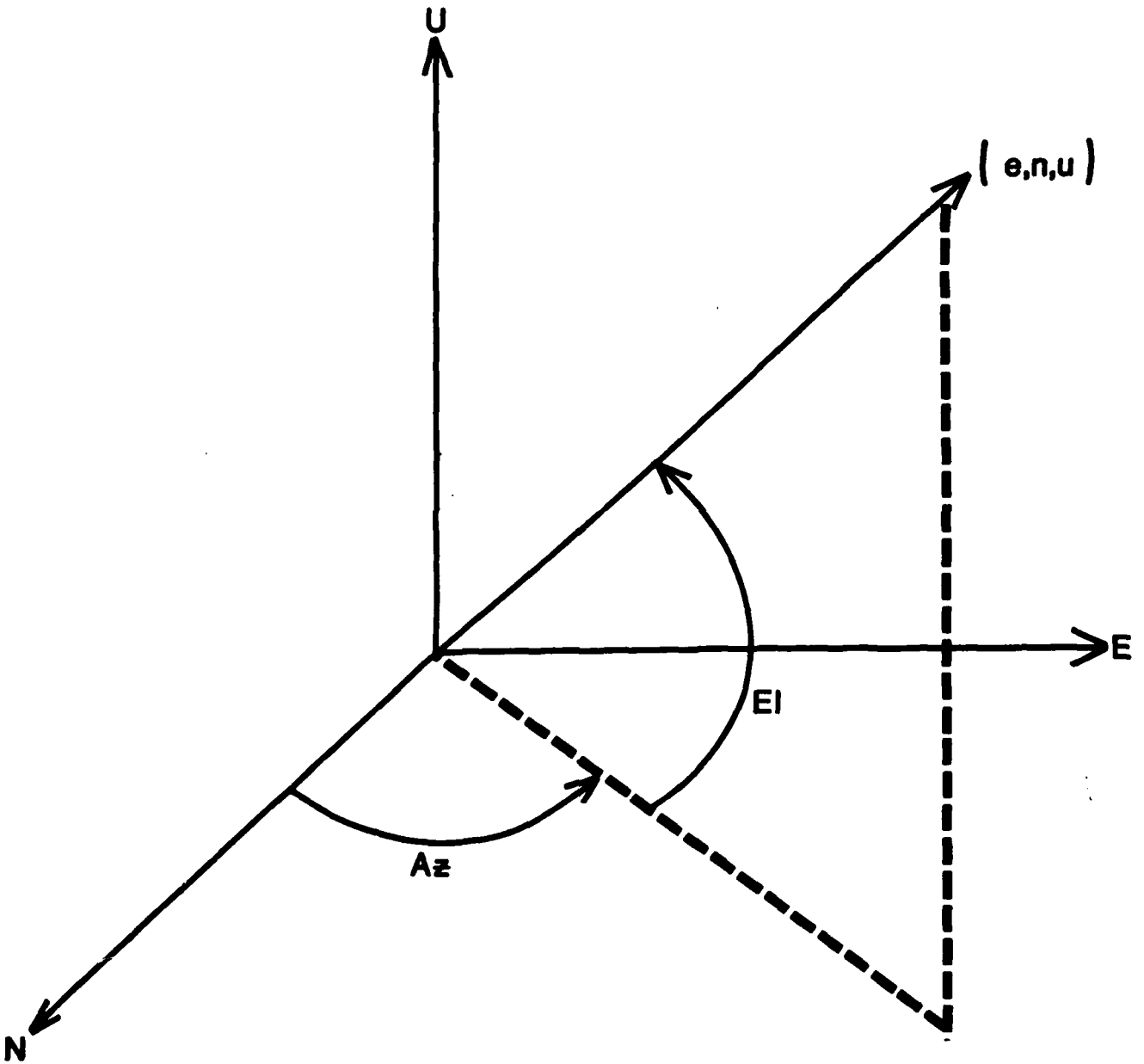


Figure 3-9 Azimuth and Elevation Angle Definition

3.4.2.1 Time of Coverage Calculations

If the satellite is in coverage at times t_i and t_j, and out of coverage at times t_{i-1} and t_{j+1}, it is known that the satellite first becomes visible at some time between t_{i-1} and t_i, and passes out of coverage at some time between t_j and t_{j+1}. These exact times are approximated by t_{i-1} and t_{j+1}, so that the satellite is considered under radar coverage over the larger time interval. Greater accuracy may be gained by using a smaller time interval.

3.4.2.2 Exceptions to Coverage Checks

Checks for coverage need not be performed for those geometries for which the ground trace of the satellite is in the opposite hemisphere from the radar site (see Figure 3-10). No checks need to be performed at all if the satellite's minimum height above the *reference* ellipsoid (that is, at periapsis) is larger than the radar range limit R (see Figure 3-11). This means there is no check if $R < a\,(1 - e) - a_e$, for an orbit with semi-major axis a and eccentricity e and earth semi-major axis a_e .

3.4.2.3 Pass Through Coverage Check

In the discrete method of Section 3.4.2, it is possible for the satellite to be reported as out of coverage, when for some time interval inside the time step size the satellite may have passed through radar coverage (see Figure 3-12). As implied by the terminology "passed through," this condition may be checked for by comparing the relations between the actual values and the limits at the two discrete time points whenever the time step size is much smaller than the period of the satellite, T_p. If the elevation angles measured at time t_i and t_{i+1} are both less than E_m, or both greater than E_M, then there has been no pass through. Similarly, if the two measured azimuth angles are both less than A_m or greater than A_M, there has been no pass through. Finally, if the ranges measured at time t_i and t_{i+1} are both greater than the range limit, and the satellite has not passed through periapsis, then there has been no pass through.

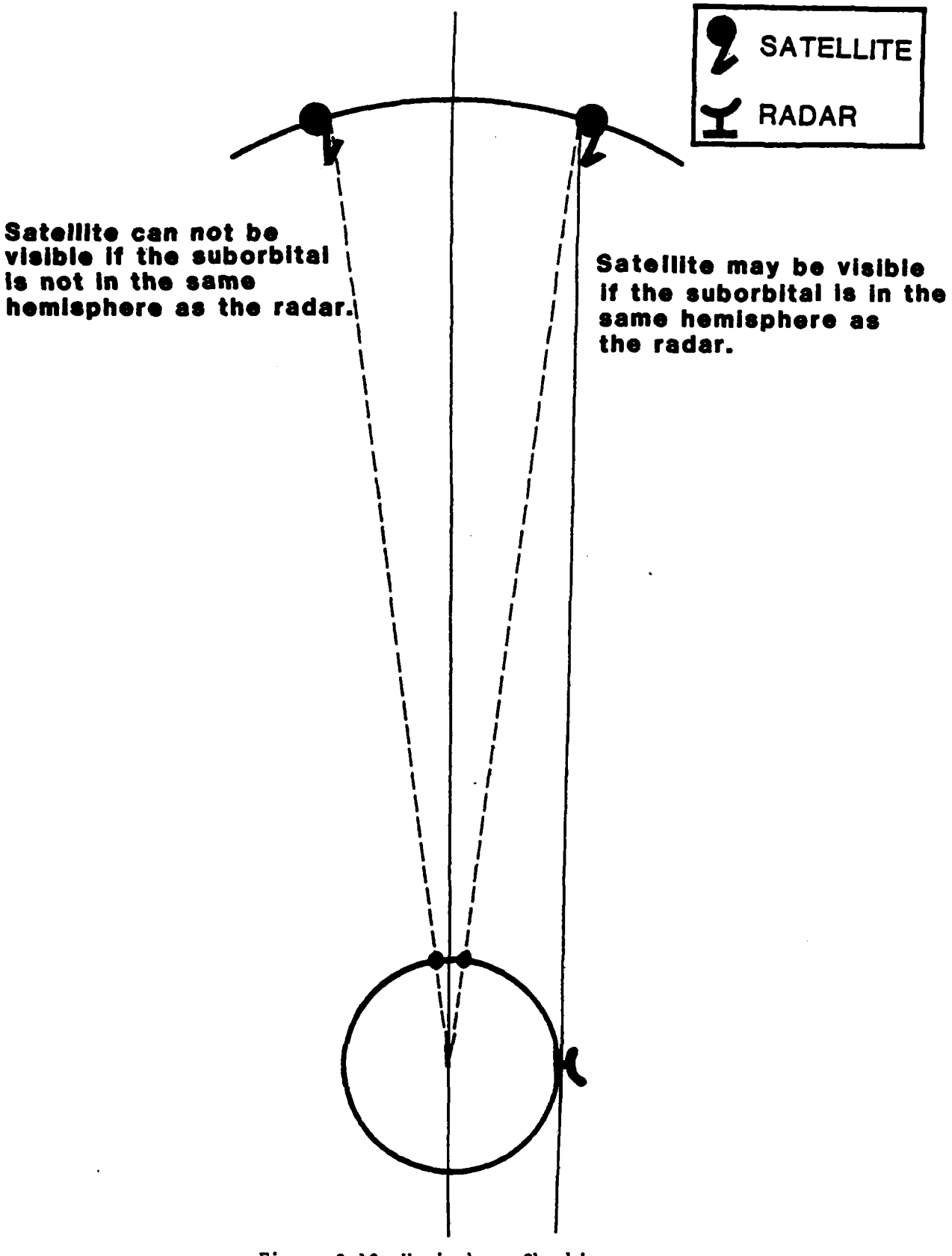
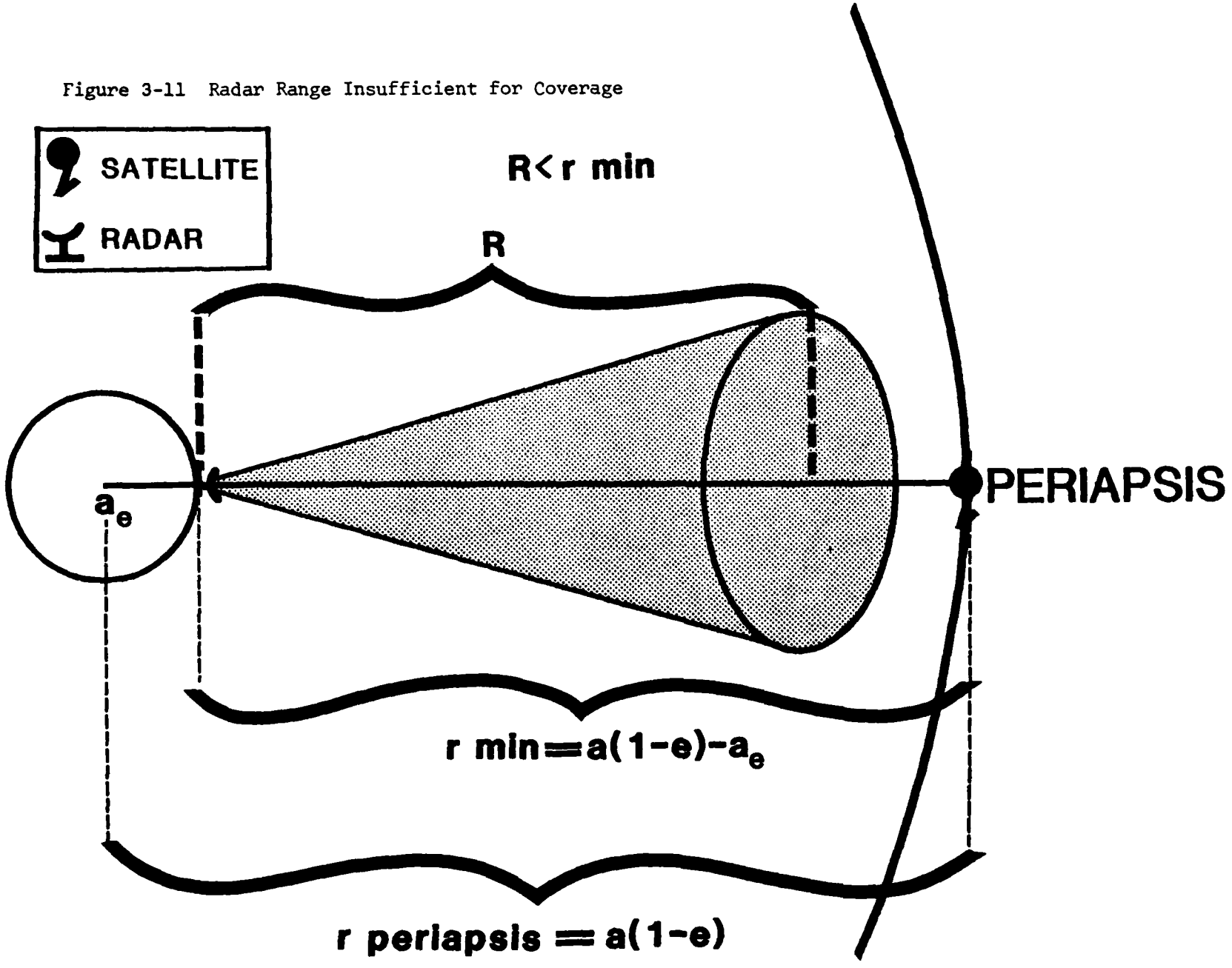


Figure 3-10 Hemisphere Checking

Figure 3-11 Radar Range Insufficient for Coverage



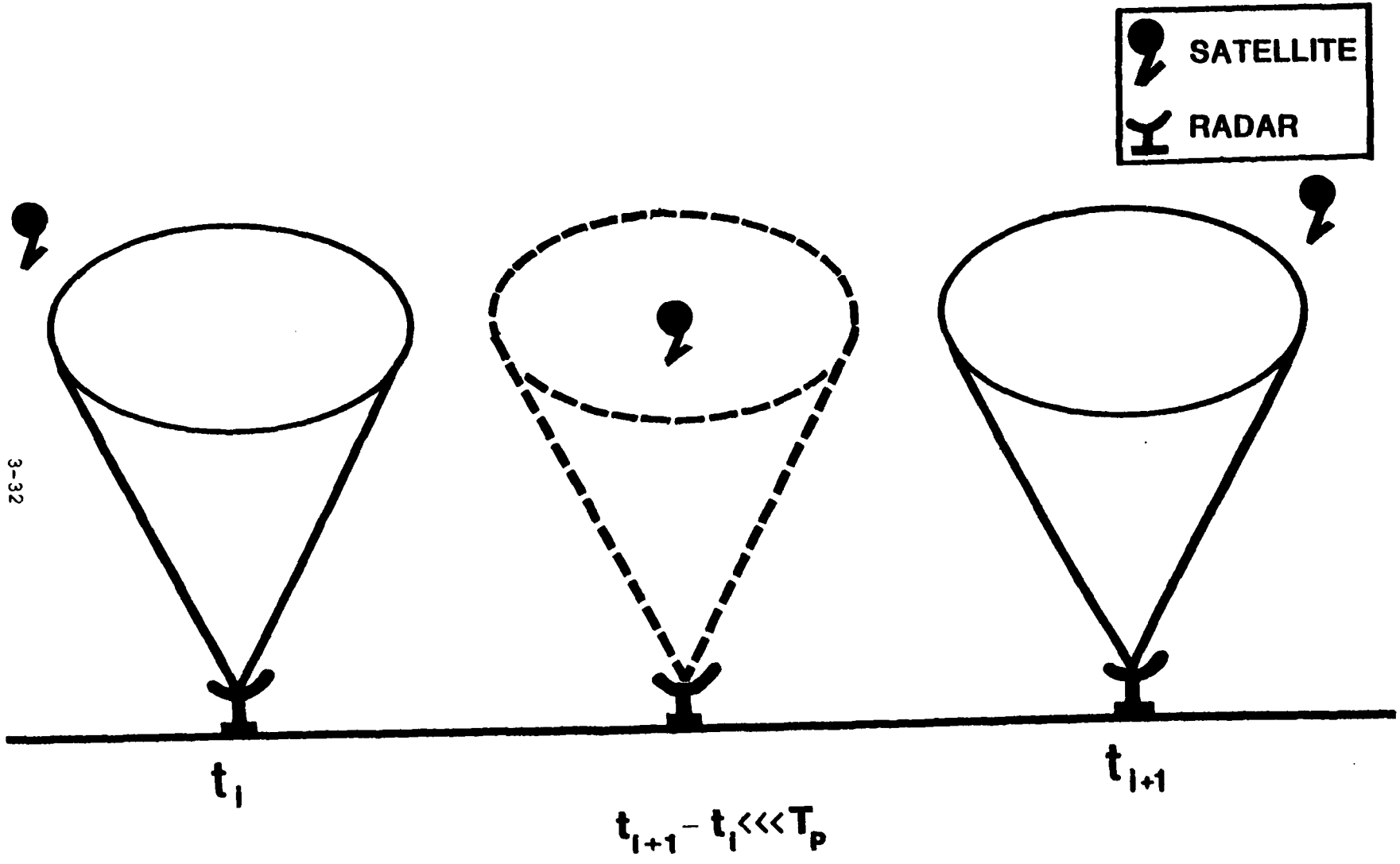


Figure 3-12 Satellite Passing Through Radar Coverage

The satellite may pass through radar coverage when the ground trace crosses from the different hemisphere to the same hemisphere as the radar site. This means that the check values are calculated for the first and last times for which the satellite is in the opposite hemisphere.

3.5 PHOTO RECONNAISSANCE PROBLEM

The satellite in orbit may carry a camera mounted in such a position that a portion of the earth's surface will be under reconnaissance coverage. In SABERS, the camera aperture is defined by the field of view angle Ψ, Figure 3-13. The camera mounting is defined by two angles: the azimuth (from local north), Az, and elevation (from down), El, both measured from the axis of the cone. If the elevation angle is zero, then the cone axis intersects the ground trace point of the satellite. This follows from the definition of the satellite sub-orbital in Section 3.3.1.

3.5.1 Photo Recconaissance Plots

Drawing the coverage of the camera requires the solution of the intersection of the oblique cone with the reference ellipsoid. The algorithm is developed for two cases: El = 0 and El $\neq$ 0.

3.5.1.1 Case 1: El = 0

As described in pages 4-424 to 4-425 in reference [6] the intersection of the ECI vector

$$\vec{X} = \vec{S} + \rho \vec{P}, \qquad (3\text{-}2)$$

where

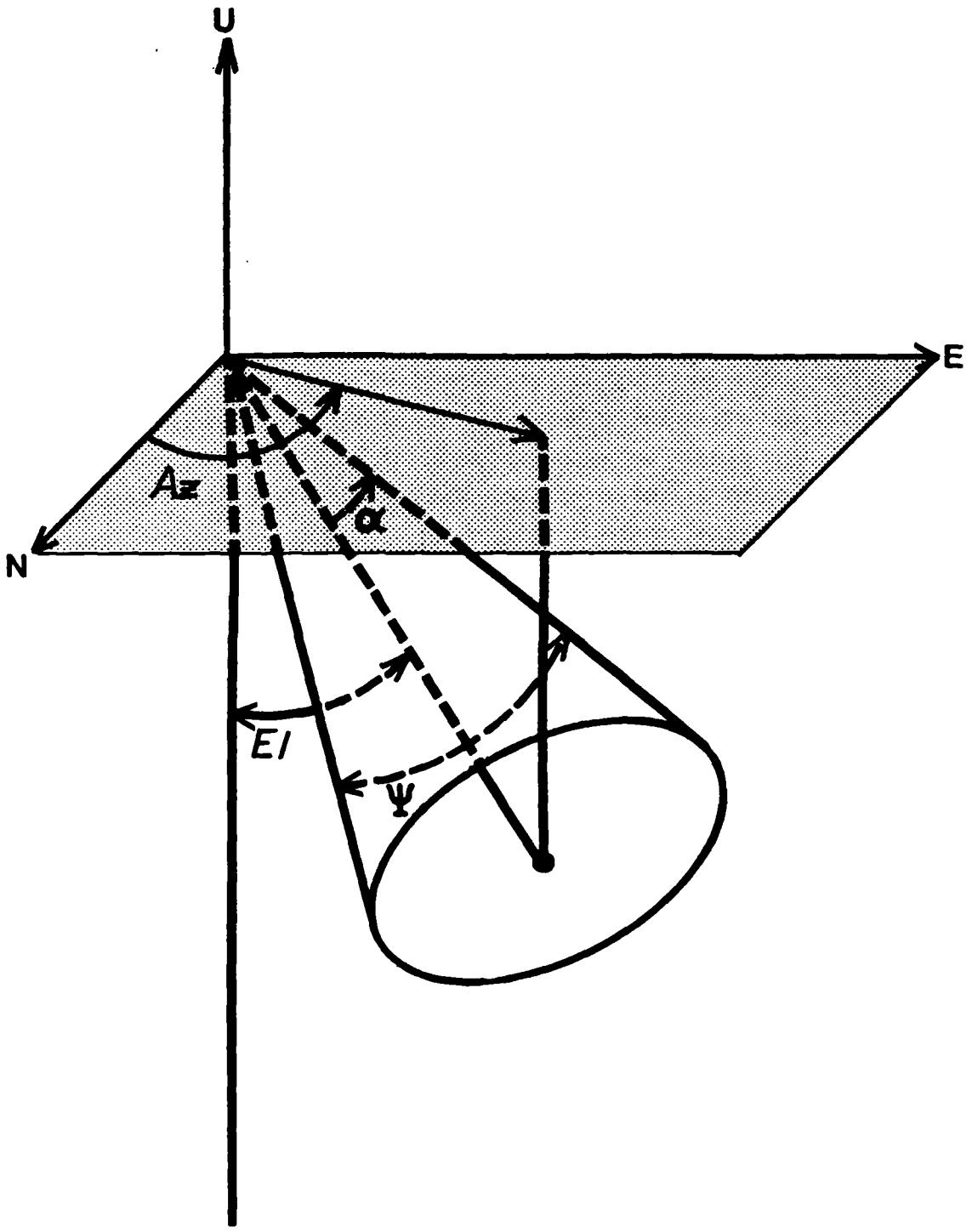


Figure 3-13 Camera Field-of-View and Mounting Angles

$\underset{\rightarrow}{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is the vector to the point of intersection

$\underset{\rightarrow}{S} = \begin{pmatrix} S_1 \\ S_2 \\ S_3 \end{pmatrix}$ is the position vector of the satellite

ρ = slant range

$\underset{\rightarrow}{P} = \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix}$ is the line of sight vector

(see Figure 3-14)

and the reference ellipsoid

$$\frac{x^2 + y^2}{a_e^2} + \frac{z^2}{b_e^2} = 1 \qquad (3\text{-}3)$$

yields the two solutions for the slant range:

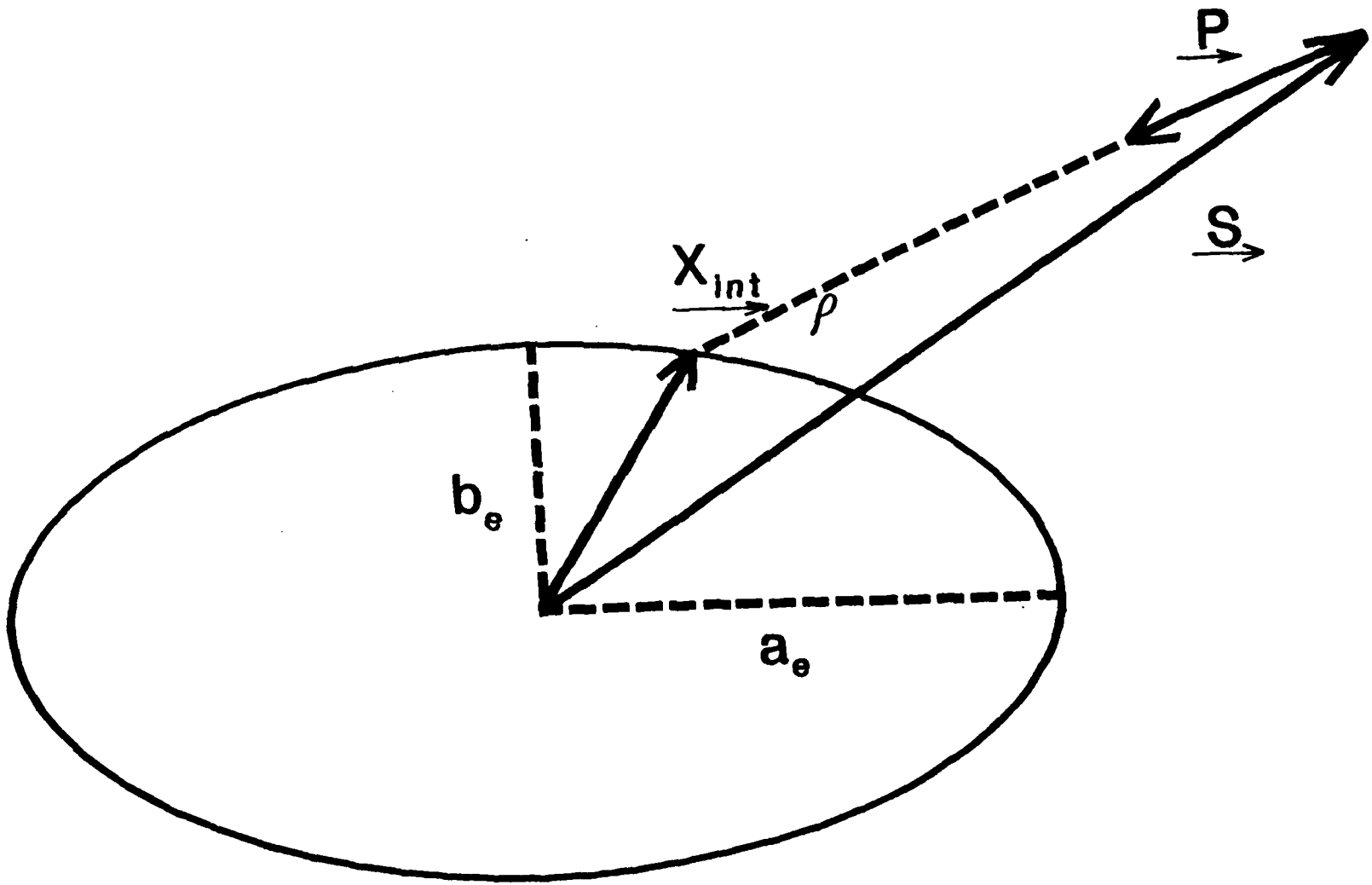


Figure 3-14 Slant Range Calculation

$$\rho = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \qquad (3\text{-}4)$$

where

$$A = \frac{P_1^2 + P_2^2}{a_e^2} + \frac{P_3^2}{b_e^2}$$

$$B = \frac{2\,S_1\,P_1 + 2\,S_2\,P_2}{a_e^2} + \frac{2\,S_3\,P_3}{b_e^2}$$

$$C = \frac{S_1^2 + S_2^2}{a_e^2} + \frac{S_3^2}{b_e^2} - 1$$

where a_e is the semi-major axis of the earth and b_e is the semi-minor axis of the earth. The minimum of the two roots is used to ensure that the solution $\vec{X}$ is the point in the same hemisphere as the satellite ground trace.

At time t, the satellite position vector $\vec{S}$ is known by the method of Section 3.2.4.2. As mentioned in Section 3.5, the cone axis is coincident with the ground trace vector. In the local topocentric coordinate system, the cone axis direction cosine vector is then $\begin{pmatrix} e \\ n \\ u \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$. The ECI cone axis direction cosine vector $\hat{R}$ is then

$$\hat{R} = G^T \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix},$$

where G is the transformation matrix defined in Section 3.3. Given any unit vector $\hat{U}$ normal to $\hat{R}$, the line of sight vector $\vec{P}$ on the surface of the cone is then defined by

$$\vec{P} = \hat{R} \cos \alpha + \hat{U} \sin \alpha$$

where $\alpha = \frac{\Psi}{2}$ is the angle of the cone measured from the cone axis (see Figure 3-15).

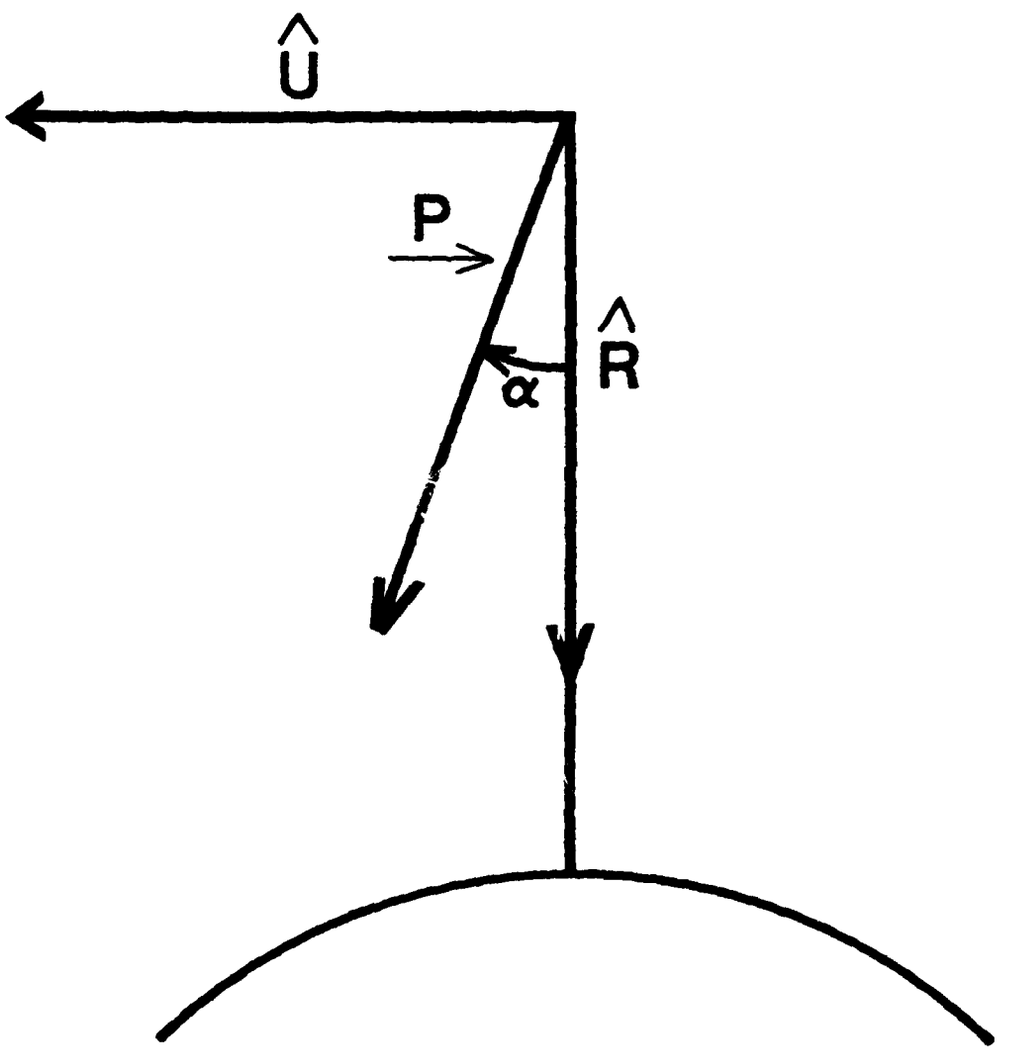
Many such vectors $\hat{U}$ may be generated by considering the local topocentric vectors $\begin{pmatrix} e \\ n \\ u \end{pmatrix} = \begin{pmatrix} \sin A_i \\ \cos A_i \\ 0 \end{pmatrix}$ centered at the satellite position where the A_i constitute a set of azimuth angles from $A_0 = 0^\circ$ to $A_N = 360^\circ$, measured from north. The set of ECI vectors normal to R is then

$$\hat{U}_i = G^T \begin{pmatrix} \sin A_i \\ \cos A_i \\ 0 \end{pmatrix} .$$

Since $\vec{S}$ and $\vec{P}$ are known, the slant range ρ may be solved for by the method of Section 3.5.1.1, upon which $\vec{X}_i$ is known in ECI coordinates. Then $\vec{X}_i$ may be plotted in geocentric coordinates by the transformation of Section 3.2.2.3.

If $\vec{P}$ does not intersect the earth, then $B^2 - 4AC < 0$ and the roots of the quadratic equation (3-4) will be complex. In this case, consider the line of sight vector $\vec{P}$

Figure 3-15 Definition of $\underset{\rightarrow}{P} = \hat{R} \cos \alpha + \hat{U} \sin \alpha$



$$\vec{P} = \hat{R} \cos \beta + \hat{U} \sin \beta$$

such that $\vec{P}$ is tangent to the earth surface. The angle β such that $\vec{P}$ is tangent may be solved for by setting the discriminant $B^2 - 4AC = 0$. Define the earth ellipsoid equation (3-3) as

$$x^2 + y^2 + D z^2 = a_e^2 \text{ , with } D = \frac{a_e^2}{b_e^2}$$

Then by substituting (3-2), we have

$$(S_1 + \rho P_1)^2 + (S_2 + \rho P_2)^2 + D (S_3 + \rho P_3)^2 = a_e^2$$

or

$$\rho^2 (P_1^2 + P_2^2 + D P_3^2) + \rho (2 S_1 P_1 + 2 S_2 P_2 + 2 D S_3 P_3) + S_1^2 + S_2^2 + D S_3^2 - a_e^2 = 0$$

where $\vec{S}$ is constant (the satellite position) and

$$\vec{P} = \hat{R} \cos \beta + \hat{U} \sin \beta$$

Letting

$$A = P_1^2 + P_2^2 + D\,P_3^2 ,$$

$$B = 2\,S_1\,P_1 + 2\,S_2\,P_2 + 2\,D\,S_3\,P_3 ,$$

and

$$C = S_1^2 + S_2^2 + D\,S_3^2 - a_e^2,$$

we want

$$B^2 - 4AC = 0$$

That is, we want by substitution of A, B, and C,

$$4\,(S_1\,P_1 + S_2\,P_2 + D\,S_3\,P_3)^2 - 4\,(P_1^2 + P_2^{\,2} + D\,P_3^2)\,C = 0$$

or

$$S_1^2\,P_1^2 + S_2^2\,P_2^2 + D^2\,S_3^2\,P_3^2 + 2\,S_1\,S_2\,P_1\,P_2 + 2\,D\,S_1\,S_3\,P_1\,P_3 + \qquad (3\text{-}5)$$
$$2\,D\,S_2\,S_3\,P_2\,P_3 - C\,P_1^2 - C\,P_2^2 - C\,D\,P_3^2 = 0$$

Noting that $\vec{P} = \hat{R}\cos\beta + \hat{U}\sin\beta$, we find

$$P_i^2 = r_i^2 \cos^2 \beta + 2 r_i u_i \cos \beta \sin \beta + u_i^2 \sin^2 \beta$$

$$P_i P_j = r_i r_j \cos^2 \beta + (r_i u_j + r_j u_i) \cos \beta \sin \beta + u_i u_j \sin^2 \beta \ .$$

This means that equation (3-5) becomes, by substitution,

$$X \cos^2 \beta + Y \cos \beta \sin \beta + Z \sin^2 \beta = 0 \qquad (3\text{-}6)$$

where

$$X = V_1 r_1^2 + V_2 r_2^2 + V_3 r_3^2 + V_4 r_1 r_2 + V_5 r_1 r_3 + V_6 r_2 r_3$$

$$Y = 2V_1 r_1 u_1 + 2V_2 r_2 u_2 + 2V_3 r_3 u_3 + V_4(r_1 u_2 + r_2 u_1) + V_5(r_1 u_3 + r_3 u_1) + V_6(r_2 u_3 + r_3 u_2)$$

$$Z = V_1 u_1^2 + V_2 u_2^2 + V_3 u_3^2 + V_4 u_1 u_2 + V_5 u_1 u_3 - V_6 u_2 u_3$$

$$V_1 = S_1^2 - C$$
$$V_2 = S_2^2 - C$$
$$V_3 = D\,(D\,S_3^2 - C)$$
$$V_4 = 2\,S_1\,S_2$$
$$V_5 = 2\,D\,S_1\,S_3$$
$$V_6 = 2\,D\,S_2\,S_3$$

Dividing (3-6) by $\cos^2 \beta$, we have a quadratic in $\tan \beta$

$$Z \tan^2 \beta + Y \tan \beta + X = 0$$

or

$$\tan \beta = \frac{-Y \pm \sqrt{Y^2 - 4XZ}}{2Z}$$

No solution for β exists if $Z = 0$ or $Y^2 < 4XZ$. This will occur if the $\hat{R} \times \hat{U}$ plane does not intersect the earth.

Choosing the larger of the two roots to get positive β, find

$$\cos \beta = (1 + \tan^2 \beta)^{-1}$$

$$\sin \beta = \tan \beta \cos \beta$$

This gives

$$\underset{\rightarrow T}{P} = \hat{R} \cos \beta + \hat{U} \sin \beta$$

where

$\underset{\rightarrow T}{P} = \begin{pmatrix} P_{T_1} \\ P_{T_2} \\ P_{T_3} \end{pmatrix}$ is the line of sight to the point of tangency

Therefore,

$$A = P_{T_1}^2 + P_{T_2}^2 + D\, P_{T_3}^2$$

$$B = 2\, S_1\, P_{T_1} + 2\, S_2\, P_{T_2} + 2\, D\, S_3\, P_{T_3}$$

and

$$\rho_T = -\frac{B}{2A}$$

The resultant $\underset{\rightarrow T}{X}$ may be plotted in geocentric coordinates by the transformation of Section 3.2.2.3

3.5.1.2 Case II, El $\neq$ 0

Let $\bar{R}'$ be the vector along the cone axis. In the local topocentric coordinate system,

$$\bar{R}' = \begin{pmatrix} e \\ n \\ u \end{pmatrix} = \begin{pmatrix} \sin El \sin Az \\ \sin El \cos Az \\ -\cos El \end{pmatrix}$$

Note that if El = 0^o, $\bar{R}' = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$ as before.

It is desirable to define two mutually perpendicular vectors perpendicular to $\bar{R}'$ to form a new coordinate system with $\bar{R}'$ as one axis. Define:

$$E' = \frac{\bar{R}' \times N}{|\bar{R}' \times N|} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ r_1 & r_2 & r_3 \\ 0 & 1 & 0 \end{vmatrix} = \frac{-r_3 \vec{i} + r_1 \vec{k}}{\sqrt{r_3^2 + r_1^2}}$$

and

$$N' = \frac{E' \times \bar{R}'}{|E' \times \bar{R}'|} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ \frac{-r_3}{\sqrt{r_3^2+r_1^2}} + n_1^2 & 0 & \frac{r_1}{\sqrt{r_3^2+r_1^2}} + n_1^2 \\ r_1 & r_2 & r_3 \end{vmatrix}$$

$$= \frac{-r_1 r_2 \vec{i} + (r_1^2 + r_3^2) \vec{j} - r_2 r_3 \vec{k}}{\sqrt{r_1^2 r_2^2 + (r_1^2 + r_3^2)^2 + r_2^2 r_3^2}}$$

noting that if E1 = 0, then $E' = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = E$ and $N' = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = N$. Now define $\bar{U}'_i = N' \cos \theta_i + E' \sin \theta_i$ for $0^\circ \leq \theta_i \leq 360^\circ$, to provide a rotating system of cone surface rays similar to before.

As before, the vectors may be expressed in ECI coordinates as $\bar{R} = G^T \bar{R}'$ and $\bar{U} = G^T \bar{U}'$, for the transformation matrix G defined in Section 3.2.3, and we may calculate $\bar{P} = \bar{R} \cos \alpha + \bar{U} \sin \alpha$ for the line of sight vector in ECI coordinates.

At this point, it is convenient to find the $\hat{U}$ vector such that $\bar{P}$ is contained in the plane defined by $\hat{R} = G^T \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$ and $\hat{U}$. This is simply

$$\hat{U} = \frac{\bar{P} - \hat{R} \cos \xi}{\sin \xi}$$

where ξ is the angle between $\hat{R}$ and $\bar{P}$ defined by

$$\cos \xi = \frac{\hat{R} \cdot \bar{P}}{|\hat{R}| \; |\bar{P}|}$$

The point $\bar{X}$ may then be found by the method of El = 0, where the field of view angle α is replaced by ξ.

3.5.2 Special Conditions

There are special conditions to be considered when the cone axis does not intersect the earth. There are no computational problems when the cone axis does intersect the earth.

3.5.2.1 Determining If the Cone Intersects the Earth

As shown in Figure 3-16, the cone may not intersect the earth or may envelop the earth. Since the algorithm described in Section 3.5.1.2 will return all the tangents, only the case of no intersection must be detected independently.

The cone will not intersect the earth if the angle between the cone axis and the vector to the center of the earth, θ, is larger than the field of view angle, α, and the angle to the tangent vector, β, from the ray to the center of the earth, i.e. $\theta > \alpha + \beta$.

Let $\bar{R}$ be the cone axis vector in ECI coordinates, and $\vec{S}$ be the satellite position derived by the method of Section 3.2.4.2. Let $\vec{P}_T$ be the vector

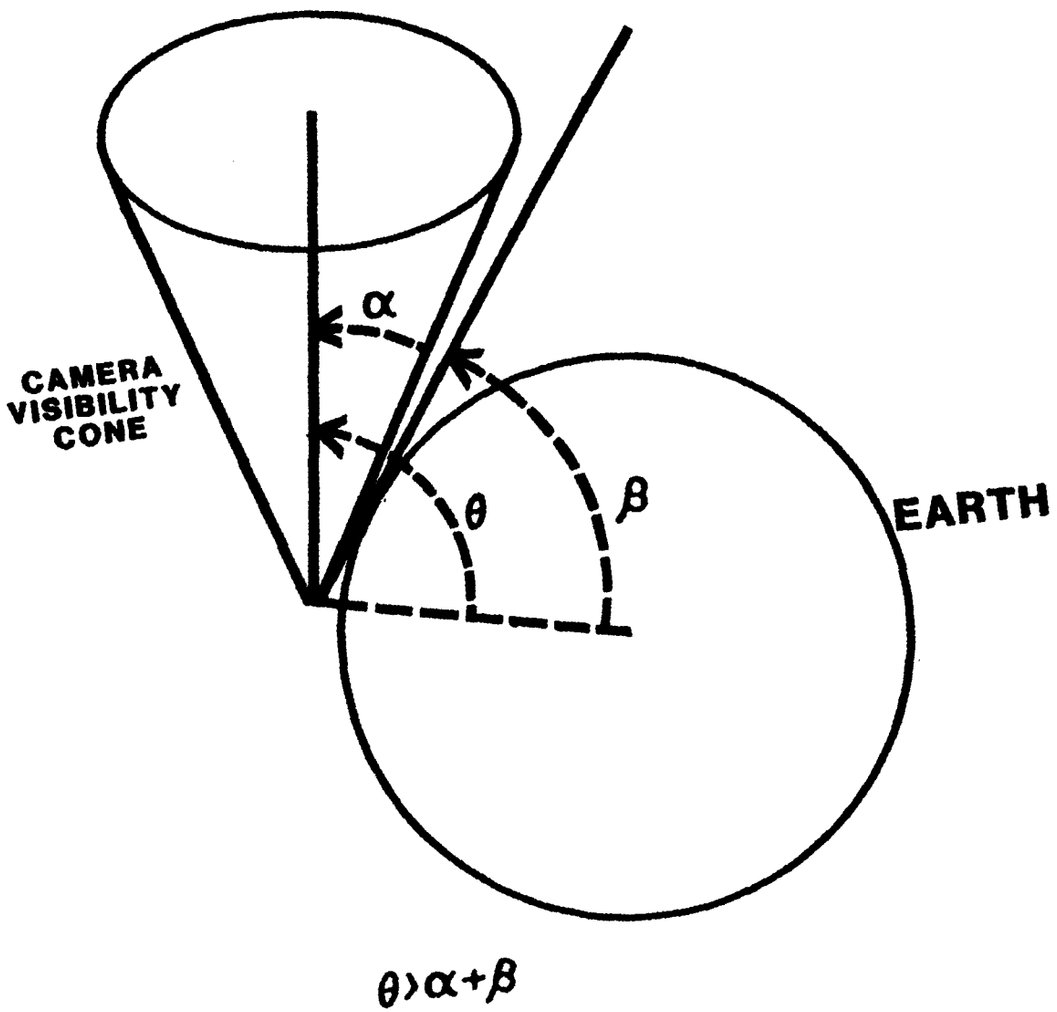


Figure 3-16 Camera Cone Does Not Intersect the Earth

tangent to the earth's surface.

Then

$$\cos \beta = \frac{\vec{P}_T \cdot -\vec{S}}{\left|\vec{P}_T\right| \left|\vec{S}\right|}$$

and

$$\sin \beta = \sqrt{1 - \cos^2 \beta}$$

also,

$$\cos \theta = \frac{\bar{R} \cdot -\vec{S}}{\left|\bar{R}\right| \left|\vec{S}\right|}$$

There is no intersection if $\theta > \alpha + \beta$, that is, if $\cos \theta < \cos \alpha \cos \beta - \sin \alpha \sin \beta$.

3.5.2.2 Removing Excess Points

As shown in Figure 3-17, some of the points of tangency should not be plotted in case the point of tangency is not in the cone coverage. This is detected by noting that the angle, θ, between the tangent ray $\bar{P}$ and the cone axis $\bar{R}$ is larger than the field of view angle α. That is:

$$\theta > \alpha$$

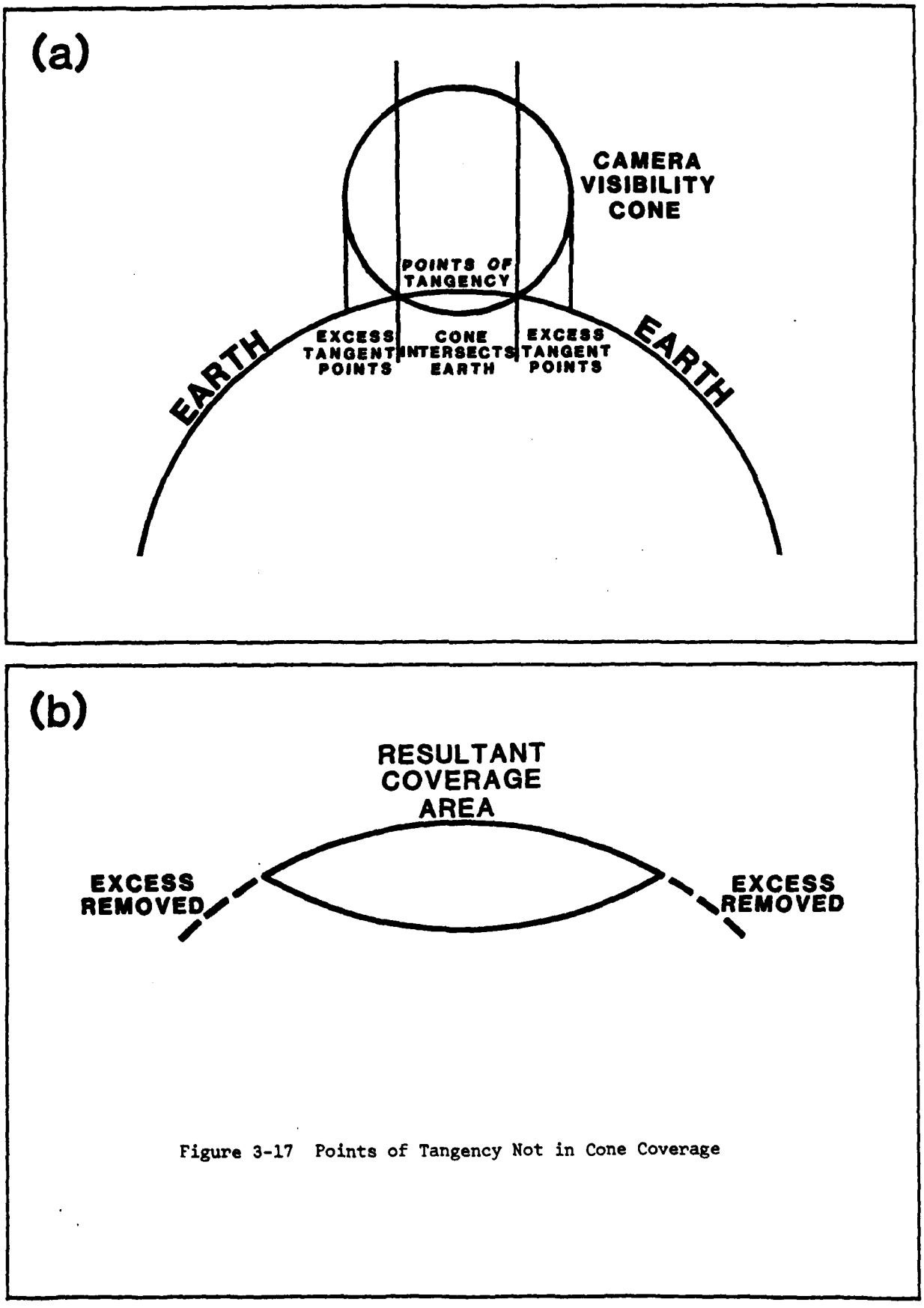


Figure 3-17 Points of Tangency Not in Cone Coverage

$$\cos \theta < \cos \alpha$$

or

$$\cos \alpha > \frac{\bar{P} \cdot \bar{R}}{|\bar{P}||\bar{R}|}$$

3.5.3 Time of Coverage Calculations

Approximations are performed in calculating the times of coverage of a point of interest on the ground by the reconnaissance satellite. These approximations result in the algorithm producing times of coverage that span the actual times of coverage. At each time step, the points of intersection may be determined by the method of Section 3.5.2. The figure determined by these points of intersection is approximated by a circle on the surface of the earth circumscribing the figure. The center of the circle is the average of all the points in the figure, and the radius of the circle is taken as the largest of the great circle distances from the center point to the figure vertices. The great circle distance formula between two points (λ_1, ϕ_1, h_1) and (λ_2, ϕ_2, h_2) is

$$r = a_e \cos^{-1} (\sin \phi_1 \sin \phi_2 + \cos \phi_1 \cos \phi_2 \cos (\lambda_1 - \lambda_2)),$$

where a_e is the semi-major axis of the earth.

3.5.3.1 Pass Over Coverage Check

To check against coverage between the discrete time steps, the following method is used. Let the two figure center points be $CP_1 = (\lambda_1, \phi_1, h_1)$ and $CP_2 = (\lambda_2, \phi_2, h_2)$. Calculate the two radii of coverage r_1 and r_2 with the

great circle distance formula. Let $r = \max(r_1, r_2)$. It is necessary to compare r with the minimum distance from the center point trace to the ground point of interest, $P = (\lambda_p, \phi_p, h_p)$.

From spherical trigonometry, (See Figure 3-18),

$$\cos a = \sin \phi_1 \sin \phi_2 + \cos \phi_1 \cos \phi_2 \cos (\lambda_1 - \lambda_2)$$

$$\cos b = \sin \phi_1 \sin \phi_p + \cos \phi_1 \cos \phi_p \cos (\lambda_1 - \lambda_p)$$

$$\cos c = \sin \phi_2 \sin \phi_p + \cos \phi_2 \cos \phi_p \cos (\lambda_2 - \lambda_p)$$

and

$$\cos c = \cos a \cos b + \sin a \sin b \cos \beta \qquad (3\text{-}7)$$

where β is the included angle between sides a and b. It is desirable to find $0 \leq \gamma \leq 1$ such that

$$c = \cos^{-1} (\cos (\gamma a) \cos b + \sin (\gamma a) \sin \beta \cos \beta)$$

is minimum.

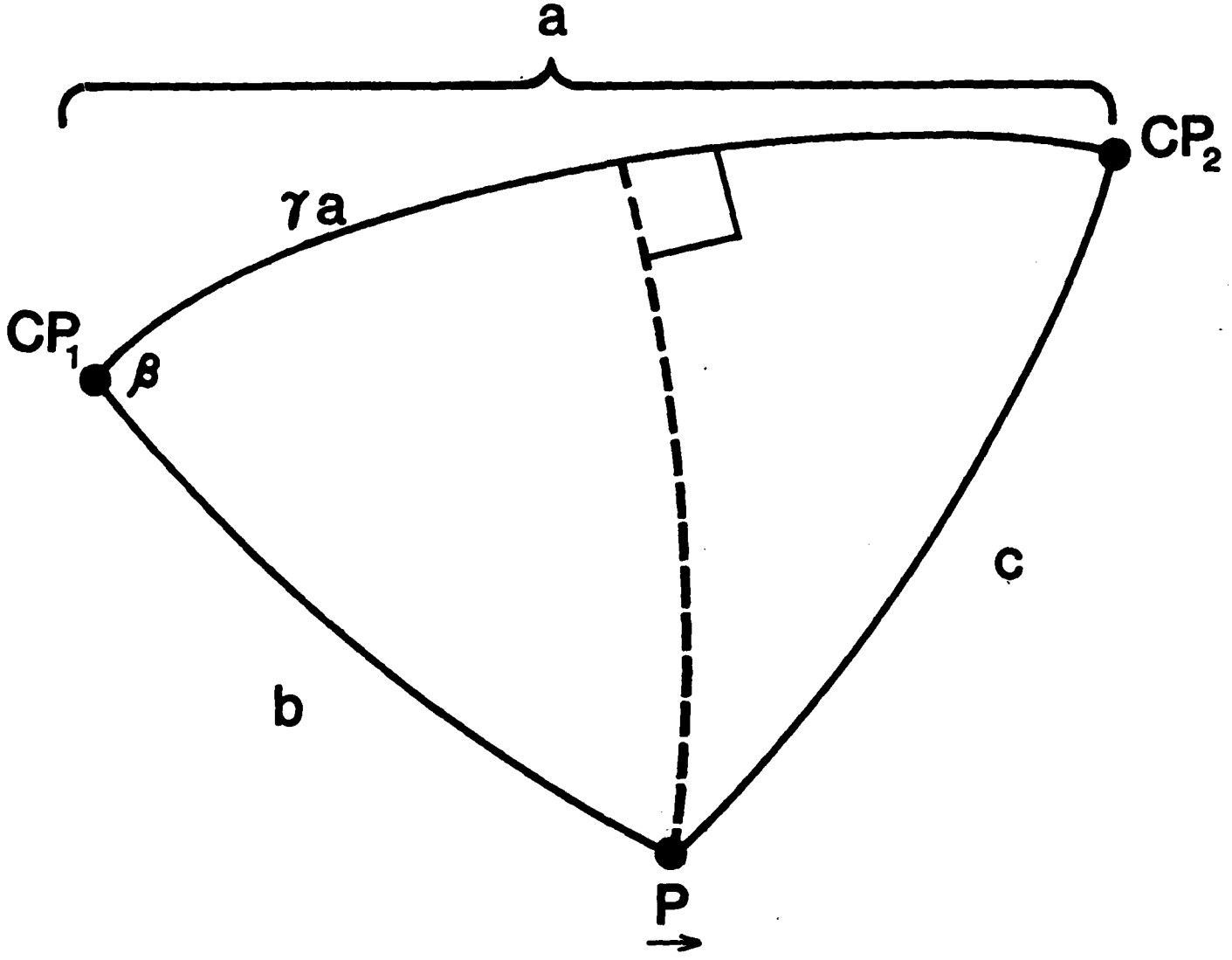


Figure 3-18 Reconnaissance Coverage

Let $\frac{dc}{d\gamma} = \frac{d \cos^{-1}(u)}{d\gamma} (-\sin (\gamma a) \cos b + \cos (\gamma a) \sin b \cos \beta) = 0.$

Since

$$\frac{d \cos^{-1}(u)}{d \gamma} = -(1 - u^2)^{1/2} \neq 0,$$

we have

$$-\sin (\gamma a) \cos b + \cos (\gamma a) \sin b \cos \beta = 0$$

$$\sin (\gamma a) \cos b = \cos (\gamma a) \sin b \cos \beta$$

$$\tan (\gamma a) = \tan b \cos \beta$$

$$\gamma = \frac{\tan^{-1} (\tan b \cos \beta)}{a}$$

Let $a' = \gamma a$ for $0 \leq \gamma \leq 1$, and then find

$$d = b_e \cos^{-1} (\cos a' \cos b + \sin a' \sin b \cos \beta)$$

with b_e = earth semi-major axis. It is not necessary to find β explicitly, since

$$\sin b \cos \beta = \frac{\cos c - \cos a \cos b}{\sin a}$$

from (3-7). Also, if a = sin a = 0, let $d = b_e \cos^{-1} (\max (\cos b, \cos c))$. If $d \leq r$, then the ground point is considered to be under coverage by the reconnaissance satellite.

3.5.3.2 Coverage Time Interval Accuracy

The set of all times such that $d < r$ for time pairs (t_{i-1}, t_i), (t_j, t_{j+1}) is reported as start, end times (t_{i-1}, t_{j+1}). This is similar to the method described in Section 3.4.2.1. However, there is a limit to the accuracy of this algorithm, due to approximating non-circular figures with circles. To ensure that the error is on the side of caution, the d calculated in Section 3.5.3.1 is multiplied by $\frac{a_e}{b_e} \doteq 1.0034$ as a model correction factor.

3.6 THREAT WINDOW ALGORITHM

The determination of launch windows (times at which a payload may be launched from a site to intercept a target satellite in its orbit) is based on LPRE, a launch prediction algorithm described on pages 2-11 to 2-17 in reference [7].

3.6.1 Existing LPRE Formulation

The empirical formulas provided in the reference are functions of the target orbital period, T_p, and the phase angle ϕ. The phase angle is defined as the angle at the center of the earth between the target satellite position and the launch site position at the time, t_p, when the site is coplanar with the orbit.

Letting t_o = window open time, t_c = window close time and t_n = nominal launch time, the following regions have been identified:

Region 1: $0^\circ < \phi < 60^\circ$

$$t_o = t_p - .05\ T_p$$

$$t_c = t_p + \frac{60^\circ - \phi}{360^\circ}\ T_p$$

$$t_n = \begin{cases} t_p + .05\ T_p, & \phi \leq 42^\circ \\ t_p & \phi > 42^\circ \end{cases}$$

Region 2: $-60^\circ < \phi < 0^\circ$

$$t_o = \begin{cases} t_p - .05\ T_p, & -42^\circ \leq \phi \\ t_p - \frac{60^\circ + \phi}{360^\circ}\ T_p, & -60^\circ \leq \phi < -42^\circ \end{cases}$$

$$t_c = t_p + \frac{\phi + 180^\circ}{360^\circ}\ T_p$$

$$t_n = t_p + .05\ T_p$$

Region 3: $-180^\circ < \phi < -60^\circ$

$$t_o = t_p + \frac{\phi + 60^\circ}{360^\circ}\ T_p$$

$$t_c = t_p + \frac{\phi + 180^\circ}{360^\circ}\ T_p$$

$$t_n = \frac{t_o + t_c}{2}$$

Region 4: $60^{o} < \phi < 180^{o}$

not viable

3.6.2 Determining Time of Coplanarity

Finding t_p, the time of coplanarity, is accomplished in an iterative fashion. There will be a maximum of two times of coplanarity in one day, and a minimum of no times of coplanarity (in case the launch site latitude is larger than the orbit plane inclination). Therefore, there are at most two launch windows for a given launch site and orbit plane in one day.

For launch site (λ, ϕ, h) and satellite position $\vec{S}$ and velocity $\vec{V}$, at current time, t, in seconds since midnight, define the ECI coordinates of the launch site as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} G_1 \cos\phi \cos\theta \\ G_1 \cos\phi \sin\theta \\ G_2 \sin\phi \end{pmatrix} \qquad (3\text{-}8)$$

where θ is the hour angle of section 3.2.2.1 at t,

$$G_1 = h + \frac{a_e}{\sqrt{1 - (2f - f^2)\sin^2\phi}}$$

$$G_2 = h + \frac{a_e(1 - f)^2}{\sqrt{1 - (2f - f^2)\sin^2\phi}}$$

a_e = 6378.16 km. = semi-major axis of the earth

$$f = \frac{1}{298.25}$$

(see page 115 of reference [3]), and denote the orbital plane as

$$A'X + B'Y + C'Z + D' = 0 \qquad (3\text{-}9)$$

when

$$(A', B', C') = \frac{\vec{S} \times \vec{V}}{\left|\vec{S} \times \vec{V}\right|} \quad \text{and} \quad D' = 0$$

Substituting (3-8) into (3-9), we have

$$A \cos\theta + B \sin\theta + C = 0 \qquad (3\text{-}10)$$

for

$$A = A' G_1 \cos\theta, \quad B = B' G_1 \cos\phi, \text{ and } C = C' G_2 \sin\phi.$$

Solve for the true roots of (3-10) from among the four solutions to

$$\theta = \cos^{-1} \frac{-AC \pm B\sqrt{A^2 + B^2 - C^2}}{A^2 + B^2}$$

$$\theta = \sin^{-1} \frac{-BC \pm A\sqrt{A^2 + B^2 - C^2}}{A^2 + B^2}$$

and for each true root, find t_i (time since midnight) as

$$t_i = \frac{\theta - \lambda - \theta_{G_o}}{\frac{d\theta}{dt}}$$

where

θ_{G_o} = Greenwich hour angle at midnight

and

$\frac{d\theta}{dt}$ = earth's rate of rotation

Using t_i as the current time, adjusted values for $\underset{\rightarrow}{S}$ and $\underset{\rightarrow}{V}$ may be used to refine the estimate of t_i until

$$\left| t_i^k - t_i^{k+1} \right| < \varepsilon$$

Then the phase angles ϕ_i for each t_p are calculated as

$$\phi_i = \cos^{-1}\left(\frac{\underset{\rightarrow i}{S} \cdot \underset{\rightarrow i}{L}}{\left|\underset{\rightarrow i}{S}\right| \left|\underset{\rightarrow i}{L}\right|}\right)$$

where $\underset{\rightarrow i}{S}$ and $\underset{\rightarrow i}{L}$ are the ECI positions of the satellite and launch site at time t_i.

3.7 MAP PROJECTIONS

In this section, the formulae required to perform the SABERS map drawing functions will be presented. These formulae permit the translation of a point

or series of points on the earth's surface, represented by latitude and longitude, to X and Y values for plotting on a Cartesian coordinate system.

There are two basic steps in this conversion. First, a scale factor must be calculated. This value is based on the size of the map display or on a specified map scale. Once the scale factor has been determined, any number of points may be translated for display.

The equations below assume a standard Cartesian coordinate output is required; that is, the origin of the display surface will be in the center, and X and Y values may be positive or negative. The SABERS display assumes a coordinate system with the origin at the lower left, and only positive values of X and Y are acceptable. Therefore, a simple offset value is subtracted from the equation results to yield the proper SABERS coordinates.

Mathematic symbols and their meanings are shown in Table 3-1. Valid values for latitude are:

$$-90^{o} \leq \rho \leq 90^{o}$$

Valid values for longitude are:

$$-180^{o} \leq \mu \leq 180^{o}$$

The formulae described were taken from reference [9].

3.7.1 Scale Factors

The following equations are used to calculate the scale factors for the projections listed.

Table 3-1 Mathematic Symbols

SYMBOL	DESCRIPTION	UNITS
E_c	The eccentricity of the earth	----
E_R	Angular distance from point to map center point	radians
i	An intermediate calculation value	----
M	Map radius for orthographic projection	inches
R	Radius of the earth	inches
S	Scale of map at true scale latitude	inches/inch
X	Abcissa of translated point	inches
X_r	Range of X values (width of map)	inches
X_s	Scale factor in X direction	inches
Y	Ordinate of translated point	inches
Y_r	Range of Y values (height of map)	inches
Y_s	Scale factor in Y direction	inches
μ	Longitude of point	radians
μ_ϕ	Longitude at map center point	radians
μ_r	Range of longitude (absolute value of eastmost longitude - westmost longitude)	radians
ρ	Latitude of point	radians
ρ_ϕ	Latitude at map center point	radians
ρ_c	Co-latitude at point ($\mu/2 - \rho$)	radians
$\rho_{c\phi}$	Co-latitude of center point ($\mu/2 - \rho_\phi$)	radians
ρ_r	Range of latitude (northmost latitude - southmost latitude)	radians
ρ_t	True scale latitude	radians
ϕ	Azimuth angle from point to center	radians

MERCATOR, MILLER, AND SINUSOIDAL

$$i = \frac{[1.0 - E_c^2 \sin^2 \rho_t]^{1/2}}{\cos \rho_t}$$

$$X_s = R/[iS]$$

$$Y_s = X_s$$

EQUIRECTANGULAR

$$X_s = \frac{X_r}{\mu_r}$$

$$Y_s = \frac{Y_r}{\rho_r}$$

ORTHGRAPHIC

$$X_s = MY_s = X_s$$

3.7.2 Projection Equations

The following equations are used to calculate the projections listed.

MERCATOR

$$X = [\mu - \mu_\phi] \cdot X_s$$

$$i = \ln \tan \left[45^\circ + \frac{|\rho|}{2}\right] + \frac{E_c}{2} \cdot \ln \left|\frac{1 - E_c \sin |\rho|}{1 + E_c \sin |\rho|}\right|$$

i assumes the sign of ρ.

$$Y = i\, Y_s$$

MILLER

$$X = [\mu - \mu_\phi] \cdot X_s$$

$$i = \frac{5}{4} \ln \left[\tan \left(45^\circ + \frac{2|\rho|}{5}\right)\right]$$

i assumes the sign of ρ

$$Y = i \cdot Y_s$$

EQUIRECTANGULAR

$$X = [\mu - \mu_\phi] \cdot X_s$$

$$Y = [\rho - \rho_\phi] \cdot Y_s$$

SINUSOIDAL

$$X = [\mu - \mu_\phi] \cdot \cos \rho \cdot X_s$$

$$Y = \rho \cdot Y_s$$

ORTHOGRAPHIC

$$E_R = \cos^{-1} \left[\cos \rho_{c\phi} \cos \rho_c + \sin \rho_{c\phi} \sin \rho_c \cos \left|(\mu - \mu_\phi)\right|\right]$$

Note: This equation in the CAM documentation is marred by two errors. This is the correct equation.

$$i = \cos^{-1} \left[\frac{\cos \rho_{c\phi} - \cos E_R \cos \rho_c}{\sin E_R \sin \rho_{c\phi}}\right]$$

For $[\mu - \mu_\phi] > \phi$, $\phi = i$
For $[\mu - \mu_\phi] < \phi$, $\phi = 180^\circ - i$

$$X = \sin E_R \cdot \sin \phi \cdot X_s$$

$$Y = \sin E_R \cdot \cos \phi \cdot Y_s$$

BIBLIOGRAPHY

[1] Bates, Roger R., Donald D. Mueller, and Jerry E. White, Fundamentals of Astrodynamics, Dover, 1971.

[2] Battin, Richard H., Astronautical Guidance, McGraw-Hill, 1964.

[3] Escobal, Pedro Ramon, Methods of Orbit Determination, John Wiley and Sons, 1965.

[4] Fliegal, Henry F. and Thomas C. Van Flanden, "A Machine Algorithm for Processing Calendar Dates," Communications of the ACM, Volume II, Number 10, October, 1968.

[5] Herrich, Samuel, Astrodynamics, vol. 1, Van Nostrand Reinhold Co., 1971.

[6] Prisling, R. H., W. F. Rearich and D. C. Walker, MPP Mission Planning Program, The Aerospace Corporation Report, SAMSO-TR-74-188, June 15, 1974.

[7] Sanders, Dr. Jon, Threat Window Study PAR Proposal for Contract RFP #F30602-79-R-0006, January 23, 1979.

[8] Sonalkar, Dr. Ranjan, Roger L. Dygert, and Dr. Probal Sanyal, Real Time Adaptive System for the Coelostat Optical Tracking Mount, PAR Report 76-32 for contract #F30602-75-C-0144, November 1976.

[9] CAM Cartographic Automatic Mapping Program Documentation 5th Edition, GC 77-10126, June 1977.

FLECS:

USER'S MANUAL

University of Oregon Edition

This manual corresponds to version 22 of Flecs.
(Revised October 1, 1975)

Author: Terry Beyer

Address: Computing Center
University of Oregon
Eugene, Oregon 97403

Telephone: (503) 686-4416

Published by: Department of Computer Science
University of Oregon

Acknowledgements

The author is indebted to many people for assistance of one form or another during the course of this project. Mike Dunlap, Kevin McCoy, and Peter Moulton deserve special thanks for many helpful and fruitful discussions, suggestions, and encouragements. I am grateful to my wife, Kathleen, who assisted in many ways including shielding me from the harsh reality of JCL and 360 Assembly Language. Text preparation was adroitly accomplished by Marva VanNatta, Allyene Tom, Diane Lane, and Kathleen Beyer.

This project was initiated while the author was working under a grant provided by the Office of Scientific and Scholarly Research of the Graduate School at the University of Oregon. Work on the project has also been supported in part by the Department of Computer Science and by the Computing Center of the University of Oregon.

TABLE OF CONTENTS

1.0 INTRODUCTION

Fortran contains four basic mechanisms for controlling program flow: CALL/RETURN, IF, DO, and various forms of the GO TO.

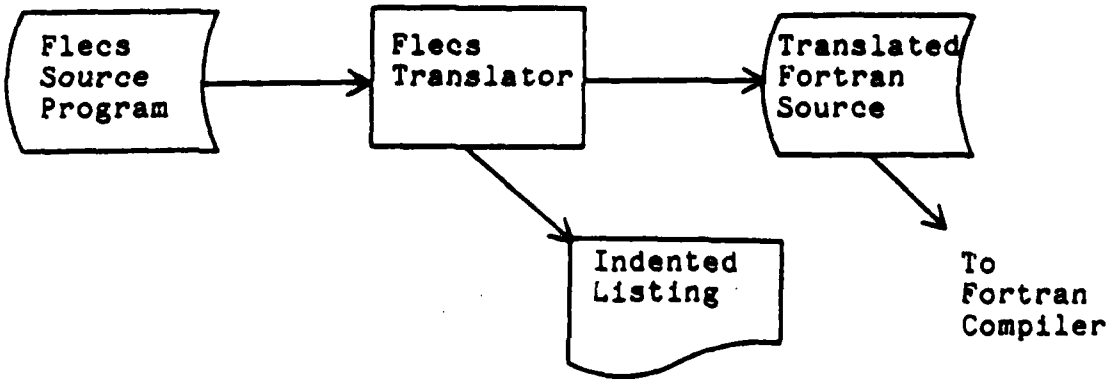
Flecs is a language extension of Fortran which has additional control mechanisms. These mechanisms make it easier to write Fortran by eliminating much of the clerical detail associated with constructing Fortran programs. Flecs is also easier to read and comprehend than Fortran.

This manual is intended to be a brief but complete introduction to Flecs. It is not intended to be a primer on Flecs or structured programming. The reader is assumed to be a knowledgeable Fortran programmer.

For programmers to whom transportability of their programs is a concern, it should be noted that the Flecs translator source code is in the public domain and is made freely available. The translator was written with transportability in mind and requires little effort to move from one machine to another. Those interested in moving Flecs to another machine or in having their own copy of Flecs should contact the author.

At Oregon, Flecs is implemented on both the PDP-10 and the IBM S/360. The manner of implementation is that of a preprocessor which translates Flecs programs into Fortran programs. The resulting Fortran program is then processed in the usual way. The translator also produces a nicely formatted listing of the Flecs program which graphically presents the control structures used.

The following diagram illustrates the translating process.



2.0 RETENTION OF FORTRAN FEATURES

The Flecs translator examines each statement in the Flecs program to see if it is an extended statement (a statement valid in Flecs but not in Fortran). If it is recognized as an extended statement, the translator generates the corresponding Fortran statements. If, however, the statement is not recognized as an extended statement, the translator assumes it must be a Fortran statement and passes it through unaltered. Thus the Flecs system does not restrict the use of Fortran statements, it simply provides a set of additional statements which may be used. In particular, GO TOs, arithmetic IFs, CALLs, arithmetic statement functions, and any other Fortran statements, compiler dependent or otherwise, may be used in a Flecs program.

3.0 CORRELATION OF FLECS AND FORTRAN SOURCES

One difficulty of preprocessor systems like Flecs is that error messages which come from the Fortran compiler must be related back to the original Flecs source program. This difficulty is reduced by allowing the placement of line numbers (not to be confused with Fortran statement numbers) on Flecs source statements. These line numbers then appear on the listing and in the Fortran source. When an error message is produced by either the Flecs translator or the Fortran compiler, it will include the line number of the offending Flecs source statement, making it easy to locate on the listing.

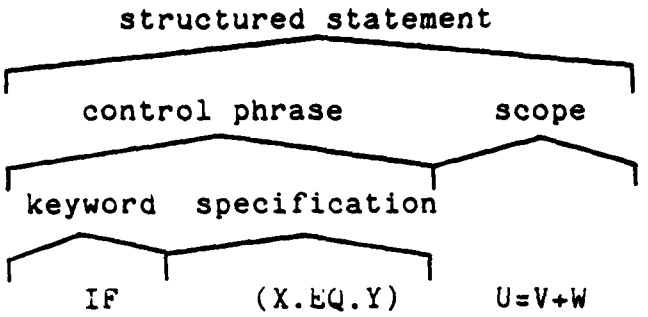
If the programmer chooses not to supply line numbers, the translator will assign sequential numbers and place them on the listing and in the Fortran source. Thus, errors from the compiler may still be related to the Flecs listing.

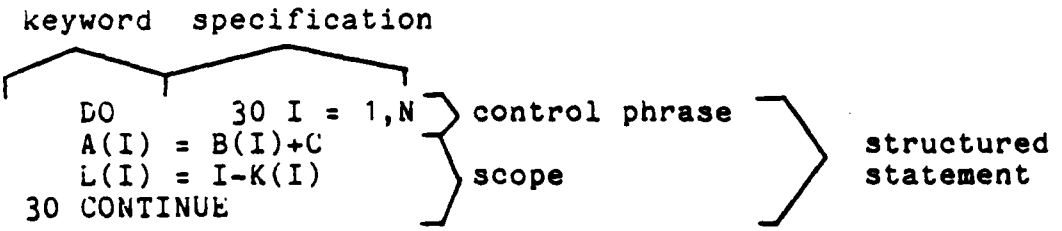
Details of line numbering are machine dependent and are given in chapter 10. On most card oriented systems, the line numbers may be placed in columns 76-80 of each card. Other systems may have special provisions for line numbers.

The beginning Flecs programmer should discover and make special note of the details of the mechanism by which Fortran compiler error messages may be traced back to the Flecs listing on the system being used.

4.0 STRUCTURED STATEMENTS

A basic notion of Flecs is that of the structured statement which consists of a control phrase and its scope. Fortran has two structured statements, the logical IF and the DO. The following examples illustrate this terminology:





Note that each structured statement consists of a control phrase which controls the execution of a set of one or more statements called its scope. Also note that each control phrase consists of a keyword plus some additional information called the specification. A statement which does not consist of a control phrase and a scope is said to be a simple statement. Examples of simple statements are assignment statements, subroutine CALLs, arithmetic IFs, and GO TOs.

The problem with the Fortran logical IF statement is that its scope may contain only a single simple statement. This restriction is eliminated in the case of the DO, but at the cost of clerical detail (having to stop thinking about the problem while a statement number is invented). Note also that the IF specification is enclosed in parentheses while the DO specification is not.

In Flecs there is a uniform convention for writing control phrases and indicating their scopes. To write a structured statement, the keyword is placed on a line beginning in column 7 followed by its specification enclosed in parentheses. The remainder of the line is left blank. The statements comprising the scope are placed on successive lines. The end of the scope is indicated by a FIN statement. This creates a multi-line structured statement.

Examples of multi-line structured statements:

```
IF (X.EQ.Y)
|  U = V+W
|  R = S+T
└─ FIN

DO (I = 1,N)
|  A(I) = B(I)+C
|  C = C*2.14-3.14
└─ FIN
```

Note: The statement number has been eliminated from the DO specification since it is no longer necessary, the end of the loop being specified by the FIN.

Nesting of structured statements is permitted to any depth.

Example of nested structured statements:

```
IF (X.EQ.Y)
|  U = V+W
|  DO (I = 1,N)
|  |  A(I) = B(I)+C
|  |  C = C*2.14-3.14
|  └─ FIN
|  R = S+T
└─ FIN
```

When the scope of a control phrase consists of a single simple statement, it may be placed on the same line as the control phrase and the FIN may be dispensed with. This creates a one-line structured statement.

Examples of one-line structured statements:

```
IF (X.EQ.Y)  U = V+W

DO (I = 1,N)  A(I) = B(I)+C
```

Since each control phrase must begin on a new line, it is not possible to have a one-line structured statement whose scope consists of a structured statement.

Example of invalid construction:

```
IF (X.EQ.Y)  DO (I = 1,N)  A(I) = B(I)+C
```

To achieve the effect desired above, the IF must be written in a multi-line form.

Example of valid construction:

```
IF (X.EQ.Y)
|  DO (I = 1,N) A(I) = B(I)+C
L  FIN
```

In addition to IF and DO, Flecs provides several useful structured statements not available in Fortran. After a brief excursion into the subject of indentation, we will present these additional structures.

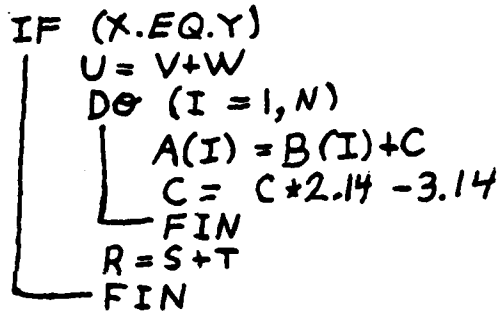
5.0 INDENTATION, LINES AND THE LISTING

In the examples of multi-line structured statements above, the statements in the scope were indented and an "L" shaped line was drawn connecting the keyword of the control phrase to the matching FIN. The resulting graphic effect helps to reveal the structure of the program. The rules for using indentation and FINs are quite simple and uniform. The control phrase of a multi-line structured statement always causes indentation of the statements that follow. Nothing else causes indentation. A level of indentation (i.e. a scope) is always terminated with a FIN. Nothing else terminates a level of indentation.

When writing a Flecs program on paper the programmer should adopt the indentation and line drawing conventions shown below. When preparing a Flecs source program in machine readable form, however, each statement should begin in column 7. When the Flecs translator produces the listing, it will reintroduce the correct indentation and produce the corresponding lines. If the programmer attempts to introduce his own indentation with the use of leading blanks, the program will be translated correctly, but the resulting listing will be improperly indented.

Example of indentation:

1. Program as written on paper by programmer.



2. Program as entered into computer.

```
IF (X.EQ.Y)
U = V+W
DO (I = 1,N)
A(I) = B(I)+C
C = C*2.14-3.14
FIN
R = S+T
FIN
```

3. Program as listed by Flecs translator.

```
IF (X.EQ.Y)
.  U = V+W
.  DO (I = 1,N)
.  .  A(I) = B(I)+C
.  .  C = C*2.14-3.14
.  ...FIN
.  R = S+T
...FIN
```

The correctly indented listing is a tremendous aid in reading and working with programs. Except for the dots and spaces used for indentation, the lines are listed exactly as they appear in the source program. That is, the internal spacing of columns 7-72 is preserved. There is seldom any need to refer to a straight listing of the unindented source.

Comment lines are treated in the following way on the listing to prevent interruption of the dotted lines indicating scope. A comment line which contains only blanks in columns 2 through 6 will be listed with columns 7 through 72 indented at the then-current level of indentation as if the line were an executable statement. If, however, one or more non-blank characters appear in columns 2 through 6 of a comment card, it will be listed without indentation. Blank lines may be inserted in the source and will be treated as empty comments.

6.0 CONTROL STRUCTURES

The complete set of control structures provided by Flecs is given below together with their corresponding flow charts. The symbol $\mathcal{L}$ is used to indicate a logical expression. The symbol $\mathcal{S}$ is used to indicate a scope of one or more statements. Some statements, as indicated below, do not have a one-line construction.

A convenient summary of the information in this chapter may be found in Appendix A.

6.1 Decision Structures

Decision structures are structured statements which control the execution of their scopes on the basis of a logical expression or test.

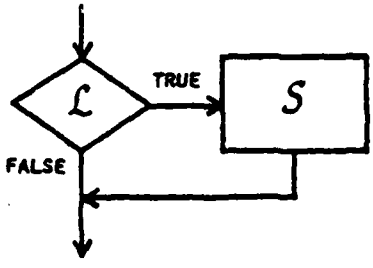
6.1.1 IF

Description: The IF statement causes a logical expression to be evaluated. If the value is true, the scope is executed once and control passes to the next statement. If the value is false, contol passes directly to the next statement without execution of the scope.

General Form:

IF ($\mathcal{L}$) $\mathcal{S}$

Flow Chart:



Examples:

```
IF (X.EQ.Y) U = V+W

IF (T.GT.0.AND.S.LT.R)
.  I = I+1
.  Z = 0.1
...FIN
```

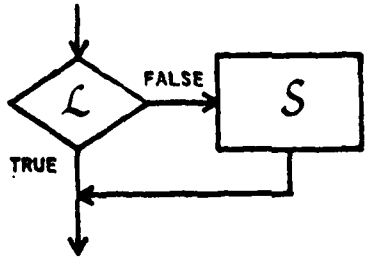
6.1.2 UNLESS

Description: "UNLESS ($\mathcal{L}$)" is functionally equivalent to "IF(.NOT.($\mathcal{L}$))", but is more convenient in some contexts.

General Form:

UNLESS ($\mathcal{L}$) $\mathcal{S}$

Flow Chart:

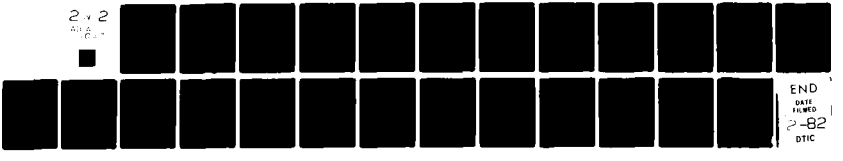


Examples:

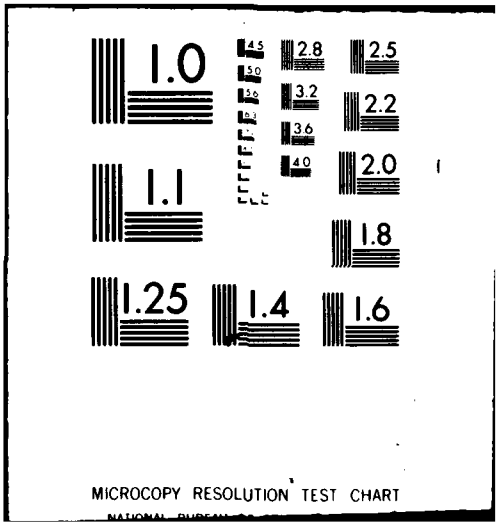
```
UNLESS (X.NE.Y) U = V+W

UNLESS (T.LE.0.OR.S.GE.R)
.  I = I+1
.  Z = 0.1
...FIN
```

AD-A110 271 PAR TECHNOLOGY CORP ROME NY F/G 9/2
SABERS. STAND-ALONE ADIC BINARY EXPLOITATION RESOURCES SYSTEM. --ETC(U)
SEP 81 A J FRANKLIN, R L CALDWELL, S COLE F30602-78-C-0078
UNCLASSIFIED RADC-TR-81-250-VOL-1 NL



END
DATE FILMED
2-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART

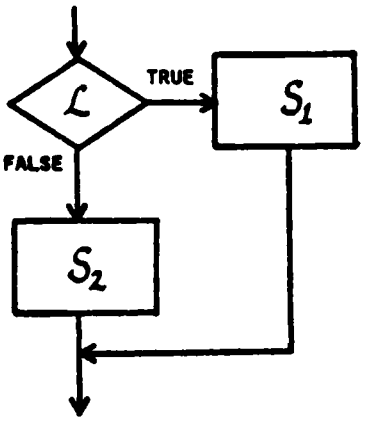
6.1.3 WHEN...ELSE

Description: The WHEN...ELSE statements correspond to the IF...THEN...ELSE statement of Algol, PL/1, Pascal, etc. In Flecs, both the WHEN and the ELSE act as structured statements although only the WHEN has a specification. The ELSE statement must immediately follow the scope of the WHEN. The specifier of the WHEN is evaluated and exactly one of the two scopes is executed. The scope of the WHEN statement is executed if the expression is true and the scope of the ELSE statement is executed if the expression is false. In either case, control then passes to the next statement following the ELSE statement.

General Form:

```
WHEN (ℒ) S1
ELSE S2
```

Flow Chart:



Examples:

```
WHEN (X.EQ.Y) U = V+W
ELSE U = V-W

WHEN (X.EQ.Y)
.  U = V+W
.  T = T+1.5
...FIN
ELSE U = V-W

WHEN (X.EQ.Y) U = V+W
ELSE
.  U = V-W
.  T = T+1.5
...FIN

WHEN (X.EQ.Y)
.  U = V+W
.  T = T-1.5
...FIN
ELSE
.  U = V-W
.  T = T+1.5
...FIN
```

NOTE: WHEN and ELSE always come as a pair of statements, never separately. Either the WHEN or the ELSE or both may assume the multi-line form. ELSE is considered to be a control phrase, hence cannot be placed on the same line as the WHEN. Thus "WHEN (ℒ) S_1 ELSE S_2 " is <u>not</u> valid.

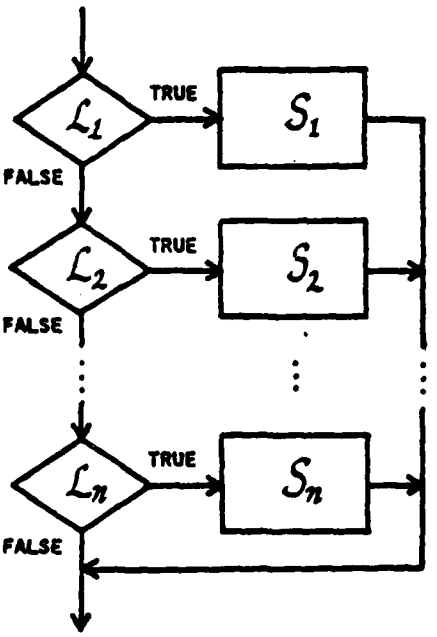
6.1.4 CONDITIONAL

Description: The CONDITIONAL statement is based on the LISP conditional. A list of logical expressions is evaluated one by one until the first expression to be true is encountered. The scope corresponding to that expression is executed, and control then passes to the first statement following the CONDITIONAL. If all expressions are false, no scope is executed. (See, however, the note about OTHERWISE below.)

General Form:

```
CONDITIONAL
. (L1)  S1
. (L2)  S2
.   .    .
.   .    .
.   .    .
. (Ln)  Sn
...FIN
```

Flow Chart:



Examples:

```
CONDITIONAL
.  (X.LT.-5.0)   U = U+W
.  (X.LE.1.0)    U = U+W+Z
.  (X.LE.10.5)   U = U-Z
...FIN

CONDITIONAL
.  (A.EQ.B)    Z = 1.0
.  (A.LE.C)
.  .  Y = 2.0
.  .  Z = 3.4
.  ...FIN
.  (A.GT.C.AND.A.LT.B)    Z = 6.2
.  (OTHERWISE)    Z = 0.0
...FIN
```

Notes: The CONDITIONAL itself does not possess a one-line form. However, each "(L_i) S_i " is treated as a structured statement and may be in one-line or multi-line form.

The reserved word OTHERWISE represents a catchall condition. That is, "(OTHERWISE) S_n " is equivalent to "(.TRUE.) S_n " in a CONDITIONAL statement.

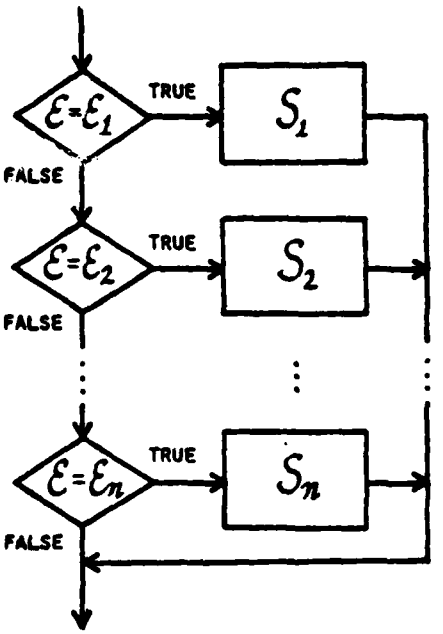
6.1.5 SELECT

Description: The SELECT statement is similar to the CONDITIONAL but is more specialized. It allows an expression to be tested for equality to each expression in a list of expressions. When the first matching expression is encountered, a corresponding scope is executed and the SELECT statement terminates. In the description below, $\mathcal{E}$, $\mathcal{E}_1$,..., $\mathcal{E}_n$ represent arbitrary but compatible expressions. Any type of expression (integer, real, complex,...) is allowed as long as the underlying Fortran system allows such expressions to be compared with an .EQ. or .NE. operator.

General Form:

```
SELECT (ℰ)
. (ℰ1)  S1
. (ℰ2)  S2
.   .   .
.   .   .
.   .   .
. (ℰn)  Sn
...FIN
```

Flow Chart:



Example:

```
SELECT (OPCODE(PC))
.  (JUMP)  PC = AD
.  (ADD)
.  .  A = A+B
.  .  PC = PC+1
.  ...FIN
.  (SKIP)  PC = PC+2
.  (STOP)  CALL STOPCD
...FIN
```

Notes: As in the case of CONDITIONAL, at most one of the S_i will be executed.

The catchall OTHERWISE may also be used in a SELECT statement. Thus "(OTHERWISE) S_n " is equivalent to "($\mathcal{E}$) S_n " within a "SELECT ($\mathcal{E}$)" statement.

The expression $\mathcal{E}$ is reevaluated for each comparison in the list, thus lengthy, time consuming, or irreproducable expressions should be precomputed, assigned to a variable, and the variable used in the specification portion of the SELECT statement.

6.2 LOOP Structures

The structured statements described below all have a scope which is executed a variable number of times depending on specified conditions.

Of the five loops presented, the most useful are the DO, WHILE, and REPEAT UNTIL loops. To avoid confusion, the REPEAT WHILE and UNTIL loops should be ignored initially.

6.2.1 DO

Description: The Flecs DO loop is functionally identical to the Fortran DO loop. The only differences are syntactic. In the Flecs DO loop, the statement number is omitted from the DO statement, the incrementation parameters are enclosed in parenthesis, and the scope is indicated by either the one line or multi-line convention. Since the semantics of the Fortran DO statement vary from one Fortran compiler to another, a flowchart cannot be given. The symbol *I* represents any legal incrementation specification.

General Form:

```
DO (I) S
```

Equivalent Fortran:

```
    DO 30 I
    S
30  CONTINUE
```

Examples:

```
DO (I = 1,N)  A(I) = 0.0

DO (J = 3,K,3)
.  B(J) = B(J-1)*B(J-2)
.  C(J) = SIN(B(J))
...FIN
```

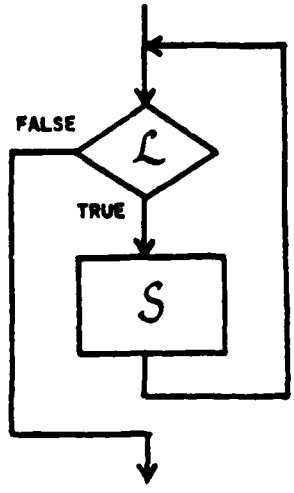
6.2.2 WHILE

Description: The WHILE loop causes its scope to be repeatedly executed while a specified condition is true. The condition is checked prior to the first execution of the scope, thus if the condition is initially false the scope will not be executed at all.

General Form:

WHILE ($\mathcal{L}$) S

Flow Chart:



Examples:

```
WHILE (X.LT.A(I))  I = I+1

WHILE (P.NE.0)
.  VAL(P) = VAL(P)+1
.  P = LINK(P)
...FIN
```

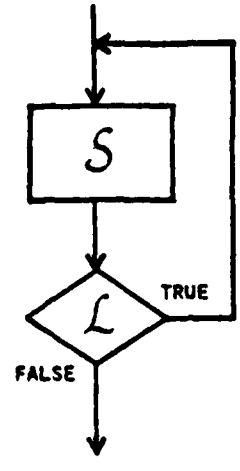
6.2.3 REPEAT WHILE

Description: By using the REPEAT verb, the test can be logically moved to the end of the loop. The REPEAT WHILE loop causes its scope to be repeatedly executed while a specified condition remains true. The condition is not checked until after the first execution of the scope. Thus the scope will always be executed at least once and the condition indicates under what conditions the scope is to be repeated. Note: "REPEAT WHILE($\mathcal{L}$)" is functionally equivalent to "REPEAT UNTIL(.NOT.($\mathcal{L}$))".

General Form:

REPEAT WHILE ($\mathcal{L}$) S

Flow Chart:



Examples:

```
REPEAT WHILE( N.EQ.M(I)) I = I+1

REPEAT WHILE (LINK(Q).NE.0)
.  R = LINK(Q)
.  LINK(Q) = P
.  P = Q
.  Q = R
...FIN
```

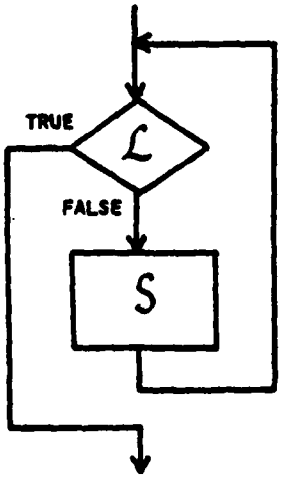
6.2.4 UNTIL

Description: The UNTIL loop causes its scope to be repeatedly executed until a specified condition becomes true. The condition is checked prior to the first execution of the scope, thus if the condition is initially true, the scope will not be executed at all. Note that "UNTIL ($\mathcal{L}$)" is functionally equivalent to "WHILE (.NOT.($\mathcal{L}$))".

General Form:

UNTIL ($\mathcal{L}$) S

Flow Chart:



Examples:

```
UNTIL (X.EQ.A(I))  I = I+1

UNTIL (P.EQ.0)
.  VAL(P) = VAL(P)+1
.  P = LINK(P)
...FIN
```

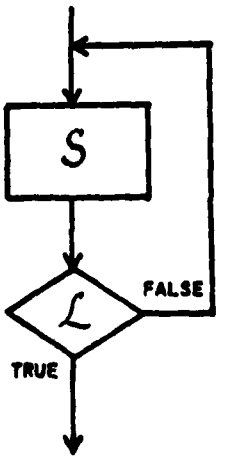
6.2.5. REPEAT UNTIL

Description: By using the REPEAT verb, the test can be logically moved to the end of the loop. The REPEAT UNTIL loop causes its scope to be repeatedly executed until a specified condition becomes true. The condition is not checked until after the first execution of the scope. Thus the scope will always be executed at least once and the condition indicates under what conditions the repetition of the scope is to be terminated.

General Form:

REPEAT UNTIL ($\mathcal{L}$) S

Flow Chart:



Examples:

```
REPEAT UNTIL (N.EQ.M(I)) I = I+1

REPEAT UNTIL (LINK(Q).EQ.0)
.   R = LINK(Q)
.   LINK(Q) = P
.   P = Q
.   Q = R
... FIN
```

7.0 Internal Procedures

In Flecs a sequence of statements may be declared an internal procedure and given a name. The procedure may then be invoked from any point in the program by simply giving its name.

Procedure names may be any string of letters, digits, and hyphens (i.e. minus signs) beginning with a letter and containing at least one hyphen. Internal blanks are not allowed. The only restriction on the length of a name is that it may not be continued onto a second line.

Examples of valid internal procedure names:

```
INITIALIZE-ARRAYS
GIVE-WARNING
SORT-INTO-DESCENDING-ORDER
INITIATE-PHASE-3
```

A procedure declaration consists of the keyword "TO" followed by the procedure name and its scope. The set of statements comprising the procedure is called its scope. If the scope consists of a single simple statement it may be placed on the same line as the "TO" and procedure name, otherwise the statements of the scope are placed on the following lines and terminated with a FIN statement. These rules are analogous with the rules for forming the scope of a structured statement.

General Form of procedure declaration:

```
TO  procedure-name
```

Examples of procedure declarations:

```
TO  RESET-POINTER  P = 0

TO  DO-NOTHING  CONTINUE

TO SUMMARIZE-FILE
.  INITIALIZE-SUMMARY
.  OPEN-FILE
.  REPEAT UNTIL (EOF)
.  . ATTEMPT-TO-READ-RECORD
.  . WHEN (EOF)  CLOSE-FILE
.  . ELSE  UPDATE-SUMMARY
.  ...FIN
.  OUTPUT-SUMMARY
...FIN
```

An internal procedure reference is a procedure name appearing where an executable statement would be expected. In fact an internal procedure reference is an executable simple statement and thus may be used in the scope of a structured statement as in the last example above. When control reaches a procedure reference during execution of a Flecs program, a return address is saved and control is transferred to the first statement in the scope of the procedure. When control reaches the end of the scope, control is transferred back to the statement logically following the procedure reference.

A typical Flecs program or subprogram consists of a sequence of Fortran declarations: (e.g. INTEGER, DIMENSION, COMMON, etc.) followed by a sequence of executable statements called the body of the program followed by the Flecs internal procedure declarations, if any, and finally the END statement.

Here is a complete (but uninteresting) Flecs program which illustrates the placement of the procedure declarations.

```
00010  C  INTERACTIVE PROGRAM FOR PDP-10 TO COMPUTE X**2.
00020  C  ZERO IS USED AS A SENTINEL VALUE TO TERMINATE EXECUTION.
00030
00040          REAL  X,XSQ
00050          REPEAT UNTIL (X.EQ.0)
00060          .  GET-A-VALUE-OF-X
00070          .  IF (X.NE.0)
00080          .  .  COMPUTE-RESULT
00090          .  .  TYPE-RESULT
00100          .  ...FIN
00110          ...FIN
00120          CALL EXIT

    ----------------------------------------

00130          TO GET-A-VALUE-OF-X
00140          .  TYPE 10
00150  10      .  FORMAT (' X = ',$)
00160          .  ACCEPT 20,X
00170  20      .  FORMAT (F)
00180          .. FIN

    ----------------------------------------

00190          TO COMPUTE-RESULT  XSQ = X*X

    ----------------------------------------

00200          TO TYPE-RESULT
00210          .  TYPE 30, XSQ
00220  30      .  FORMAT(' X-SQUARED  =   ,F7.2)
00230          ...FIN
00240          END
```

Notes concerning internal procedures:

1. All internal procedure declarations must be placed at the end of the program just prior to the END statement. The appearance of the first "TO" statement terminates the body of the program. The translator expects to see nothing but procedure declarations from that point on.

2. The order of the declarations is not important. Alphabetical by name is an excellent order for programs with a large number of procedures.

3. Procedure declarations may not be nested. In other words, the scope of a procedure may not contain a procedure declaration. It may of course contain executable procedure references.

4. Any procedure may contain references to any other procedures (excluding itself).

5. Dynamic recursion of procedure referencing is not permitted.

6. All program variables within a main or subprogram are global and are accessable to the statements in all procedures declared within that same main or sub program.

7. There is no formal mechanism for defining or passing parameters to an internal procedure. When parameter passing is needed, the Fortran function or subroutine subprogram mechanism may be used or the programmer may invent his own parameter passing methods using the global nature of variables over internal procedures.

8. The Flecs translator separates procedure declarations on the listing by dashed lines as shown in the preceding example.

8.0 Restrictions and Notes

If Flecs were implemented by a nice intelligent compiler this section would be much shorter. Currently, however, Flecs is implemented by a sturdy but naive translator. Thus the Flecs programmer must observe the following restrictions.

1. Flecs must invent many statement numbers in creating the Fortran program. It does so by beginning with a large number (usually 99999) and generating successively smaller numbers as it needs them. Do not use a number which will be generated by the translator. A good rule of thumb is to avoid using 5 digit statement numbers.

2. The Flecs translator must generate integer variable names. It does so by using names of the form "Innnnn" when nnnnn is a 5 digit number related to a generated statement number. Do not use variables of the form Innnnn and avoid causing them to be declared other than INTEGER. For example the declaration "IMPLICIT REAL (A-Z)" leads to trouble. Try "IMPLICIT REAL (A-H, J-Z)" instead.

3. The translator does not recognize continuation lines in the source file. Thus Fortran statements may be continued since the statement and its continuations will be passed through the translator without alteration. (See chapter 2.) However, an extended Flecs statement which requires translation may not be continued. The reasons one might wish to continue a Flecs statement are 1) It is a structured statement or procedure declaration with a one statement scope too long to fit on a line, or 2) it contains an excessively long specification portion or 3) both of the above. Problem 1) can be avoided by going to the multi-line form. Frequently problem 2) can be avoided when the specification is an expression (logical or otherwise) by assigning the expression to a variable in a preceding statement and then using the variable as the specification.

4. Blanks are meaningful separators in Flecs statements: don't put them in dumb places like the middle of identifiers or key words and do use them to separate distinct words like REPEAT and UNTIL.

5. Let Flecs indent the listing. Start all statements in col. 7 and the listing will always reveal the true structure of the program. (as understood by the translator, of course).

6. As far as the translator is concerned, FORMAT statements are executable Fortran statements since it doesn't recognize them as extended Flecs statements. Thus, only place FORMAT statements where an executable Fortran statement would be acceptable. Don't put them between the end of a WHEN

statement and the beginning of an ELSE statement. Don't put them between procedure declarations.

Incorrect Examples:

```
    WHEN (FLAG)  WRITE(3,30)
30  FORMAT(7H TITLE:)
    ELSE  LINE = LINE+1
```

```
    TO WRITE-HEADER
    .  PAGE = PAGE+1
    .  WRITE(3,40) H,PAGE
    ...FIN
40  FORMAT (70A1,I3)
```

Corrected Examples:

```
    WHEN (FLAG)
    .  WRITE(3,30)
30  .  FORMAT(7H TITLE:)
    ...FIN
    ELSE  LINE = LINE+1
```

```
    TO WRITE-HEADER
    .  PAGE = PAGE+1
    .  WRITE(3,40) H,PAGE
40  .  FORMAT(70A1,I3)
    ...FIN
```

7. The translator, being simple-minded, recognizes extended Flecs statements by the process of scanning the first identifier on the line. If the identifier is one of the Flecs keywords IF, WHEN, UNLESS, FIN, etc., the line is assumed to be a Flecs statement and is treated as such. Thus, the Flecs keywords are reserved and may not be used as variable names. In case of necessity, a variable name, say WHEN, may be slipped past the translator by embedding a blank within it. Thus "WH EN" will look like "WH" followed by "EN" to the translator which is blank sensitive, but like "WHEN" to the compiler which ignores blanks.

8. In scanning a parenthesized specification, the translator scans from left to right to find the parenthesis which matches the initial left parenthesis of the specification. The translator, however, is ignorant of Fortran syntax including the concept of Hollerith constants and will treat Hollerith parenthesis as syntactic parenthesis. Thus, avoid placing Hollerith constants containing unbalanced parenthesis within specifications. If necessary, assign such constants to a variable, using a DATA or assignment statement, and place the variable in the specification.

Incorrect Example:

```
IF (J.EQ.'(')
```

Corrected Example:

```
LP = '('
IF(J.EQ.LP)
```

9. The Flecs translator will not supply the statements necessary to cause appropriate termination of main and sub-programs. Thus it is necessary to include the appropriate RETURN, STOP, or CALL EXIT statement prior to the first internal procedure declaration. Failure to do so will result in control entering the scope of the first procedure after leaving the body of the program. Do not place such statements between the procedure declarations and the END statement.

9.0 Errors

This section provides a framework for understanding the error handling mechanisms of version 22 of the Flecs Translator. The system described below is at an early point in evolution, but has proven to be quite workable.

The Flecs translator examines a Flecs program on a line by line basis. As each line is encountered it is first subjected to a limited syntax analysis followed by a context analysis. Errors may be detected during either of these analysis. It is also possible for errors to go undetected by the translator.

9.1 Syntax Errors

When a syntax error is detected by the translator, it ignores the statement. On the Flecs listing the line number of the statement is overprinted with -'s to indicate that the statement has been ignored. The nature of the syntax error is given in a message on the following line.

The fact that a statement has been ignored may, of course, cause some context errors in later statements. For example the control phrase "WHEN (X(I).LT.(3+4)" has a missing right parenthesis. This statement will be ignored, causing as a minimum the following ELSE to be out of context. The programmer should of course be aware of such effects. More is said about them in the next section.

9.2 Context Errors

If a statement successfully passes the syntax analysis, it is checked to see if it is in the appropriate context within the program. For example an ELSE must appear following a WHEN and nowhere else. If an ELSE does not appear at the appropriate point or if it appears at some other point, then a context error has occurred. A frequent source of context errors in the initial stages of development of a program comes from miscounting the number of FIN's needed at some point in the program.

With the exception of excess FIN's which do not match any preceding control phrase and are ignored (as indicated by overprinting the line number), all context errors are treated with a uniform strategy. When an out-of-context source statement is

encountered, the translator generates a "STATEMENT(S) NEEDED" message. It then invents and processes a sequence of statements which, if they had been included at that point in the program, would have placed the original source statement in a correct context. A message is given for each such statement invented. The original source statement is then processed in the newly created context.

By inventing statements the translator is not trying to patch up the program so that it will run correctly, it is simply trying to adjust the local context so that the original source statement and the statements which follow will be acceptable on a context basis. As in the case of context errors generated by ignoring a syntactically incorrect statement, such an adjustment of context frequently causes further context errors later on. This is called propagation of context errors.

One nice feature of the context adjustment strategy is that context errors cannot propagate past a recognizable procedure declaration. This is because the "TO" declaration is in context only at indentation level 0. Thus to place it in context, the translator must invent enough statements to terminate all open control structures which preceed the "TO". The programmer who modularizes his program into a collection of relatively short internal procedures, limits the potential for propagation of context errors.

9.3 Undetected Errors

The Flecs translator is ignorant of most details of Fortran syntax. Thus most Fortran syntax errors will be detected by the Fortran compiler not the Flecs translator. In addition there are two major classes of Flecs errors which will be caught by the compiler not the translator.

The first class of undetected errors involve misspelled Flecs keywords. A misspelled keyword will not be recognized by the translator. The line on which it occurs will be assumed to be a Fortran statement and will be passed unaltered to the compiler which will no doubt object to it. For example a common error is to spell UNTIL with two L's. Such statements are passed to the compiler, which then produces an error message. The fact that an intended control phrase was not recognized frequently causes a later context error since a level of indentation will not be triggered.

The second class of undetected errors involves unbalanced parentheses. (See also note 8 in section 8.0). When scanning a parenthesized specification, the translator is looking for a

matching right parenthesis. If the matching parenthesis is encountered before the end of the line the remainder of the line is scanned. If the remainder is blank or consists of a recognizable internal procedure reference, all is well. If neither of the above two cases hold, the remainder of the line is assumed (without checking) to be a simple Fortran statement which is passed to the Compiler. Of course, this assumption may be wrong. thus the statement

```
"WHEN (X.LT.A(I)+Z))  X = 0"
```

is broken into

```
keyword "WHEN"
specification "(X.LT.A(I)+Z)"
Fortran statement ")  X = 0"
```

Needless to say the compiler will object to ") X = 0" as a statement.

Programmers on batch oriented systems have less difficulty with undetected errors due to the practice of running the program through both the translator and the compiler each time a run is submitted. The compiler errors usually point out any errors undetected by the translator.

Programmers on timesharing systems tend to have a bit more difficulty since an undetected error in one line may trigger a context error in a much later line. Noticing the context error, the programmer does not proceed with compilation and hence is not warned by the compiler of the genuine cause of the error. One indication of the true source of the error may be an indentation failure at the corresponding point in the listing.

9.4 Other Errors

The translator detects a variety of other errors such as multiply defined, or undefined procedure references. The error messages are self-explanatory. (Really and truly!)

10.0 Procedure for use

The following subsections describe the procedures for using the Flecs translator on various machines at the University of Oregon.

10.1 On the PDP-10

10.1.1 Source Preparation

Prepare a Flecs source file with any name of your choosing and an extension of ".FLX". The translator will accept either line-numbered (SOS, LINED, EDITS) or non-line-numbered (TECO) files. The advantage of line numbered files is that the translator and compiler error messages may be related directly to the source file without reference to a listing. As with Fortran the "tab to column 7" convention may be used.

10.1.2 Compile Commands

The Compile class commands (COMPILE, EXECUTE, LOAD, etc.) recognize the extension .FLX and will invoke the Flecs translator when necessary. When invoked, the Flecs translator will send any error messages to the TTY and will normally produce an .F4 file. The /NOBIN switch will suppress production of the .F4 file and should only be used with the "COMPILE" command. The /LIST switch will cause the translator to produce an indented and formatted source listing with extension .LST which may then be TYPE or PRINT´ed.

Examples: (Assume files A.FLX, B.F4, C.MAC).

.EX A, B, C	Produce A.F4 using Flecs, then compile A.F4 and B.F4, then assemble C.MAC, then load and execute A.REL, B.REL, and C.REL.
.COM/NOBIN/LIST A	Produce an indented listing of A.FLX but don´t produce A.F4.
.COM A	Run A.FLX through Flecs, then A.F4 through Fortran.

Note 1: Uninvoked internal procedures and too many or too few "FIN"´s preceeding a TO or END statement are considered minor errors by Flecs. All others are considered major. If any major errors are detected by the translator, it will abort any following compilation, loading, and execution.

Note 2: (COMPIL invokes the Flecs translator whenever the ".F4" file is missing or older than the ".FLX" file, regardless of the existance or time of a ".REL" file. If you wish to save disk space by deleting the .F4 file, you must then use .EX A.REL or ,EX/REL A to avoid retranslation and recompilation.

Note 3: If COMPIL finds an .F4 file which is newer than the .FLX file it assumes (without looking) that the .REL file also exists. LINK will be unhappy if this is not true. To create a new .REL file without retranslation, do .EX A.F4.

10.1.3 Explicit Invocation

Flecs may be invoked explicitely by ".R FLECS". Flecs will prompt with a "*" to which you may respond with any of the command formats below:

COMMAND	ACTION
<CR>	TERMINATE EXECUTION
C	F4 LST ERR
=C	ERR
,=C	ERR
A=C	F4 ERR
A,=C	F4 ERR
,B=C	LST ERR
A,B=C	F4 LST ERR

where blanks may be used freely and
<CR> represents a carriage return
A,B,C represent file specifications (see below).
F4 means an ".F4" file will be produced
LST means an ".LST" file will be produced.
ERR means error messages will be sent to the TTY.

File Specification Format

DEV:FNAME.EXT[PPN]

SYMBOL	MEANING	DEFAULT IF OMITTED
DEV:	device	DSK:
FNAME	file name	must be specified in all cases except command format "C" where the name given to the ".FLX" file is also used for the ".F4", and ".LST" files.
.EXT	extension	".F4" for A above ".LST" for B above ".FLX" for C above
[PPN]	proj,prog #	same as job using Flecs.

Note: the Flecs translator will run approximately 20% faster for each output file omitted.

10.2 On the IBM S/360

On the IBM S/360 there are two ways of accessing Flecs which have come to be known as WATFLECS and Standard Flecs. WATFLECS is a specially adapted version of the Flecs translator which processes batches of short jobs using the WATFIV compiler and is used primarily in connection with Computer Science classes. Standard Flecs is a stand alone Flecs translator used for larger production programs, usually in conjunction with the level G Fortran compiler. Catalogued procedures which are analogous to those for Fortran(G) exist for using Standard Flecs. WATFLECS is accessed through a special submission process. The same Flecs translation logic is used for both systems so the only language differences are those due to the incompatibilities of the corresponding Fortrans.

10.2.1 WATFLECS

The procedure for preparing and submitting a program under WATFLECS is almost identical to the procedure for submitting a job under WATFIV. The deck set up is shown below

```
$JOB  100557/yourname,any-desired-watfiv-parameters,KP=29
      .
      .
  Flecs source program
      .
      .
$ENTRY
      .
      .
  data cards (if any)
      .
      .
```

Steps in submitting a WATFLECS job:

1. Prepare the Flecs program or programs and data cards on an 029 keypunch. Although WATFIV will, the WATFLECS translator will not accept cards punched on an 026 keypunch.

2. Prepare a <u>solid pink $JOB card</u> as shown above.
   a) The characters "$JOB" should begin in column 1.
   b) The account number 100557 should begin in column 7, followed by a "/".
   c) Fill in your name followed by a comma.
   d) Supply any desired WATFIV parameters. Note: the WATFIV part of the run will be limited to 6 seconds.
   e) Supply the required "KP=29" parameter.

3. Place the Flecs program behind the $JOB card.

4. Place a card with "$ENTRY" beginning in column 1 behind the program. The $ENTRY card must always be present whether or not there are any data cards.

5. Place any data cards behind the $ENTRY card.

6. Place a rubber band around the deck and submit to program reception. The receptionist will place a numbered comment card in the program and give a duplicate card as a receipt.

7. Check the latest WATFLECS job number posted on the blackboard at program reception. As soon as the posted number is greater than or equal to the receipt number, pick up the deck and listings by presenting the receipt card.

Notes on preparing a WATFLECS program:

1. An 029 keypunch must be used.

2. WATFIV does not follow the ANSI standard for Fortran in that it does not allow a Fortran program to jump out of the scope of a DO and later jump back in. Since Flecs internal procedure calls are implemented by GO TO's, it is not possible to reference an internal procedure within the scope of a DO loop when using WATFLECS. The other loop structures may be used to simulate a DO loop, however.

3. Terminate a WATFLECS main program by placing a STOP statement ahead of the first internal procedure declaration.

4. The following unit numbers are available in WATFLECS.

Unit	Purpose
1	card input (from $ENTRY cards)
2	undefined
3	printed output
4-7	scratch disk (read/write)
8-10	class input data sets (read only)

5. The various "$" cards which control the listing of a WATFIV program may be included in the program but will be ignored by the current WATFLECS system.

6. The user may wish to employ the NOWARN and NOEXTEN WATFIV-parameters since Flecs generated Fortran triggers a lot of warnings and extensions.

10.2.2 Standard Flecs

A cataloged JCL procedure exists for using the Flecs translator as a stand alone program. In addition cataloged procedures exist for running Flecs followed by Fortran (G).

Data sets for the translator:

The translator requires three data sets with the following DD names.

LIST	is the output data set containing the Flecs listing.
FORTOUT	is the output data set containing the Fortran source produced by the translator.
SYSIN	is the input data set containing the Flecs source.

The DCB information for these data sets is given below. It is fixed by and contained in the program hence need not be specified in the JCL.

```
LIST      DCB=(RECFM=FA,BLKSIZE=133)
FORTOUT   DCB=(RECFM=F,BLKSIZE=80)
SYSIN     DCB=(RECFM=F,BLKSIZE=80)
```

Using the translator as a stand alone program:

The cataloged procedure FLECS is a one step procedure which runs the Flecs translator. The user must supply the SYSIN data set. Default DD statements in the procedure send LIST to the printer and FORTOUT to a dummy data set. If desired, these DD statements may be overridden as illustrated in the examples below.

Example: Obtaining a printed Flecs listing and ignoring the Fortran source produced.

```
// jobname      JOB      accounting information
// stepname     EXEC     FLECS
// SYSIN        DD       *

                  Flecs source program

/*
```

Example: Obtaining a printed version of both the Flecs listing and the Fortran source.

```
// jobname      JOB      accounting information
// stepname     EXEC     FLECS
// FORTOUT      DD       SYSOUT=A
// SYSIN        DD .     *

                  Flecs source program

/*
```

Example: Obtaining a printed version of the Flecs listing and passing the Fortran source to the Fortran(H) compiler for compilation only. (This example illustrates the general method of passing the Fortran source on to a subsequent step and suppressing the Fortran listing.)

```
// jobname      JOB      accounting information
// stepname1    EXEC     FLECS
// FORTOUT      DD       DSNAME=&TEMP,DISP=(NEW,PASS),
//                       UNIT=SYSDA,SPACE=(CYL,(1,1))
// SYSIN        DD       *

                     Flecs source

// stepname2    EXEC     FORTHC,PARM.FORT='NOSOURCE'
// FORT.SYSIN   DD       DSNAME=&TEMP,DISP=(OLD,DELETE)
```

Using the translator with Fortran (G):

Several cataloged procedures have been established which simplify the process of using Flecs together with Fortran (G). The procedure names are given below together with the Fortran procedures to which they correspond.

FLECSC	FORTGC
FLECSCL	FORTGCL
FLECSCLG	FORTGCLG
FLECSGO	FORTGO

The reader is assumed to be familiar with the "FORTG" procedures. The Flecs procedures have been derived from the FORTG procedures by adding an initial step named FLECS which runs the Flecs translator and which passes the FORTOUT data set to the following step. Since error messages from the Fortran compiler will contain the line number of the original Flecs source statement, the programmer will have little occasion to use the source listing produced by the Fortran compiler. For this reason the source listing from the Fortran compiler has been suppressed by including

a 'NOSOURCE' parameter for the Fortran compiler.

Example: Translating, compiling, linkage editing and executing a Flecs program with previously compiled object decks and data.

```
// jobname      JOB      accounting information
// stepname     EXEC     FLECSCLG
// FLECS.SYSIN  DD       *

                Flecs source program

/*
// LKED.SYSIN   DD       *

  Previously compiled or assembled object decks

/*
// GO.SYSIN     DD       *

                    input data

/*
```

Note: To obtain the source listing from the Fortran compiler, replace the "EXEC" card above with the following:

```
// stepname     EXEC     FLECSCLG,PARM.FORT='SOURCE'
```

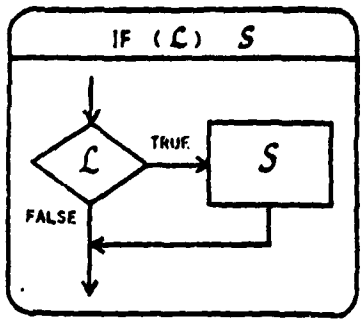
In general, a Flecs run using one of the Flecs procedures is identical to a Fortran(G) run using the corresponding FORTG procedures with the following changes:

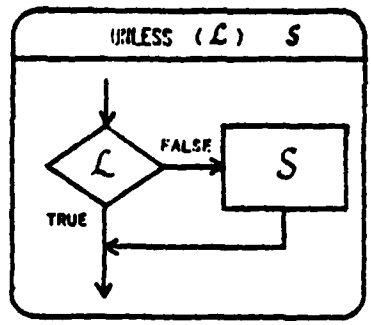
1. Change the procedure name from FORTGxxx to FLECSxxx (exception FORTGO becomes FLECSGO)

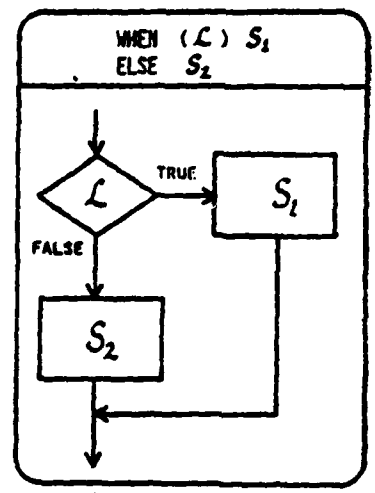
2. Change the SYSIN DD card from FORT.SYSIN to FLECS.SYSIN.

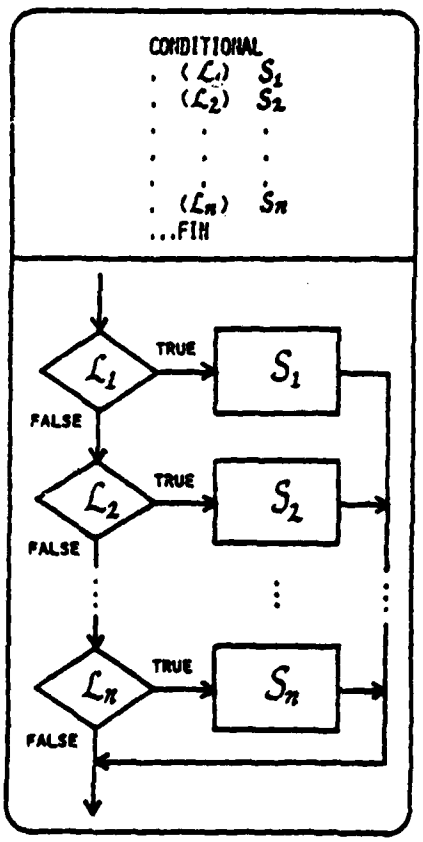
3. If desired, override the suppression of the source listing by including PARM.FORT='SOURCE' on the EXEC card as described above.

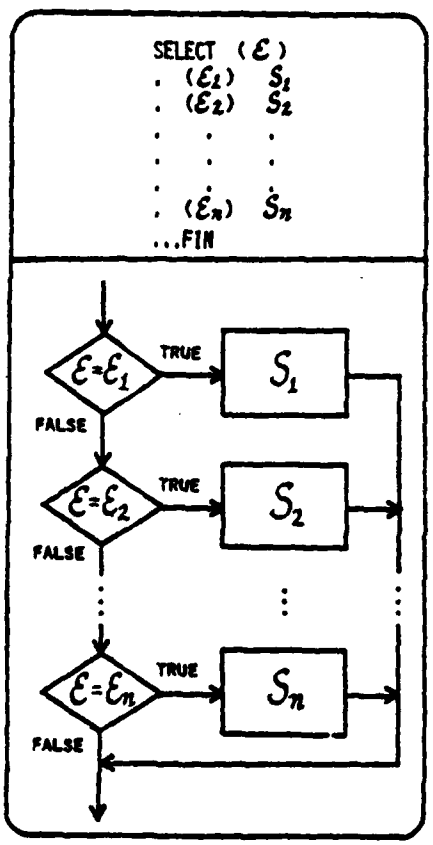
Appendix A: Flecs Summary Sheet

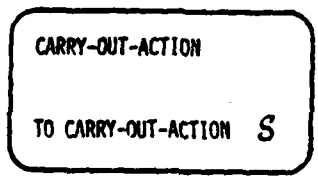


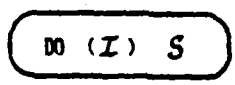








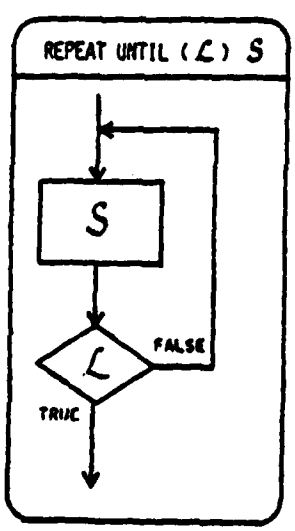


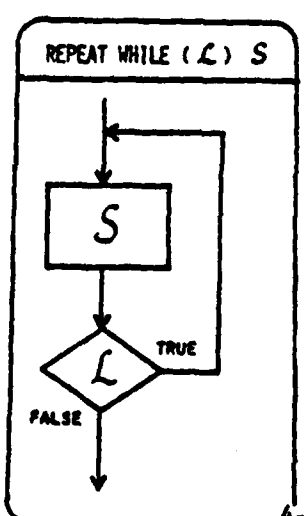


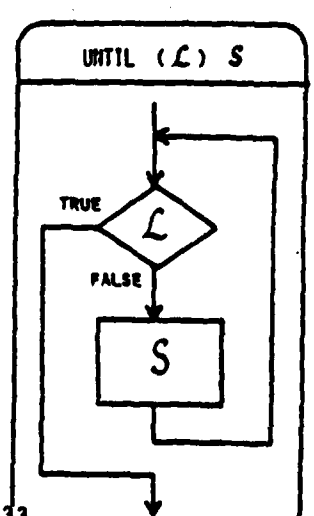
NOTE: PLACE A RETURN, STOP, OR CALL EXIT STATEMENT AHEAD OF THE FIRST TO STATEMENT.

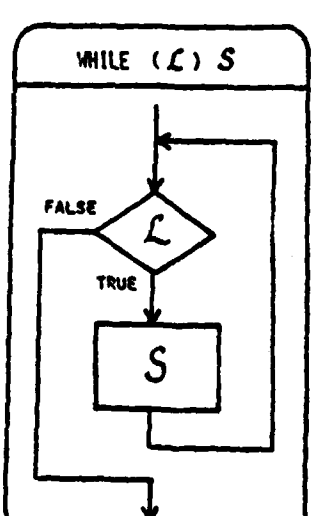
NOTE: OTHERWISE CAN BE USED AS A CATCHALL CONDITION OR EXPRESSION IN CONDITIONAL AND SELECT STATEMENTS.

LEGEND: $\mathcal{L}$ = LOGICAL EXPRESSION
$\mathcal{S}$ = STATEMENT(S)
$\mathcal{E}$ = EXPRESSION
$\mathcal{I}$ = DO SPECIFICATION

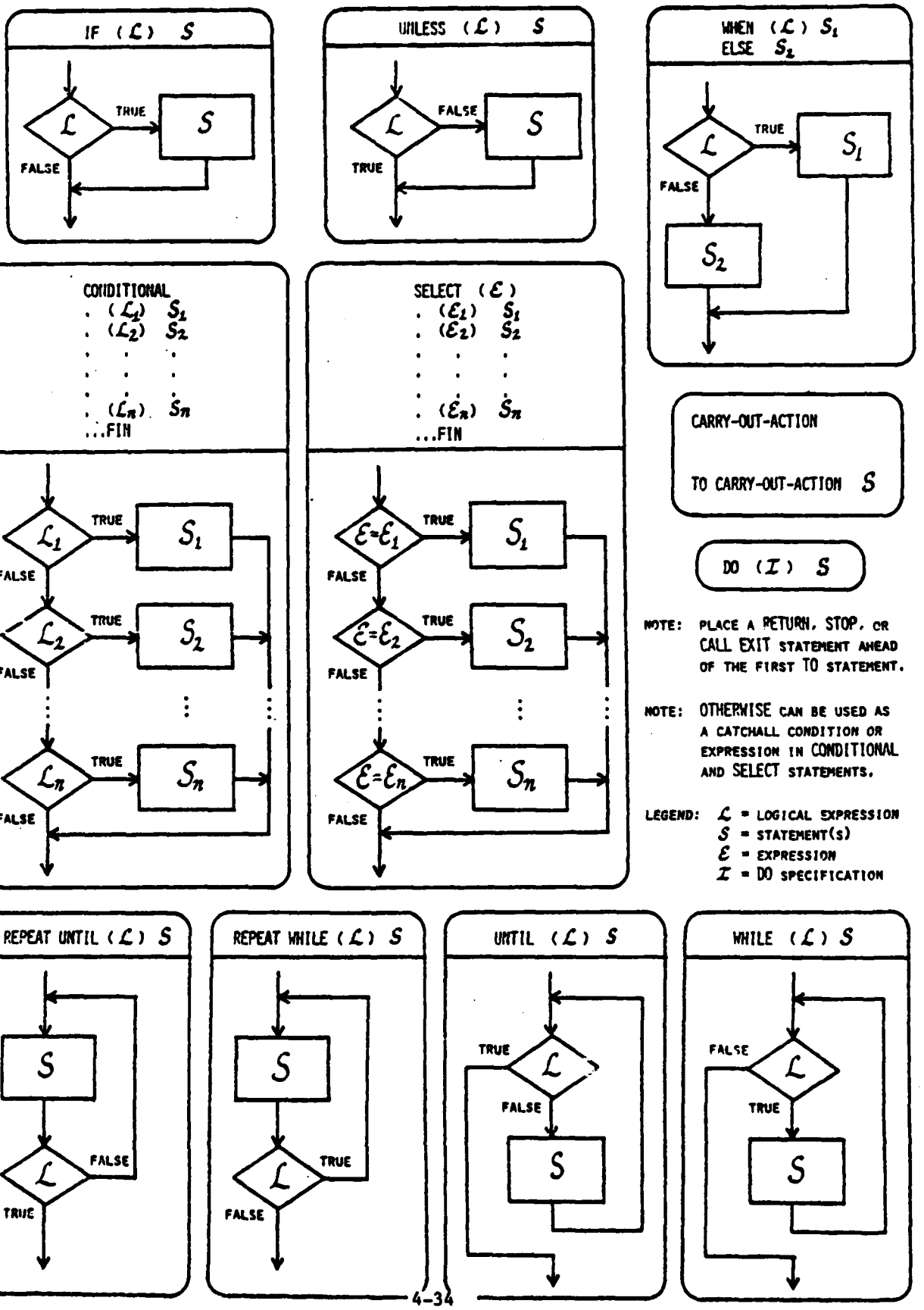








This duplicate Flecs Summary Sheet may be removed from the manual.



APPENDIX B: Available Documentation Concerning Flecs
(As of December 1974.)

Beyer, T., Flecs Users Manual (University of Oregon Edition)

Contains a concise description of the Flecs extension of Fortran and of the details necessary to running a Flecs program on the PDP-10 or the IBM S/360 at Oregon.

Beyer, T., Flecs: System Modification Guide

Contains information of interest to anyone who wishes to install or adapt the Flecs system to a new machine or operating system. Also of interest to those who wish to improve the efficiency of the system by rewriting portions of the system in assembly language.