60455940

CONTROL DATA CORPORATION

TERMINAL-INDEPENDENT GRAPHICS SYSTEM (TIGS) VERSION 1.1 REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:

CYBER 170 SERIES

CYBER 70

MODELS 72, 73, 74

6000 SERIES

TIGS SUBROUTINE SUMMARY

Subroutine	Page
ALARM(lon)	8-1
ARCA(cx,cy,x,y)	2-1
ARCA3(cx,cy,cz,x,y,z,xdir,ydir,zdir)	2-2
ARCDA(cx,cy,deg)	2-1
ARCDA3(cx,cy,cz,ddeg,xdir,ydir,zdir)	2-2
ARCDR(cdx,cdy,ddeg)	2-1
ARCDR3(cdx,cdy,cdz,ddeg,xdir,ydir,zdir)	2-2
ARCR(cdx,cdy,dx,dy)	2-1
ARCR3(cdx,cdy,cdz,dx,dy,dz,xdir,ydir,zdir)	2-2
BLINDS(idpict,ldown)	3-2
CLRSCR	8-1
CLRSTK	5-4
CLRST3	5-4
CLSPIC	3-2
CLSSEG	3-2
COPY(idseg,newseg)	3-2
DELPIC(idpict)	3-3
DELSEG(idseg)	3-3
DELVUP(idport)	4-2
DELWIN(idwind)	4-2
DOTA(x,y)	2-3
DOTA3(x,y,z)	2-3
DOTR(dx,dy)	2-3
DOTR3(dx,dy,dz)	2-3
DRAWA(x,y)	2-3
DRAWA3(x,y,z)	2-3
DRAWR(dx,dy)	2-3
DRAWR3(dx,dy,dz)	2-3
DSPLAY	8-1
EMPTY(idseg)	3-3
ENDPAR(cx,cy,x1,y1,x2,y2,narcs,xn1,yn1,xn2,yn2)	6-1
ENDPLN(x1,y1,x2,y2,ishow,xn1,yn1,xn2,yn2)	6-2
ENDPL3(x1,y1,z1,x2,y2,z2,ishow,xn1,yn1,zn1,xn2, yn2,zn2)	6-2
EVENT(lky,ids,coords,iremng)	7-2
EXTPIC(idpict)	3-3
EXTSEG(idseg)	3-4
IERROR(ierr)	9-1
INITIG(lsquar,lnwfil,filnam)	8-2
KYAC(idky,iactn)	7-3
KEYBRD(maxchr,nchrs,itext)	7-3
KYOFF	7-4
KYON	7-4
LCKSEG	3-4
LOCATE(x,y,iremng)	7-4
MOVEA(x,y)	2-4
MOVEA3(x,y,z)	2-4
MOVER(dx,dy)	2-4
MOVER3(dx,dy,dz)	2-4
OPNPIC(idpict)	3-4
OPNSEG(idseg)	3-5
PLOTA(npoint,xary,yary,line)	2-4
PLOTA3(npoint,xary,yary,zary,line)	2-4
PLOTR(npoint,dxary,dyary,line)	2-4
PLOTR3(npoint,dxary,dyary,dzary,line)	2-4
POP	5-4
POP3	5-4
PREEVN(lucord,idvuwi)	7-5
PRELOC(lucord,idvuwi)	7-5
PROMPT(nchar,itext)	7-6
PUSH	5-4
PUSH3	5-4
QUITIG(ldelet)	8-3
RAAC(idseg,iactn)	7-6
RACSIZ(idseg,wide,high)	2-5
RADSIZ(idseg,wide,high)	2-6
RAFONT(idseg,ifont)	2-7
RAHILT(idseg,lhilit)	2-7
RAINFO(idseg,ninfo,info)	7-7
RAINT(idseg,finten)	2-8
RAPICT(idseg,idpict)	3-5
RAROT(idseg,deg)	2-8
RAROT3(idseg,xbase,ybase,zbase,xplane,yplane, zplane)	2-9
RASTYL(idseg,istyle)	2-9
RASYM(idseg,isym)	2-10
RAVIS(idseg,lvis)	2-10
RAXFA(idseg,bmat23)	5-8
RAXFA3(idseg,bmat34)	5-8
RAXFL(idseg,bmat23)	5-8
RAXFL3(idseg,bmat34)	5-8
RAXFR(idseg,bmat23)	5-8
RAXFR3(idseg,bmat34)	5-8
REMSCR	8-3
RENAME(idold,idnew)	3-5
RTANGL(xll,yll,xur,yur)	6-2
RTANG3(xllh,yllh,zllh,xury,yury,zury)	6-2
SCRNUR(idwind,xscrn,yscrn,xuser,yuser)	8-3
SMAC(iactn)	7-6
SMCSIZ(wide,high)	2-5
SMDSIZ(wide,high)	2-6
SMERR(routin)	9-1
SMFONT(ifont)	2-7
SMHILT(lhilit)	2-7
SMID(idintr)	7-7
SMINFO(ninfo,info)	7-7
SMINT(finten)	2-8
SMLOCR(ilocr)	7-7
SMPICT(idpict)	3-5
SMPLIM(xll,yll,xur,yur)	3-6
SMPLI3(xllh,yllh,zllh,xury,yury,zury)	3-6
SMPORT(idport)	4-7
SMROT(deg)	2-8
SMROT3(xbase,ybase,zbase,xplane,yplane,zplane)	2-9
SMSTYL(istyle)	2-9
SMSVP(lscren,xll,yll,xur,yur)	4-8
SMSYM(isym)	2-10
SMVIS(lvis)	2-10
SMXFA(bmat23)	5-8
SMXFA3(bmat34)	5-8
SMXFL(bmat23)	5-8
SMXFL3(bmat34)	5-8
SMXFR(bmat23)	5-8
SMXFR3(bmat34)	5-8
TAAC(idseg,iactn)	7-6
TACSIZ(idseg,widout,hiout)	2-5
TADSIZ(idseg,widout,hiout)	2-6
TAFONT(idseg,ifont)	2-7
TAHILT(idseg,lhilit)	2-7
TAINFO(idseg,ninfo,info)	7-7

60455940

CONTROL DATA
CORPORATION

TERMINAL-INDEPENDENT GRAPHICS SYSTEM (TIGS) VERSION 1.1 REFERENCE MANUAL

CDC® COMPUTER SYSTEMS:
CYBER 170 SERIES
CYBER 70
MODELS 72, 73, 74
6000 SERIES

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(06-12-78)	
B	
(01-12-79)	Manual revised to add description of Sanders Graphic 7 postprocessor and to make miscellaneous corrections.
C	
(07-20-79)	Manual revised to reflect version 1.1 features and to make miscellaneous corrections. This manual obsoletes
	all previous editions.
Publication No. 60455940	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:
Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112
or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Front Cover	-
Inside Cover	-
Title Page	-
ii	C
iii/iv	C
v/vi	C
vii	C
viii	C
1-1	A
1-2	A
1-3	A
1-4	C
1-5	A
2-1	C
2-2	A
2-3	A
2-4	C
2-5	C
2-6	C
2-7	C
2-8	C
2-9	C
2-10	C
2-11	C
2-12	C
2-13	C
3-1	B
3-2	A
3-3	C
3-4	C
3-5	C
3-6	C
3-7	C
3-8	A
3-9	A
4-1	A
4-2	C
4-3	C
4-4	C
4-5	C
4-6	C
4-7	C
4-8	C
4-9	C
4-10	C
4-11	C
5-1	A
5-2	A
5-3	A
5-4	A
5-5	B
5-6	B
5-7	B
5-8	A
5-9	C
5-10	A
5-11	A
5-12	A
5-13	A
5-14	A
5-15	A
6-1	A
6-2	C
6-3	A
7-1	B
7-2	C
7-3	C
7-4	B
7-5	A
7-6	C
7-7	A
7-8	C
7-9	B
7-10	A
7-11	A
8-1	C
8-2	C
8-3	C
8-4	C
8-5	C
8-6	C
9-1	C
9-2	C
A-1	A
B-1	A
B-2	A
C-1	A
C-2	C
C-3	A
C-4	A
C-5	C
C-6	C
C-7	C
C-8	C
C-9	C
D-1	A
D-2	A
E-1	C
E-2	C
E-3	C
E-4	C
E-5	C
F-1	C
F-2	C
F-3	C
F-4	C
F-5	C
F-6	C
F-7	C
F-8	C
F-9	C
F-10	C
F-11	C
F-12	C
F-13	B
F-14	C
G-1	C
G-2	C
G-3	C
G-4	C
G-5	C
G-6	C
G-7	C
G-8	C
Index-1	C
Index-2	C
Index-3	C
Comment Sheet	C
Inside Back Cover	-
Back Cover	-

PREFACE

This manual describes the Terminal-Independent Graphics System (TIGS). It is in reference format and contains descriptions of TIGS external characteristics together with descriptions and calling formats of each of the software routines that compose TIGS.

TIGS is available through the NOS and NOS/BE operating systems of the CONTROL DATA® CYBER 70 Series Models 71, 72, 73, and 74, CDC® CYBER 170 Series, and CDC® 6000 Series Computer Systems. TIGS software communicates with the host computer via the interactive facility of the operating system: NAM/IAF or the Time-Sharing Module under NOS, and INTERCOM under NOS/BE.

This manual is intended as a reference source and programming guide for all users of TIGS. A familiarity with FORTRAN and the operating system under which TIGS will operate is assumed.

RELATED PUBLICATIONS

Control Data Publication	Publication Number
NOS/BE 1 Reference Manual	60493800
INTERCOM Version 4 Reference Manual	60494600
INTERCOM Version 5 Reference Manual	60455010
INTERCOM Version 4 Guide for Users of FORTRAN Extended	60495000
INTERCOM Version 5 Guide for Users of FORTRAN Extended	60455950
NOS Version 1 Reference Manual – Volume 1	60435400
Time-Sharing User's Reference Manual	60435500
Network Products Interactive Facility Version 1 Reference Manual	60455250
CYBER Loader Reference Manual	60429800
UNIPLOT Version 2.1 Reference/ User Guide	60454730
Terminal Independent Graphics System Instant	60456360
Terminal Independent Graphics System (TIGS) User's Guide	60456040
FORTRAN Extended Version 4 Reference Manual	60497800
Beginning Graphics User's Guide	76077300

The following manuals are available only from Tektronix, Inc., P.O. Box 500, Beaverton, Oregon, 97077.

Tektronix 4006 and 4006-1 Terminal User's Guide	070-1891-00
Tektronix 4010 and 4010-1 User's Manual	070-1225-00
Tektronix 4014 and 4014-1 Computer Display Terminal User's Instruction Manual	070-1647-00
Tektronix 4631 Hardcopy Unit User's Manual	070-1830-00
Tektronix 4953/4954 Graphics Tablet Instruction Manual	070-1791-00
Tektronix Data Communications Interface Instruction Manual	070-2026-00

The following manual is available only from Sanders Associates, Inc., Daniel Webster Highway, Nashua, New Hampshire, 03061.

Graphic 7 Computer Graphics Display System Graphics Support Software (GSS-4) Version 1.1 User's Manual	H-78-0047

DISCLAIMER

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or undefined parameters.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

GENERAL DESCRIPTION

The Terminal-Independent Graphics System (TIGS) is a graphics software package designed to support a variety of graphics display terminals while providing a high degree of application program independence from constraints caused by differing terminal characteristics. Under TIGS, it is not necessary to write programs tailored for the capabilities of a particular terminal, although a programmer may do so if he chooses.

TIGS consists of a single software preprocessor providing display generation and interactive capabilities for a general class of terminals and software postprocessors which translate the generalized preprocessor output into specific display instructions for the various terminal types supported by TIGS. Data is conveyed between the two processor packages by means of a neutral display file (NDF). Version 1 of TIGS supports a single postprocessor which is the interface for storage tube graphics terminals. Postprocessor information is contained in appendixes to this manual; the main body of the manual describes only the terminal independent preprocessor.

OPERATING ENVIRONMENT

TIGS is available through the NOS and NOS/BE operating systems of the CDC CYBER 70 Series, CDC CYBER 170 Series, and 6000 Series computers. TIGS software communicates with the host computer via the interactive facility of the operating system: NAM/IAF or the Time-Sharing Module under NOS, and INTERCOM under NOS/BE. Full time-sharing facilities are available at the terminal. This manual does not provide basic operating system information related to interactive or batch job processing. Only those points of job processing which are unique to TIGS will be described here.

The manual also assumes a good working knowledge of FORTRAN. All TIGS routines are called as subroutines or functions from programs coded in FORTRAN Extended (FTN) and are contained on special TIGS program libraries. An effort has been made to adhere to ANSI FORTRAN usage in the interest of clarity. FORTRAN coding conventions are followed.

TIGS has been designed for ease of learning by having a small number of parameters in any graphics routine call. Parameter conventions are described at the end of this section.

MANUAL OVERVIEW

Each section of this manual contains the TIGS calls appropriate to the topic being discussed and supporting explanatory material. The calls dealt with in a given section are listed alphabetically at the beginning of each section. The format for each call includes the call and all its parameters, an explanation of the call's function, a list of parameter descriptions, programming notes if indicated, and usage examples where the routine's function is not self-evident.

Besides the postprocessor material mentioned already, the appendixes include a glossary, a list of error messages, and other TIGS supplementary material related to the main body of the manual.

STRUCTURAL OVERVIEW OF TIGS

TIGS gives a programmer the capability to construct, organize, and display graphic material in either two- or three-dimensional mode, and to interactively modify any display. In order to understand how to use TIGS calls properly and efficiently, it is desirable to know how these displays are constructed.

To produce a display on the terminal screen, the programmer must complete a two-part task. First, to model the display, the programmer uses TIGS structural building blocks: primitives, segments, and pictures to define and spatially organize objects in two- or three-dimensional space. Using TIGS window and viewport routines, the programmer then develops a viewing pattern, consisting of instructions on how the constructed model will appear on the terminal screen. TIGS applies this viewing pattern against the model to produce the display on the terminal screen. Figure 1-1 shows how these TIGS elements relate to each other.

PRIMITIVES

Primitives are the structural building blocks of TIGS. They include lines, arcs, dots, plotting symbols, and text. These are the basic drawing elements which the graphics terminals are capable of producing, and which, in combination, make up even the most complex graphics displays. There are separate routines for two- and three-dimensional primitives. Three-dimensional primitives are identical to two-dimensional primitives with the addition of sufficient parameters to describe the primitives in the added dimension. In generating primitives, the programmer is exercising the primary capabilities of TIGS to draw pictures. All other TIGS routines are concerned with organizing or otherwise affecting what has already been laid out in primitives. Section 2 fully describes primitives.

SEGMENTS AND PICTURES

A segment is a collection of primitives organized for purposes of display manipulation. Segments constitute the lowest organizational level at which display modification can take place. In a TIGS program, a segment is defined and primitives are placed into it; thereafter, no modification of the display can affect a level lower than a single segment. Segments can be left undefined so that all primitives in a display belong to one default segment, but usually the display is logically grouped into segments for purposes of later segment-by-segment manipulation.

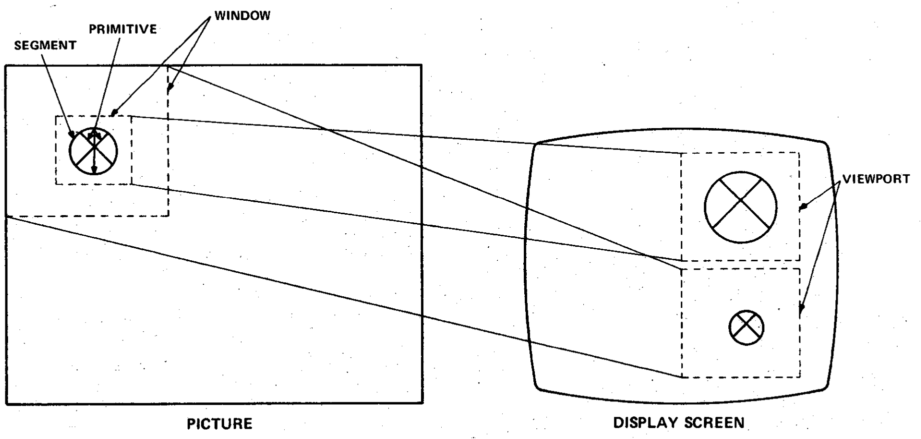


Figure 1-1. Relationship of TIGS Elements

Similarly, segments can be grouped into pictures for purposes of display manipulation. Certain TIGS operations can be specified which affect whole pictures. The programmer may use the default picture, or one or more nondefault pictures. A picture is a bounded area defined by programmer-supplied or defaulted picture limits. A three-dimensional picture is more properly called a picture space, analogous to a two-dimensional picture but with the inclusion of the third dimension.

Pictures and segments are described in section 3.

WINDOWS AND VIEWPORTS

The programmer uses a window to control how much of the picture is displayed by selecting all or part of the picture to include in the window. The programmer can use the default window, or one or more nondefault windows, and can overlap windows and superimpose them on one another.

The viewport is a certain portion of the display screen. The display screen can consist of the default viewport which covers the usable screen or one or more nondefault viewports dividing the screen to suit the application program.

Window is to picture as viewport is to the terminal display screen. The assigning of window to viewport determines how the display will eventually appear to the viewer.

A three-dimensional window is a subset of the total volume of the picture space. The three-dimensional window specifies how the simulation of space behind the display screen will occur. Orthogonal projections and perspective projections are both possible.

MODES, ATTRIBUTES, AND FEATURES

In keeping with the TIGS design objective of programming simplicity, TIGS employs the concept of modality. A mode is a condition, set or defaulted, which affects all subsequent program operations until changed. There are always modes in effect. Modes control a wide variety of program conditions, among them the assigning of window to viewport and the assigning of qualities to primitives (this process is described in a following paragraph).

There are obvious advantages to the mode system. When it is desirable to set a program condition which will prevail for a series of calls, a modal setting obviates the need to specify that condition with every call. Also, the mode system permits the TIGS calls to be more compact and easily remembered with fewer parameters. For example, to construct a triangle for display, the programmer uses primitives to specify the spatial nature and location of the triangle. Describing such qualities as the style of the line used to draw the triangle is done by setting the appropriate line style mode before the triangle is constructed.

For ease of usage, TIGS provides defaults for all modes, so that a programmer need learn only the routines that set or change modes he wishes to use. It is possible to construct a functioning TIGS program which does not make any calls to subroutines which affect modes. All modes can be left as defaulted, set to a value, or changed from a current value. In addition, routines exist to test for modal settings currently in effect.

Attributes of a segment are qualities inherited from modal settings in effect when that segment was defined. When the first primitive in a segment is defined, those attributes become a part of the segment definition. Attributes can be tested or changed later in the program through the attribute testing and resetting routines. For example, a programmer can change the visibility attribute of a triangle from visible (inherited from the default mode in effect when the triangle was defined) to invisible by specifying the proper reset attribute call.

For ease of learning, TIGS uses a call prefix sequence for the routines of the mode/attribute system. All test mode routines begin with TM prefixes, all set mode routines with SM, all test attribute routines with TA, and all reset attribute routines with RA. For example, when a programmer has learned that there is a series of routines affecting line style, xxSTYL, he knows that there is a test mode call TMSTYL for testing current line style mode, a set mode call SMSTYL to set line style mode, a test attribute call TASTYL to test the line style attribute of a segment, and a reset attribute call RASTYL to reset that attribute. Exceptions to this system are noted in the descriptions of the individual mode/attribute routines.

For most modal qualities there is a companion test feature routine (TF prefix) to ascertain whether and how a given terminal type supports a given feature. The test feature routines allow a programmer to inquire of the postprocessor if a particular feature is supported by that postprocessor, or to allow the postprocessor to describe a particular feature. All but three of the test feature routines (TFHARD, TFNSIZ, and TFSCRN) have companion mode setting and testing routines.

Some test feature routines test for hardware support of a given feature. It should be noted that such features are supported for all postprocessors, regardless of the value returned to the test feature routine. If the test feature routine indicates that a given feature is not hardware-supported, the programmer assumes the feature is software-supported. This capability was included in TIGS because, in general, execution time is greatly increased if a feature must be software-supported; the programmer may elect not to perform operations which use a given software-supported feature because of this consideration.

Each of the reference sections of the manual contains a subsection dealing with the mode/attribute/feature routines which affect the material covered in that section. For example, xxSTYL is described in Primitives (section 2), xxPORT under Windows and Viewports (section 4), and so on.

The following general information about modes and attributes should be kept in mind. Additional notes, cautions, and usage examples are included where necessary in the mode/attribute/feature subsections of each of the reference sections.

- Test routines that return only one value can be called as functions or subroutines; ANSI usage requires that they be called as functions.

- Modes can be set at any point in a program, except as noted in the description of the routines SMPLIx (section 3), SMPICT (section 3), SMAC (section 7), SMHILT (section 2), and SMVIS (section 2). However, caution should be used in setting modes because modes become attributes of segments and if a mode is changed within a segment definition, a subsequent attempt to reset that attribute will yield unpredictable results (refer also to Pictures and Segments).

- There are no reset attribute (RA) routines which do not have companion mode routines. There are modes, however, which cannot become attributes. These are viewport mode (xxPORT, section 4), system viewport (xxSVP, section 4), error routine (xxERR, section 9), locator (xxLOCR, section 7), and intrasegment identifier (xxID, section 7). All other modes can become attributes.

- All attributes can be tested (TA routines). All attributes can be reset except picture limits (xxPLIx, section 3), and picture dimensionality (xx3D, section 3).

- Modes in effect when the first primitive of a segment is defined become attributes of that segment and are stored as part of the segment definition. Calls to the routines SMPICT, EXTPIC, and EXTSEG (all in section 3) reset current modal settings to the modal settings in effect when the picture or segment was defined. SMPICT and EXTPIC reset picture limits, picture identification, and picture dimensionality only. EXTSEG resets all segment attributes.

TRANSFORMATION ROUTINES

Three of the ways a graphics display can be altered are by translation, rotation, and scaling. TIGS uses transformation matrices to carry out these three functions, in both two- and three-dimensional modes. In addition, a programmer can accomplish other matrix multiplication operations, such as shearing, by providing his own matrix and using the proper mode/attribute routine to perform the desired transformation. Transformation matrix operations, associated utility routines, and explanatory material are contained in section 5.

GEOMETRY UTILITIES

TIGS provides the programmer with tools for determining in advance how a display will appear in a given two- or three-dimensional space. These geometry utilities are described in section 6.

INTERACTION

TIGS permits not only passive modification of the terminal graphics display, but has also the programming capabilities to permit interactive display modification based on information returned to the application program as terminal input. Because TIGS is independent of terminal input devices which may vary from terminal to terminal, the programmer can write his programs for virtual devices defined for the preprocessor and let the postprocessor link these virtual devices to actual terminal input devices at the terminal on which the program is run. Interaction is discussed in section 7.

TERMINAL FUNCTIONS

TIGS provides a number of routines for determining terminal characteristics, for influencing the display in very basic ways, and for communicating with terminals. These routines are documented in section 8.

ERROR PROCESSING

Programmers can obtain current error status by using routines supplied as part of the TIGS software. Section 9 deals with this topic.

SUBROUTINE AND PARAMETER CONVENTIONS

A number of arbitrary but orderly rules have been adopted for ordering and naming subroutines and parameters which appear in the calling sequences defined in this manual.

- For coordinates: X variable before Y variable before Z variable.

- For dimensioned quantities: width variable before height variable before depth variable.
- All input variables appear before output variables.
- A count of the number of elements in arrays and character strings appears before the elements. The first character of such parameters is N.
- Identifier names are of the form idxxxx; for example, **idseg**, **idpict**, and so on. The name reflects the kind of identifier.
- Identifiers always appear before modifiers or attributes.
- A d (delta) denotes variables whose values are relative; for example, **dx**, **dy**, and so on.
- Names of logical variables begin with L and the name will reflect the .TRUE. condition of the variable.
- Subroutine names with a noun and a verb always have the verb first; for example, OPNSEG, CLRSCR, EXTSEG, DELWIN, PRELOC, and SMINT.

There are no optional parameters in a TIGS subroutine call; if a programmer calls a subroutine, he must specify all parameters. Default conditions described in this manual are valid only if no call is made to the routine. The programmer may supply any names he wishes for subroutine parameters. However, all error messages use the parameter names given in the subroutine descriptions in this manual.

All variables describing coordinates must be floating point variables.

This manual uses designations for classes of routines (for example, xxPLIx) in which the lower case x's refer to prefixes and suffixes that distinguish individual routines in that class of routines.

All lowercase parameter names reflect the general condition of the parameter. In specific examples of subroutine usage, the parameters are uppercase. Lowercase parameter names are in boldfaced type.

GENERAL FORMAT OF A TIGS PROGRAM

A TIGS program is written in FORTRAN and follows basic FORTRAN conventions. Specific adaptations required by a particular postprocessor are noted in the appropriate postprocessor appendix.

In general, a TIGS program must do the following:

- Initialize communication with TIGS (INITIG, section 8).
- Construct desired pictures (sections 2, 3).
- Organize pictures (section 4).
- Display pictures (DSPLAY, section 8).
- Modify display if desired (sections 5 through 7).
- End communication with TIGS (QUITIG, section 8).

Except for the INITIG and QUITIG calls which are the first and last graphics calls in the program, the order of steps described above need not be rigidly followed.

The following general program example and the flowchart (figure 1-2) illustrate this process. Examples in the reference sections of this manual are sometimes only program fragments to illustrate a particular point and do not adequately show the broader outline of a TIGS program.

```
PROGRAM statement†
    (DIMENSION statement, LOGICAL statement, etc. for reserving storage)
CALL INITIG (.TRUE.,.TRUE.,5LIFILE)                Must be first graphics call
    (set desired modes prior to opening pictures, segments)
CALL OPNSEG(n)
    (primitives)
CALL CLSSEG
CALL DSPLAY
    (interaction calls, if desired)
CALL QUITIG (.TRUE.)
STOP
END
```

†Warning: format of PROGRAM statement is partially determined by postprocessor or operating system requirements. Refer to appropriate postprocessor and operating system appendices.

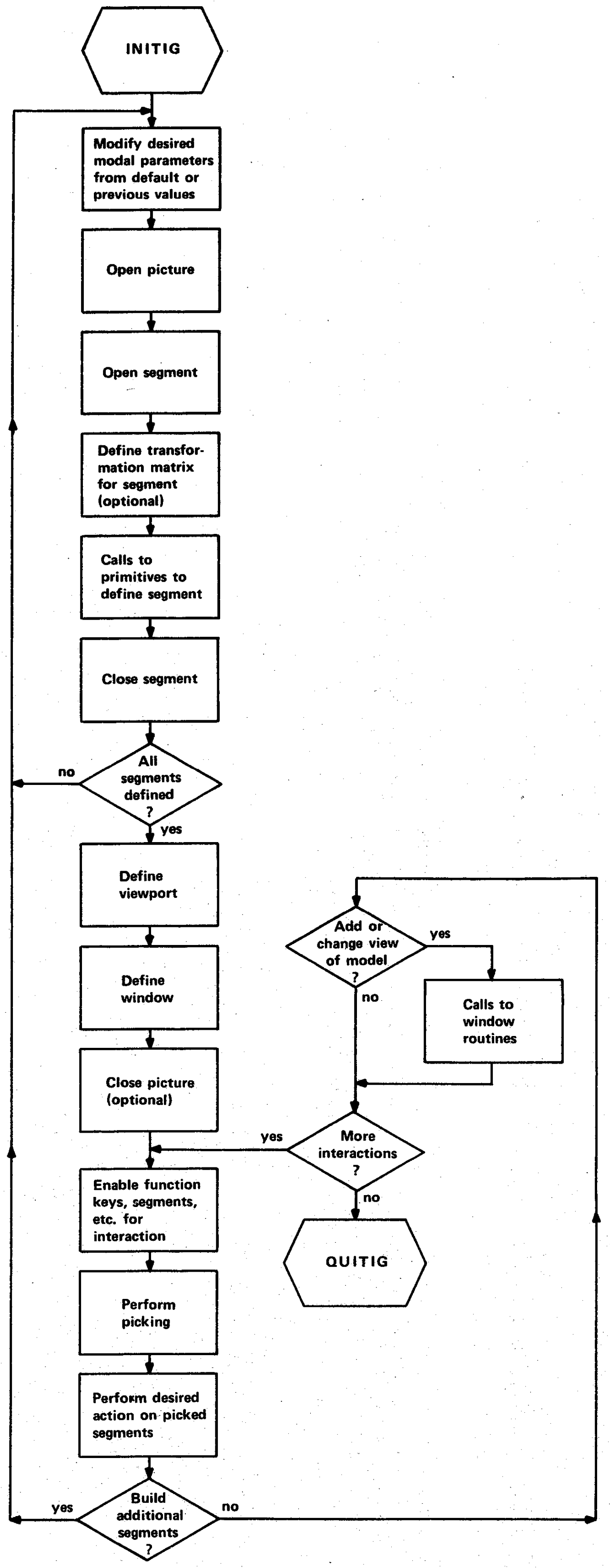

INITIG
Modify desired modal parameters from default or previous values
Open picture
Open segment
Define transformation matrix for segment (optional)
Calls to primitives to define segment
Close segment
All segments defined ?
no
yes
Define viewport
Define window
Close picture (optional)
Enable function keys, segments, etc. for interaction
Perform picking
Perform desired action on picked segments
Build additional segments ?
yes
no
Add or change view of model ?
yes
no
Calls to window routines
More interactions ?
yes
no
QUITIG

Figure 1-2. TIGS General Program Flow

The following TIGS calls are documented in this section.

ARCA	MOVER3	SMINT	TFDSIZ
ARCA3	PLOTA	SMROT	TFFONT
ARCDA	PLOTA3	SMROT3	TFHILT
ARCDA3	PLOTR	SMSTYL	TFINT
ARCDR	PLOTR3	SMSYM	TFNSIZ
ARCDR3	RACSIZ	SMVIS	TFROT
ARCR	RADSIZ	TACSIZ	TFSTYL
ARCR3	RAFONT	TADSIZ	TFSYM
DOTA	RAHILT	TAFONT	TFVIS
DOTA3	RAINT	TAHILT	TMCSIZ
DOTR	RAROT	TAINT	TMDSIZ
DOTR3	RAROT3	TAROT	TMFONT
DRAWA	RASTYL	TAROT3	TMHILT
DRAWA3	RASYM	TASTYL	TMINT
DRAWR	RAVIS	TASYM	TMROT
DRAWR3	SMCSIZ	TAVIS	TMROT3
MOVEA	SMDSIZ	TEXT	TMSTYL
MOVEA3	SMFONT	TEXT3	TMSYM
MOVER	SMHILT	TFCSIZ	TMVIS

This list includes routines which generate primitives, and those mode/attribute/feature routines which affect primitives. The mode/attribute/feature routines are contained in a separate group following the primitive generation calls.

Following the TIGS calls is an example using many of the primitives routines described in this section.

GENERAL

Primitives subroutines are used in the application program to draw lines, arcs, dots, plot symbols, and text on the terminal display surface. Two- and three-dimensional drawing modes use separate routines, but the routines are described together. Text primitives can be used in either two- or three-dimensional segments. However, other two-dimensional primitives can only be used with two-dimensional pictures, windows, and viewports; the situation is similar for three-dimensional pictures, windows, and viewports. Refer to section 3 for more information.

Coordinates for primitives may be specified in absolute or relative terms. Absolute coordinates are the actual coordinates at which the primitive is to be drawn, in terms of user coordinates established when a picture is defined and picture limits are specified (section 3). Relative coordinates state that the primitive is to be drawn at a position relative to the current drawing beam position. Current drawing beam position is defined as the location in user coordinates where the drawing beam is left upon completion of the last TIGS routine affecting beam movement. Beam movement is continuous. Primitives routines with relative coordinates are especially useful in constructing subroutines to a main application program when a figure is to be drawn at an arbitrary location. Present beam position can be determined by a call to WHERE or WHERE3 (section 8).

The programmer should be aware that TIGS employs a right-handed coordinate system for modeling. That is, the positive coordinate axes in three-dimensional space have the basic orientation to each other illustrated in figure 2-1.

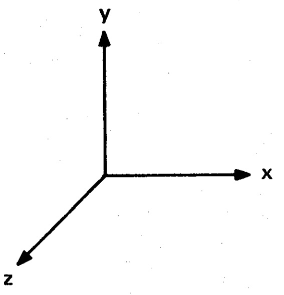


Figure 2-1. Three-Dimensional Coordinate Axes Orientation

The following paragraphs give capsule descriptions of the primitives routines in this section.

The MOVExx, DRAWxx, and DOTxx subroutines position the drawing beam, draw lines, and draw dots, respectively. Absolute or relative coordinates in either two- or three-dimensional mode can be specified for these routines.

The TEXTx subroutines draw a text string starting at the current beam position. Mode and attribute routines specify text characteristics.

Using the PLOTxx routine, a series of points can be plotted with optional plot symbols at each point and optional lines connecting the plotted points. Both two- and three-dimensional plots can be constructed.

Circular arcs can be drawn using the ARCxx routines in two ways: by supplying the coordinates of the center and the end point, or by specifying the center and the angular position of the end point. Either method allows the use of absolute or relative coordinates in either two- or three-dimensional mode.

PRIMITIVES ROUTINES

ARCxxx

Format

ARCA(**cx,cy,x,y**)

Draw absolute 2-D arc to endpoint.

ARCR(**cdx,cdy,dx,dy**)

Draw relative 2-D arc to endpoint.

ARCDA(**cx,cy,deg**)

Draw absolute 2-D arc through angle.

ARCDR(**cdx,cdy,ddeg**)

Draw relative 2-D arc through angle.

ARCA3(**cx,cy,cz,x,y,z,xdir,ydir,zdir**)
: Draw absolute 3-D arc to endpoint.

ARCR3(**cdx,cdy,cdz,dx,dy,dz,xdir,ydir,zdir**)
: Draw relative 3-D arc to endpoint.

ARCDA3(**cx,cy,cz,ddeg,xdir,ydir,zdir**)
: Draw absolute 3-D arc through angle.

ARCDR3(**cdx,cdy,cdz,ddeg,xdir,ydir,zdir**)
: Draw relative 3-D arc through angle.

Parameters

cx,cy,cz	Input parameters; absolute user coordinates of the arc center.
cdx,cdy,cdz	Input parameters; user coordinates of the arc center relative to current beam position.
x,y,z	Input parameters; absolute user coordinates of the arc endpoint.
dx,dy,dz	Input parameters; relative user coordinates of the arc endpoint.
deg	Input parameter; absolute angular position of the arc endpoint, measured in degrees counterclockwise from the X axis. If **deg** is negative, angular measurement is clockwise. Not used with ARCDA3.
ddeg	Input parameter; relative angular position of the arc endpoint, measured in degrees from the radius defined by the arc center and the current beam position. Positive **ddeg** values indicate counterclockwise arc direction for 2-D arcs and arcs in the direction of the direction cosines for 3-D arcs. Negative **ddeg** values imply clockwise direction for 2-D and arcs opposite to the direction of the direction cosines for 3-D arcs.
xdir,ydir,zdir	Input parameters; direction cosines defining the direction of 3-D arcs. In the special cases of semicircles and full circles, the direction cosines also define the plane of the arc.

Programming Notes

Calls to ARCxx are affected by the following subset of the mode/attribute/feature routines described in the mode/attribute/feature subsection.

xxHILT
xxINT
xxSTYL
xxVIS

The modal settings in effect when the first primitive in the segment containing the ARCxx routines is defined determine the attributes that segment will have.

Arcs are drawn as sections of circles only. Drawing of the arc begins from current beam position.

Direction of two-dimensional arcs is counterclockwise except as noted in the description of the **deg** and **ddeg** parameters.

Direction of three-dimensional arcs is in the direction of the direction cosines except as noted in the description of the **ddeg** parameter.

Calls to ARCDA may not yield the results the programmer expects when the y coordinates of the current beam position and the arc center are not the same (that is, the arc radius defined by the arc center and current beam position is not parallel to the x axis). The absolute angular position is measured from the x axis, but drawing always begins from current beam position.

Direction cosines are used for three-dimensional arcs, since clockwise and counterclockwise are not readily applicable terms in three dimensions. The direction indicated by the vector from (x_1,y_1,z_1) toward (x_2,y_2,z_2) corresponds to direction cosines defined:

$$\text{xdir}=\frac{x_2-x_1}{\sqrt{\Delta c^2}} \qquad \text{ydir}=\frac{y_2-y_1}{\sqrt{\Delta c^2}} \qquad \text{zdir}=\frac{z_2-z_1}{\sqrt{\Delta c^2}}$$

$$\sqrt{\Delta c^2} = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}$$

These direction cosines unequivocally point the direction of the arc in the plane of the arc. They are supplied as floating-point values to three-dimensional arc routines. In the case of semicircles and full circles and for the ARCDA3 and ARCDR3 routines, the direction cosines also define the plane of the arc because a single line (established either by the three colinear points of a semicircle or the two colinear points of a circle) does not define a plane, but a line and the vector described by the direction cosines do define a plane.

All that is required of the direction cosines is that they lie in the plane of the circle and on the same side of the line drawn between the arc starting point (current beam position) and the arc center as the direction in which the arc is to be drawn. Direction cosines are considered to be based at current beam position. This situation, viewed in the plane of the arc, is illustrated in figure 2-2.

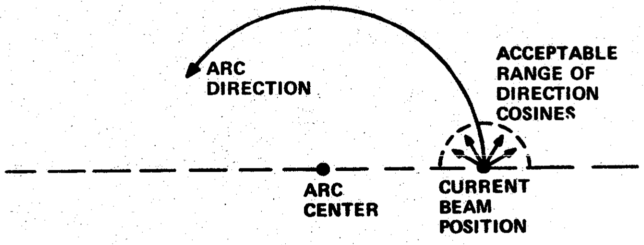


Figure 2-2. Direction Cosines

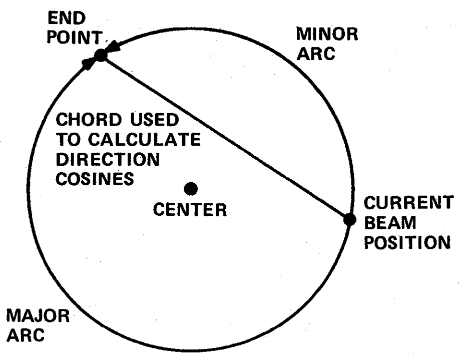


Figure 2-3. Major and Minor Arcs

In many instances, the computation of direction cosines is facilitated by the selection of points that are related to the desired arc. In the cases of the ARCA3 and ARCR3 routines where the arc endpoint is known (and the arc is not a full circle or semicircle), the direction cosines can be computed using the current beam position as one point and the arc endpoint as the other, as shown in figure 2-3.

For ARCDA3 and ARCDR3 and for all full and semicircles where the direction cosines define the plane of the arc as well as the direction of the arc, it is necessary to establish another point to calculate the direction cosines. A point on the arc may be found, or a point that yields a tangent to the circle of the arc, or any other point of the plane on the proper side of the center/starting point line. For full circles, the direction cosines can lie on either side of this line.

To draw arcs opposite to the direction indicated by the direction cosines, multiply the direction cosines by -1. The same effect is achieved by specifying the **ddeg** parameter with a negative sign (ARCDA3 and ARCDR3). For example, the major arc in figure 2-3 can be drawn without recomputing direction cosines by using either of the techniques described.

DOTxx

Format

DOTA(**x,y**)

Draw dot at absolute 2-D position.

DOTR(**dx,dy**)

Draw dot at relative 2-D position.

DOTA3(**x,y,z**)

Draw dot at absolute 3-D position.

DOTR3(**dx,dy,dz**)

Draw dot at relative 3-D position.

Parameters

x,y,z	Input parameters; absolute user coordinates of the dot.
dx,dy,dz	Input parameters; user coordinates of the dot relative to the current beam position.

Programming Notes

Calls to DOTxx are affected by the following subset of the mode/attribute/feature routines described in the mode/attribute/feature subsection.

xxHILT
xxINT
xxVIS

Modal settings in effect when the first primitive in the segment containing the DOTxx routine is defined determine the attributes of that segment.

DRAWxx

Format

DRAWA(**x,y**)

Draw line to absolute 2-D endpoint.

DRAWR(**dx,dy**)

Draw line to relative 2-D endpoint.

DRAWA3(**x,y,z**)

Draw line to absolute 3-D endpoint.

DRAWR3(**dx,dy,dz**)

Draw line to relative 3-D endpoint.

Parameters

x,y,z	Input parameters; absolute user coordinates of endpoint.
dx,dy,dz	Input parameters; user coordinates of the endpoint relative to current beam position.

Programming Notes

Calls to DRAWxx are affected by the following subset of the mode/attribute/feature routines described in the mode/attribute/feature subsection.

xxHILT
xxINT
xxSTYL
xxVIS

Modal settings in effect when the first primitive in the segment containing the DRAWxx routine is defined determine the attributes of that segment.

Lines are drawn from current beam position to the specified endpoint.

MOVExx

Format

MOVEA(**x,y**)

Move beam to absolute 2-D position.

MOVER(**dx,dy**)

Move beam to relative 2-D position.

MOVEA3(**x,y,z**)

Move beam to absolute 3-D position.

MOVER3(**dx,dy,dz**)

Move beam to relative 3-D position.

Parameters

x,y,z	Input parameters; absolute user coordinates of endpoint.
dx,dy,dz	Input parameters; user coordinates of endpoint relative to current beam position.

Programming Notes

Calls to MOVExx are not affected by the mode/attribute/feature routines listed in this section. MOVExx is not a primitive in the same sense as other routines in this section since it does not draw anything. However, MOVExx routines are necessary in positioning the drawing beam when constructing displays with the other primitives.

Calls to MOVExx move the drawing beam from its current position to the specified endpoint with the drawing beam off. When a given segment is displayed, calls to MOVExx within it do not appear as lines on the terminal display surface. Drawing beam movement is continuous, and the beam must either be moved with calls to other primitives when display information is involved, or with MOVExx when simple beam repositioning is desired. When a picture is defined (section 3), the drawing beam is initially positioned in the center of the picture. When a segment is opened, the drawing beam is positioned at the last position described in the previous segment. Refer to the examples at the end of this section.

Refer also to WHEREx in section 8.

PLOTxx

Format

PLOTA(**npoint,xary,yary,line**)

Plot symbols at absolute 2-D endpoints.

PLOTR(**npoint,dxary,dyary,line**)

Plot symbols at relative 2-D endpoints.

PLOTA3(**npoint,xary,yary,zary,line**)

Plot symbols at absolute 3-D endpoints.

PLOTR3(**npoint,dxary,dyary,dzary,line**)

Plot symbols at relative 3-D endpoints.

Parameters

npoint	Input parameter; the number of endpoints in each array (same for all arrays).
xary,yary,zary	Input arrays of length **npoint** containing the absolute user coordinates of each endpoint, in the proper sequence.
dxary,dyary,dzary	Input arrays of length **npoint** containing the relative user coordinates of each endpoint, in sequence. The coordinates of each point are given relative to the last point in the array, not relative to the initial beam position.
line	Input logical variable specifying whether or not lines are to be drawn between the plotted points. Applies to all points in the plot. If LINE=.TRUE., lines are drawn between the end points, starting at current beam position. If LINE=.FALSE., no lines are drawn. The symbol is plotted regardless of the value of **line**.

Programming Notes

Calls to PLOTxx are affected by the following subset of the mode/attribute/feature routines described in the mode/attribute/feature subsection.

xxHILT
xxINT
xxSTYL
xxSYM
xxVIS

Modal settings in effect when the first primitive in the segment containing the PLOTxx routine is defined determine the attributes of that segment.

Calls to PLOTxx draw the current modal plotting symbol at each point in a series of endpoints. Refer to xxSYM, in this section, for more information on selecting plotting symbols.

The current modal plotting symbol is drawn at the end of the line segment when PLOTxx is called for one point.

When LINE=.TRUE., line style for the lines connecting the plotted points is determined by the current line style mode as set by SMSTYL.

TEXTx

Format

TEXT(**nchar,itext**)

Draw specified text string at current 2-D position.

TEXT3(**nchar,itext**)

Draw specified text string at current 3-D position.

Parameters

nchar	Input parameter; number of characters in text string.
itext	Input array containing the **nchar** characters to be drawn. The maximum number of characters per word is machine dependent (10 characters per word for CDC CYBER 70, CDC CYBER 170, and 6000 computers).

Programming Notes

Calls to TEXTx are affected by the following subset of the mode/attribute/feature routines described in the mode/attribute/feature subsection.

xxCSIZ
xxDSIZ
xxFONT
xxHILT
xxINT
xxROT
xxROT3
xxVIS

In addition, the programmer may call TFNSIZ to find the number of discrete character sizes supported.

Modal settings in effect when the first primitive in the segment containing the TEXTx call is defined determine the attributes of that segment.

The TEXTx routines draw the specified string of Hollerith characters. The string is drawn beginning with the lower left corner of the first character at current beam position.

At the completion of the TEXTx primitives, the position of the drawing beam is not predictable.

Before numerical information can be output via a TEXTx call, data must be reformatted. The FORTRAN ENCODE statement, although non-ANSI standard, will reformat the data as needed. (Refer to the FORTRAN Reference Manual.)

Refer to xxROT and xxROT3, in this section, for information on character rotation.

MODE/ATTRIBUTE/FEATURE ROUTINES

xxCSIZ

Format

TFCSIZ(**lcchar**)

Test to see if continuous character sizes are supported.

SMCSIZ(**wide,high**)

Modally set the continuous character size.

TMCSIZ(**widout,hiout**)

Test the current modal setting of continuous character size.

RACSIZ(**idseg,wide,high**)

Reset the continuous character size attribute of a segment.

TACSIZ(**idseg,widout,hiout**)

Test the continuous character size attribute of a segment.

Parameters

lcchar	Output parameter; if LCCHAR=.TRUE., postprocessor supports continuous character sizes.
wide,high	Input parameters specifying the width and height of the rectangle that is to contain the character. These measurements are in relative user coordinates and include any spacing for the character.

	DEFAULT is a discrete character size. Refer to xxDSIZ.
widout,hiout	Output parameters specifying the width and height of the rectangle which contains the character.
idseg	Input parameter identifying the segment whose attribute is being tested or reset:
	1≤**idseg**≤32,767

Programming Notes

The set of characters supported by xxCSIZ is shown in table 2-1. Refer to the appropriate postprocessor appendix for any modification/additions to the supported character set for continuous characters.

TABLE 2-1. SUPPORTED CONTINUOUS CHARACTER SET

Code	Graphic	Code	Graphic
0	:	32	5
1	A	33	6
2	B	34	7
3	C	35	8
4	D	36	9
5	E	37	+
6	F	38	-
7	G	39	*
8	H	40	/
9	I	41	(
10	J	42	)
11	K	43	$
12	L	44	=
13	M	45	
14	N	46	,
15	O	47	.
16	P	48	#
17	Q	49	[
18	R	50	]
19	S	51	%
20	T	52	"
21	U	53	—
22	U	54	!
23	W	55	&
24	X	56	'
25	Y	57	?
26	Z	58	<
27	0	59	>
28	1	60	@
29	2	61	\
30	3	62	^
31	4	63	;

Continuous character mode is complementary to discrete mode. Continuous character mode is set by calling SMCSIZ; discrete character mode is set by calling SMDSIZ.

If discrete size is in effect, then **(widout,hiout)** will be returned as (0.,0.) for TMCSIZ and TACSIZ.

If the attribute is discrete, resetting it to continuous is an error.

Continuous character size modes/attributes affect only segments containing text primitives. Modes may be set and attributes altered for segments not containing text primitives, but there is no effect on the segment.

This routine affects the TEXTx routines only; refer to TEXTx routines.

xxDSIZ

Format

TFDSIZ(**wide,high,widout,hiout**)

Test to see what discrete size of characters best approximates a size of character the programmer wishes to use.

SMDSIZ(**wide,high**)

Modally set the discrete character size.

TMDSIZ(**widout,hiout**)

Test current modal setting of discrete character size.

RADSIZ(**idseg,wide,high**)

Reset the discrete character size attribute of a segment.

TADSIZ(**idseg,widout,hiout**)

Test the discrete character size attribute of a segment.

Parameters

wide,high	Input parameters specifying the width and height of the desired character size as fractions of the screen size.
	DEFAULT is the hardware character size most closely approximating:
	WIDE=.0125 HIGH=.0167
	Refer to appropriate postprocessor appendix for more information on default character sizes.
widout,hiout	Output parameters specifying the width and height of the hardware discrete character size that best approximates the desired size (TFDSIZ) or the character size that is being used (TMDSIZ and TADSIZ).
idseg	Input parameter identifying the segment whose attribute is to be tested or reset:
	1≤**idseg**≤32,767

Programming Notes

Discrete character sizes are those character sizes the terminal hardware is capable of generating.

TIGS accepts the values given by the **wide** and **high** parameters and matches them to the terminal-supported discrete size which best approximates the values (SMDSIZ and RADSIZ). Best approximation is defined to be the largest size that is less than or equal to the requested size. The matching is done first on width; within the best approximate width, the best approximate height is found.

If there is no discrete character size less than or equal to the desired size, the smallest available character size is used. To test for the smallest available discrete character size

```
CALL TFDSIZ(0.,0.,WIDOUT,HIOUT)
```

Character sizes for input parameters **wide** and **high** and output parameters **widout** and **hiout** are expressed as fractions of the screen area. For example,

	WIDE=.0125
and	HIGH=.0167

represent 1/80 of the screen horizontally and 1/60 of the screen vertically. This allows 80 characters per line and 60 lines, which is the default. Some terminals support only one character size.

If continuous size is in effect, then (**widout,hiout**) will be returned as (0.,0.) for TMDSIZ and TADSIZ.

If the attribute is continuous, resetting it to discrete is an error.

Discrete character size modes/attributes affect only segments containing text primitives. Modes may be set and attributes altered for segments not containing text primitives, but there is no effect on the segment.

This routine affects the TEXTx routines only; refer to TFNSIZ and TEXTx routines.

xxFONT

Format

TFFONT(**nfont**)

Test the number of character fonts available on a terminal.

SMFONT(**ifont**)

Modally set the character font.

TMFONT(**ifont**)

Test the current modal setting of character font.

RAFONT(**idseg,ifont**)

Reset the character font attribute of a segment.

TAFONT(**idseg,ifont**)

Test the character font attribute of a segment.

Parameters

nfont Output parameter indicating the number of character fonts supported.

ifont Input parameter (SMFONT and RAFONT) or output parameter (TMFONT and TAFONT) specifying character font used.

Range is 0 through 63. If IFONT > NFONT, NFONT is used.

IFONT=1	Use normal font-DEFAULT.
IFONT=2	Use italicized characters if supported.
IFONT=3	Use third font if supported.
.	
.	
.	
IFONT=n	Use nth font if supported.
IFONT=0	Reserved for future expansion.

Number of supported fonts depends on postprocessor.

idseg Input parameter identifying the segment whose attribute is to be tested or reset:

$1 \leq$ **idseg** $\leq 32{,}767$

Programming Notes

Character font modes/attributes affect only segments containing text primitives. Modes may be set and attributes altered for segments not containing text primitives but there is no effect on the segment.

This routine affects the TEXTx routines only; refer to TEXTx routines.

xxHILT

Format

TFHILT(**lhilt**)

Test to see if highlighting of display is supported by postprocessor.

SMHILT(**lhilit**)

Set highlighting mode on or off.

TMHILT(**lhilit**)

Test current highlighting mode.

RAHILT(**idseg,lhilit**)

Reset the highlighting attribute of a segment.

TAHILT(**idseg,lhilit**)

Test the highlighting attribute of a segment.

Parameters

lhilt Output parameter; if LHILT=.TRUE., highlighting is supported; otherwise, it is not.

lhilit Input parameter (SMHILT and RAHILT) or output parameter (TMHILT and TAHILT) indicating highlighting mode or attribute.

If LHILIT=.TRUE., affected segments are highlighted; otherwise, they are not.

DEFAULT is LHILIT=.FALSE.

idseg	Input parameter identifying the segment whose attribute is to be tested or reset: $1 \leq$ **idseg** $\leq 32,767$

Programming Notes

The implementation of highlighting is terminal dependent. It can be done by blinking the item, using a reserved intensity level, line style, color, font (for text), and so on. Refer to the appropriate postprocessor appendix for more information.

The highlighting mode cannot be changed after the first primitive in a segment is defined and for as long as that segment is open.

The method used to echo a segment pick is independent of current highlighting mode. Refer to section 7.

xxINT

Format

TFINT(**ninten**)

Test the number of intensity levels available on a terminal.

SMINT(**finten**)

Modally set the intensity.

TMINT(**finten**)

Test the current modal setting for intensity.

RAINT(**idseg,finten**)

Reset the intensity attribute of a segment.

TAINT(**idseg,finten**)

Test the intensity attribute of a segment.

Parameters

ninten	Output parameter indicating the number of discrete intensity levels supported: $1 \leq$ **ninten** $\leq 32,767$
finten	Input parameter (SMINT and RAINT) or output parameter (TMINT and TAINT) indicating intensity level. Range is from 0. (dimmest) to 1. (brightest). DEFAULT is FINTEN=.5.
idseg	Input parameter identifying segment whose attribute is to be tested or reset: $1 \leq$ **idseg** $\leq 32,767$

xxROT

Format

TFROT(**lninty,lcont**)

Test to see if character rotation by ninety degree increments is supported, and if continuous character rotation is supported.

SMROT(**deg**)

Modally set the angle of 2-D character rotation.

TMROT(**deg**)

Test current modal setting of angle of 2-D character rotation.

RAROT(**idseg,deg**)

Reset the 2-D character rotation attribute of a segment.

TAROT(**idseg,deg**)

Test the 2-D character rotation attribute of a segment.

Parameters

lninty	Output parameter; if LNINTY=.TRUE., terminal hardware supports 90° character rotation.
lcont	Output parameter; if LCONT=.TRUE., continuous character rotation is supported by either terminal hardware or by TIGS software.
deg	Input parameter (SMROT and RAROT) or output parameter (TMROT and TAROT) specifying magnitude of rotation in degrees. Measured counterclockwise from X axis. DEFAULT is no rotation.
idseg	Input parameter identifying the segment whose attribute is to be tested or reset: $1 \leq$ **idseg** $\leq 32,767$

Programming Notes

Character rotation modes/attributes affect only 2-D segments containing text primitives. Modes can be set and attributes altered for segments not containing text primitives, but there is no effect on the segment.

Consult the appropriate postprocessor appendix for specific information on how the character rotation is implemented.

This routine affects the TEXT routine only; refer to TEXT routine.

xxROT3

Format

SMROT3(**xbase,ybase,zbase,xplane,yplane,zplane**)

Modally set the rotation angle and plane for 3-D characters.

TMROT3(**xbase,ybase,zbase,xplane,yplane,zplane**)

Test current modal setting of rotation angle and plane for 3-D characters.

RAROT3(**idseg,xbase,ybase,zbase,xplane,yplane, zplane**)

Reset the 3-D character rotation and plane attribute of a segment.

TAROT3(**idseg,xbase,ybase,zbase,xplane,yplane, zplane**)

Test the 3-D character rotation and plane attribute of a segment.

Parameters

xbase,ybase, zbase	Input parameter (SMROT3 and RAROT3) or output parameter (TMROT3 and TAROT3) specifying a relative vector (base vector) which defines a line along which 3-D text will be written. DEFAULT is (1.,0.,0.).
xplane,yplane, zplane	Input parameter (SMROT3 and RAROT3) or output parameter (TMROT3 and TAROT3) specifying a relative vector (plane vector), which together with the base vector, defines the plane in which the text will be written (that is, the plane vector determines the up direction). DEFAULT is (0.,1.,0.)
idseg	Input parameter identifying the segment whose attribute is to be tested or reset: $1 \leq$ **idseg** $\leq 32{,}767$

Programming Notes

The values of the base and plane vectors are specified in user coordinates relative to the beam position at the time a call to TEXT3 is made.

3-D character rotation angle/plane modes/attributes affect only 3-D segments containing text primitives. Modes can be set and attributes altered for segments not containing text primitives, but there is no effect on the segment.

If the current mode for a 3-D segment is a discrete character size, the 3-D character angle and plane are ignored. Discrete characters will be drawn with no rotation in 3-D segments.

An error occurs if the base vector and the plane vector are colinear (lie along the same line), or if either vector consists of all zeros.

This routine affects the TEXT3 routine only; refer to the TEXT3 routine.

xxSTYL

Format

TFSTYL(**lhard**)

Test for line styles supported by terminal hardware.

SMSTYL(**istyle**)

Modally set line style.

TMSTYL(**istyle**)

Test current modal setting of line style.

RASTYL(**idseg,istyle**)

Reset line style attribute of a segment.

TASTYL(**idseg,istyle**)

Test the line style attribute of a segment.

Parameters

lhard An output array of six logical variables specifying what line styles are supported by terminal hardware.

If LHARD(1)=.TRUE., solid line style is supported.

If LHARD(2)=.TRUE., long dashed line style is supported.

If LHARD(3)=.TRUE., short dashed line style is supported.

If LHARD(4)=.TRUE., dash-dotted line style is supported.

If LHARD(5)=.TRUE., dotted line style is supported.

If LHARD(6)=.TRUE., bit pattern line style is supported.

istyle An input parameter (SMSTYL and RASTYL) or output parameter (TMSTYL and TASTYL) specifying line style.

ISTYLE=1	Solid line style.
ISTYLE=2	Long dashed line style.
ISTYLE=3	Short dashed line style.
ISTYLE=4	Dash-dotted line style.
ISTYLE=5	Dotted line style.
ISTYLE>5	Pattern line style.

$1 \leq$ **istyle** $\leq$ 7777B

DEFAULT is ISTYLE=1, solid line style.

idseg An input parameter identifying the segment whose attribute is to be tested or reset:

$1 \leq$ **idseg** $\leq 32{,}767$

Programming Notes

xxSTYL routines do not affect text primitives.

As the value for **istyle** increases to 5, the percentage of space in the line increases.

istyle values above 5 are used for patterned line styles which allow the programmer to create his own line styles for special purposes. For patterned line styles, the **istyle** value is given as a four-figure octal number. Numbers of fewer than four figures are treated as right-justified, zero-filled quantities. This octal number is translated into its 12-bit binary equivalent which becomes the line style pattern. Each 0 value is a drawing beam off segment and each 1 value is a beam on segment. This 12-bit pattern is repeated as many times as is necessary, depending on the line length. For example,

ISTYLE=7652B

yields a 12-bit pattern

111110101010

which is interpreted as a long dash followed by three dots

—— . . .

and becomes the pattern for all lines drawn, until changed.

Consult the appropriate postprocessor appendix for information on the algorithm used to transfer this bit pattern line style to the terminal display screen.

xxSYM

Format

TFSYM(**nsym**)

Test for maximum defined symbol number in a given postprocessor.

SMSYM(**isym**)

Modally set the selected plotting symbol for use with PLOTxx routines.

TMSYM(**isym**)

Test current modal setting for plot symbol.

RASYM(**idseg,isym**)

Reset the plotting symbol attribute of a segment.

TASYM(**idseg,isym**)

Test the plotting symbol attribute of a segment.

Parameters

nsym Output parameter indicating maximum symbol number for which a symbol is defined:

-1≤**nsym**≤32,767

NSYM=-1 implies no symbol is defined.

isym Input parameter (SMSYM and RASYM) or output parameter (TMSYM and TASYM) specifying symbol used. Refer to appropriate postprocessor appendix for supported symbol set.

DEFAULT is ISYM=-1; no symbol defined. This option allows the plotting of arrays of data without symbols.

If **isym** > **nsym**, **nsym** is used.

idseg Input parameter identifying the segment whose attribute is to be tested or reset:

1≤**idseg**≤32,767

Programming Notes

TIGS supports the Calcomp/UNIPLOT compatible set of centered symbols shown in table 2-2.

TABLE 2-2. SUPPORTED PLOTTING SYMBOLS

Code	Graphic
0	□
1	○
2	△
3	+
4	×
5	◇
6	↑
7	⊠
8	Z
9	Y
10	⌑
11	✳
12	⧖
13	|

Refer to the appropriate postprocessor appendix for any modifications/additions to this set.

This subroutine affects the PLOTxx routines only; refei to PLOTxx.

xxVIS

Format

TFVIS(**ltran**)

Test to see if complete screen retransmission is required to make any one segment invisible.

SMVIS(**lvis**)

Set visibility mode for visible or invisible.

TMVIS(**lvis**)

Test current setting of visibility mode.

RAVIS(**idseg,lvis**)

Reset visibility attribute of a segment.

TAVIS(**idseg,lvis**)

Test visibility attribute of a segment.

arameters

ltran	Output parameter; if LTRAN=.TRUE., then retransmission of data to the screen is necessary to make a visible segment invisible.
lvis	Input parameter (SMVIS and RAVIS) or output parameter (TMVIS and TAVIS) indicating visibility mode or attribute. If LVIS=.TRUE., affected segments are visible; otherwise, they are not. Visibility mode may not be changed during segment definition; the entire segment is either visible or not visible. DEFAULT is LVIS=.TRUE.
idseg	Input parameter identifying the segment whose attribute is to be tested or reset: 1≤**idseg**≤32,767

rogramming Notes

he effects of a reset visibility operation will typically e seen immediately on a refresh terminal.

FNSIZ

ormat

TFNSIZ(**nsize**)

Test to see how many discrete character sizes are supported by the terminal.

Parameters

nsize	Output parameter giving number of discrete character sizes supported.

Programming Notes

This routine affects the TEXTx routines only; refer to TFDSIZ and TEXTx routines.

EXAMPLE OF GRAPHICS PRIMITIVES USAGE

The following example illustrates the use of many of the graphics primitives covered in this section. Many of the mode/attribute/feature routines which affect primitives are used. The example is not a complete program; it illustrates only the postprocessor-independent portions of a TIGS program. TIGS calls that are not documented in this section (for example, INITIG and QUITIG) may be found elsewhere in the manual.

Figure 2-4 illustrates the display generated by this example.

The sample program at the end of section 4 contains an elementary example of three-dimensional primitives usage.

WARNING

Refer to appropriate postprocessor and operating system appendices for format of PROGRAM statement.

```
      PROGRAM statement
      DIMENSION LHARD(6),X(21),Y(21)
      INTEGER TMSTYL
      LOGICAL LHARD,LINE
      CALL INITIG(.TRUE.,.TRUE.,5LIFILE)      Initialize TIGS; refer to section 8.
      CALL TFSTYL(LHARD)                   }
      DO 100 I=1,4                         }
      J=6-I                                }  Find out which line styles are hardware-supported. If dotted,
      IF (LHARD(J)) GO TO 150              }  dash-dotted, short dashed or long dashed lines are hardware-
  100 CONTINUE                             }  supported, set line style parameter to the appropriate one
      ISTYLE=1                             }  (first hardware-supported line style encountered is used).
      GO TO 200                            }  Otherwise, set line style parameter to solid line. This is a
  150 ISTYLE=J                             }  good example of postprocessor-independent programming.
  200 CONTINUE                             }  Set mode, line style. Note that this is the line style attribute
      CALL SMSTYL(ISTYLE)                  }  for the entire segment even though the line style for this seg-
                                              ment does not remain constant.
      CALL SMPLIM(-100.,-100.,100.,100.)      Set picture limits; refer to section 3.
      CALL OPNSEG(1)                          Open segment 1; refer to section 3.
      CALL MOVEA(75.,-50.)
      THETA=180.                                    Draw arc in lower right quadrant.
      CALL ARCDA(50.,-50.,THETA)
      CALL MOVEA(-25.,-50.)                         Draw circle in lower left quadrant.
      CALL ARCA(-50.,-50.,-25.,-50.)
      CALL MOVEA(-55.,-40.)
      CALL TEXT(4,4HTIGS)
      CALL MOVEA(-57.,-60.)                         Add text; default character size is used.
      CALL TEXT(7,7HEXAMPLE)
      CALL MOVEA(44.,-50.)
      CALL TEXT(6,6HY=X**2)
      CALL MOVEA(-100.,0.)
      CALL DRAWR(200.,0.)                           Draw X and Y axes using relative draw routines.
      CALL MOVER(-100.,100.)
      CALL DRAWR(0.,-200.)
      VAL=-10.
      DO 300 I=1,21
      X(I)=VAL                                      Set up X and Y arrays for plotting points.
      Y(I)=VAL**2
      VAL=VAL+1
  300 CONTINUE
      CALL SMSYM(12)
      LINE=.TRUE.                                   Set mode for special plotting symbol number 12 (hourglass
      CALL MOVEA(X(1),Y(1))                         shape) and set for connecting lines between points.
      CALL PLOTA(21,X(1),Y(1),LINE)
      CALL MOVEA(-65.,90.)
      CALL TEXT(9,9H(-10,100))                      Label 1st and 21st points; default character size is used.
      CALL MOVEA(55.,90.)
      CALL TEXT(8,8H(10,100))
      JSTYLE=5
      IF (TMSTYL(ISTYLE).EQ.5) JSTYLE=3
      CALL SMSTYL(JSTYLE)
      CALL MOVEA(54.,92.)
      CALL DRAWA(12.,99.)                           Add arrows to point from text to graph. Use dotted line style
      CALL SMSTYL(1)                                if it was not used to draw the arcs; otherwise, use short
      CALL DRAWR(0.,-1.5)                           dashes. Use function subprogram to test the line style used for
      CALL DRAWR(1.5,1.5)                           arc drawing. Use relative draw routine to draw arrowheads.
      CALL DRAWR(-1.5,0.)
```

```
CALL SMSTYL(JSTYLE)
CALL MOVEA(-46.,92.)
CALL DRAWA(-12.,99.)
CALL SMSTYL(1)                     Draw second arrow and arrowhead.
CALL DRAWR(0.,-1.5)
CALL DRAWR(-1.5,1.5)
CALL DRAWR(1.5,0.)
CALL CLSSEG                        Close segment; refer to section 3.
CALL DSPLAY                        Display picture; refer to section 8.
CALL QUITIG(.TRUE.)                Terminate TIGS program; refer to section 8.
STOP
END
```

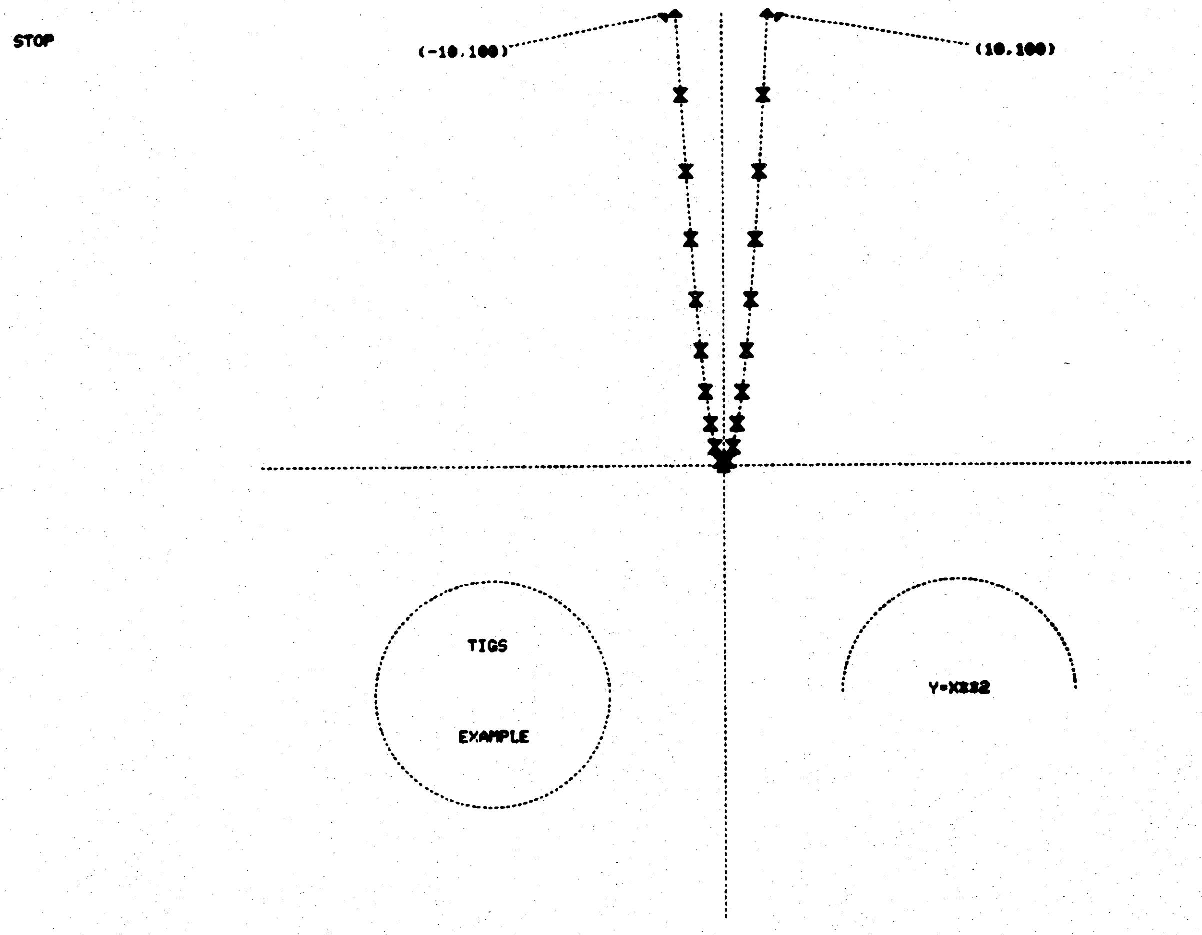


Figure 2-4. Sample Primitives Program Display

The following TIGS calls are documented in this section.

BLINDS	LCKSEG	TAPLIM
CLSPIC	OPNPIC	TAPLI3
CLSSEG	OPNSEG	TA3D
COPY	RAPICT	TFPICT
DELPIC	RENAME	TMPICT
DELSEG	SMPICT	TMPLIM
EMPTY	SMPLIM	TMPLI3
EXTPIC	SMPLI3	TM3D
EXTSEG	TAPICT	

This list includes routines which define and manipulate segments and pictures, and those mode/attribute/feature routines which affect segments and pictures. The mode/attribute/feature routines are placed in a separate group following the calls which define and manipulate segments and pictures.

Following the description of the routines is an example in which segment and picture usage is demonstrated.

GENERAL

To aid the programmer in effectively organizing a graphics display, TIGS has a three-level organizational scheme. From least comprehensive to most comprehensive, these three levels are primitives, segments, and pictures. This scheme allows effective planning of a display: how that display will ultimately appear on the terminal viewing area is also dependent upon the window and viewport calls described in section 4 of this manual.

A segment consists of a group of one or more graphics primitives. The programmer defines a segment through the use of the OPNSEG and CLSSEG subroutines, and assigns each segment an identification number. Segments, like pictures, are employed for purposes of selective display modification. By using these organizational tools, a programmer can manipulate and modify certain portions of a total display without affecting other portions. Segments compose the lowest organizational level at which such modification can take place; that is, no modification can be done on a primitive-by-primitive basis unless each primitive is a separate segment. TIGS operations such as translation, deletion, and interactive picking act on individual segments through proper specification of segment identifiers.

A segment possesses attributes. Attributes such as intensity, line style, and sensitivity to interactive picks are automatically inherited by the segment from the modal settings in effect when the segment is opened.

One or more segments compose a picture. The programmer defines a picture with the OPNPIC and CLSPIC calls and assigns a segment to a picture with either the OPNPIC call or the SMPICT or EXTPIC calls (they perform identical functions), whichever is encountered latest. A segment can belong to only one picture at a time. For example, the sequence of calls

```
CALL OPNPIC(5)

CALL SMPICT(18)

CALL OPNSEG(2)
```

would result in segment 2 belonging to picture 18. If the OPNPIC and SMPICT calls were reversed, the segment would belong to picture 5.

Only one picture can be open at any given point in the program. The same is true for segments. Once opened and then closed, a segment cannot be reopened with OPNSEG. It can only be extended using the EXTSEG routine. A picture cannot be reopened using OPNPIC, but SMPICT or EXTPIC may be used to assign new segments to a previously defined picture. If SMPICT is used with a picture identification number not previously defined with an OPNPIC call, a warning message is issued and a picture with that identification number is opened.

The dimensionality of a picture (2-D or 3-D) is established when picture limits are specified. In TIGS, picture limits and dimensionality are modally set values. A call to SMPLIM is made for two-dimensional pictures, and a call to SMPLI3 is made for three-dimensional pictures. Only two-dimensional primitives, windows, and viewports can be used with two-dimensional pictures. Only three-dimensional primitives, windows, and viewports can be used with three-dimensional pictures.

Default program conditions provide for situations in which a programmer might not need TIGS multiple picture capability. Segments may be defined without defining a picture in which to place them. In this case, all segments belong to a single default picture (picture ID=0). The default picture can be either two- or three-dimensional, but not both. When using the default picture, no calls to OPNPIC, SMPICT, or EXTPIC should be made. Multiple pictures are useful where a programmer is composing displays from different sources (for example, several different engineering drawings) or needs flexibility in combining display elements.

Only when the default picture is used can the default window and viewport be used. Any calls to OPNPIC, SMPICT, or EXTPIC necessitate parallel calls to window and viewport routines to display the graphics material (refer to Windows and Viewports, section 4).

The following paragraphs give capsule descriptions of the segment and picture routines in this section.

The definition of a picture or segment is accomplished with the OPNPIC and CLSPIC or OPNSEG and CLSSEG routines. LCKSEG may be used to close a segment that is not to be extended.

Primitives can be added to existing segments by using the EXTSEG routine. New segments may be added to existing pictures using the EXTPIC routine.

Existing segments are copied by using the COPY routine.

EMPTY removes all primitives from an existing segment.

Specific pictures and segments are deleted by using the DELPIC and DELSEG subroutine.

Picture visibility is controlled by BLINDS.

The RENAME subroutine is used to give a segment a different identification.

The xxPICT mode/attribute/feature routines control assignment of segments to pictures.

The xxPLIx mode/attribute/feature routines define picture limits.

The xx3D mode/attribute routines test picture dimensionality.

SEGMENT AND PICTURE ROUTINES

BLINDS

Format

BLINDS(**idpict,ldown**)

Control display of all parts of picture through all windows with which the picture is associated.

Parameters

idpict	Input parameter specifying the desired picture.
ldown	Input parameter specifying whether or not the picture will be seen in associated windows. If LDOWN=.TRUE., the blinds are down and no part of the picture is seen in any window. If LDOWN=.FALSE., the blinds are up and the picture is visible in all associated windows. DEFAULT is LDOWN=.FALSE.

Programming Notes

BLINDS controls picture visiblity by controlling whether or not the windows of a given picture are mapped to their assigned viewports on the terminal display screen. Refer to Windows and Viewports, section 4, for further information.

To use BLINDS in the default picture, window, and viewport situation, use IDPICT=0.

CLSPIC

Format

CLSPIC

Close an open picture.

Parameters

None.

Programming Notes

Together with OPNPIC, CLSPIC is used to define the beginning and end of a picture composed of segments. CLSPIC will close any open picture. CLSPIC also closes any open segment.

Refer to OPNPIC.

CLSSEG

Format

CLSSEG

Define the end of the currently open segment.

Parameters

None.

Programming Notes

Together with OPNSEG, CLSSEG is used to define the beginning and end of a segment composed of primitives. CLSSEG will close any open segment.

Refer to OPNSEG.

COPY

Format

COPY(**idseg,newseg**)

Generate a copy of the specified segment and assign it the segment identifier **newseg.**

Parameters

idseg	Input parameter specifying the identifier of the segment to be copied. If the segment does not exist or is open, an error occurs and the call is ignored.

newseg	Input parameter specifying the identifier to be assigned to the copy: 1≤**newseg**≤32,767 If this ID is already is use, an error occurs and the call is ignored.

Programming Notes

The currently open picture does not influence the COPY operation. The copy of the segment is inserted in the same picture as the original. To put the copy in a different picture, use the RAPICT routine.

OPNSEG and CLSSEG are not used to define the beginning and end of a copied segment.

DELPIC

Format

DELPIC(**idpict**)

Delete specified picture.

Parameters

idpict	Input parameter specifying the ID of the picture that is to be deleted.

Programming Notes

Deleting a picture also deletes all windows and all segments in that picture. The default picture cannot be deleted. An error occurs if an attempt to do so is made. A warning message is issued if an attempt is made to delete a nonexistent picture (**idpict** does not exist).

An **idpict** which has been deleted is reusable.

DELSEG

Format

DELSEG(**idseg**)

Delete specified segment.

Parameters

idseg	Input parameter specifying the ID of the segment that is to be deleted.

Programming Notes

An open segment cannot be deleted. A warning message is issued if an attempt is made to delete a nonexistent segment (**idseg** does not exist).

An **idseg** which has been deleted is reusable.

The associated picture does not need to be open for the DELSEG operation.

The effects of the delete operation will be seen immediately on a refresh terminal.

EMPTY

Format

EMPTY(**idseg**)

Delete contents of a segment.

Parameters

idseg	Input parameter specifying the ID of the segment that is to be emptied.

Programming Notes

This subroutine deletes all primitives from a segment. It also restores modal settings to the values that prevailed when the segment was opened if modes were reset within the segment definition. All attributes of the segment (inherited from modal settings in effect when the segment was originally opened) remain as they were.

The associated picture does not need to be open for the EMPTY operation.

The effects of the EMPTY operation will be seen immediately on a refresh terminal.

Refer to EXTSEG.

EXTPIC

Format

EXTPIC(**idpict**)

Add segments to an existing picture.

Parameters

idpict	Input parameter specifying the ID of the picture to be extended.

Programming Notes

This subroutine reopens an existing picture for extension. An attempt to reopen an existing picture using OPNPIC is an error. All segments created while the picture is open belong to the picture. Pictures reopened with EXTPIC must be closed. Calls to CLSPIC, EXTPIC, SMPICT, and OPNPIC will close an open picture.

Picture limits and picture dimensionality associated with **idpict** become the current modal settings when EXTPIC is called. That is, if a new picture is defined after a call to EXTPIC, it will inherit its dimensionality and limits from picture **idpict** unless changed by the programmer.

After a call to EXTPIC, beam position is the same as it was when picture **idpict** was last closed.

If no picture with identifier **idpict** exists, one is created and a warning message is issued.

This subroutine is identical to SMPICT.

EXTSEG

Format

EXTSEG(**idseg**)

Add primitives to an existing segment.

Parameters

idseg	Input parameter specifying the ID of the segment to be extended.

Programming Notes

This subroutine reopens an existing segment for extension. An attempt to reopen an existing segment using OPNSEG is an error. All primitives specified while the segment is reopened belong to the segment. Segments reopened with EXTSEG must be closed again. Calls to CLSSEG, OPNSEG, LCKSEG, or EXTSEG close any open segment; however, segments closed by OPNSEG and EXTSEG result in warning messages.

Segments closed with LCKSEG cannot be reopened for extension.

After a call to EXTSEG, all modal settings in effect when segment **idseg** was closed become the current modal settings. For example, a call to EXTSEG results in a resetting of current picture mode to the picture which contains segment **idseg**. If segment 3 is in picture 1, the sequence

```
CALL SMPICT(9)

CALL EXTSEG(3)
```

means that picture 1, not picture 9, is the current picture modal setting for subsequent operations, unless changed.

Current beam position is reset to where it was when segment **idseg** was last closed.

If a segment **idseg** does not exist, such a segment is created and a warning message is issued.

LCKSEG

Format

LCKSEG

Lock the currently open segment.

Parameters

None.

Programming Notes

This subroutine permanently closes an open segment so that the segment can never be extended. When the programmer is certain that a segment will not be extended, it is more efficient to use LCKSEG than CLSSEG to close a segment because CLSSEG must store all modal settings in effect when the segment is closed.

OPNPIC

Format

OPNPIC(**idpict**)

Define the beginning of a picture.

Parameters

idpict	Input parameter specifying ID of new picture: $1 \leq$ **idpict** $\leq 32{,}767$

Programming Notes

OPNPIC defines the beginning of picture **idpict**. Subsequent segments are placed in picture **idpict** until another picture is specified. All segments are placed in the currently open picture.

If **idpict** has previously been used, a warning message is issued and the OPNPIC call is ignored.

If a picture is open when the OPNPIC call is made, it is closed before picture **idpict** is opened.

If there is no open picture, segments are placed in the default picture, IDPICT=0.

The only attributes of a picture are dimensionality (two- or three-dimensional) and picture limits. If nondefault picture limits are to be used, they must be set by using SMPLIM for two-dimensional pictures or SMPLI3 for three-dimensional pictures, and they must be set before any TIGS call which describes coordinates in the picture. These attributes cannot be reset. Refer to SMPLIM and SMPLI3 in this section.

Current beam position is moved to the center of the new picture after a call to OPNPIC.

When using nondefault pictures defined by OPNPIC calls, the default picture should not be used.

Refer to CLSPIC.

OPNSEG

Format

OPNSEG(**idseg**)

Define the beginning of a segment.

Parameters

idseg	Input parameter specifying ID of new segment: 1≤**idseg**≤32,767

Programming Notes

OPNSEG defines the beginning of segment **idseg**. Subsequent primitives are placed in segment **idseg** until another segment is specified. Primitives are always placed in the currently open segment.

If **idseg** has previously been used, a warning message is issued and the OPNSEG call is ignored.

If a segment is open when the OPNSEG call is made, it is closed before segment **idseg** is opened and a warning message is issued. Open segments should be closed with CLSSEG.

If primitives are encountered when no segment is open, a warning message is issued and the default segment IDSEG=0 is opened or extended as needed.

Segments inherit attributes from modal settings in effect when the segment is defined. Because modes do not become attributes of a segment until the first primitive of that segment is encountered, modes may be set before or after the OPNSEG call. Since a segment as a whole can logically possess only one value for any of its attributes, some care must be exercised when setting modes if attributes are later to be reset. Refer to Mode/Attribute/Feature Routines in this section for more information.

RENAME

Format

RENAME(**idold,idnew**)

Replace old segment identifier **idold** with new identifier **idnew**.

Parameters

idold	Input parameter; segment ID of segment which is to be renamed: 1≤**idold**≤32,767
idnew	Input parameter; new ID of segment: 1≤**idnew**≤32,767

Programming Notes

idold is as established by OPNSEG.

The associated picture does not need to be open for the RENAME operation.

MODE/ATTRIBUTE/FEATURE ROUTINES

Certain rules must be observed in the use of mode/attribute/feature calls which affect segments and pictures. Modes described in this section can be set at any point in a program except as noted under SMPICT, SMPLIM, and SMPLI3. However, if a modal setting is changed in the middle of a segment definition, a subsequent attempt to reset the attribute represented by that modal setting will yield unpredictable results, since a segment as a whole can have only one value for a given attribute. Thus, for any given segment, a programmer can:

- Change modes within a segment definition, or
- Subsequently reset the attribute of that segment,

but not both.

Segment attributes are not established until the first primitive is defined for that segment; until that time, modes may be changed without involving that segment in the situation described above.

All set mode subroutines described below, except SMPLI3, have defaults. A programmer who is content with default conditions need not call any of the mode setting routines.

xxPICT

Format

TFPICT(**npict**)

Test for maximum number of pictures postprocessor supports.

SMPICT(**idpict**)

Modally set the already-existing picture to which subsequent segments are added.

TMPICT(**idpict**)

Test for ID of current mode set picture.

RAPICT(**idseg,idpict**)

Reset the picture attribute of a segment.

TAPICT(**idseg,idpict**)

Test the picture attribute of a segment.

Parameters

npict	Output parameter indicating the maximum number of pictures supported by the postprocessor: 1≤**npict**≤32,767
idpict	Input parameter (SMPICT and RAPICT) or output parameter (TMPICT and TAPICT) specifying a picture identifier. For SMPICT and RAPICT: 1≤**idpict**≤32,767 For TMPICT and TAPICT: 0≤**idpict**≤32,767
idseg	Input parameter identifying the segment whose attribute is to be tested or reset.

Programming Notes

If no call to SMPICT is made, all segments are placed in a single default picture, IDPICT=0. However, if more than one picture is used, the default picture should not be used. When IDPICT=0, SMPICT and RAPICT calls produce errors.

Default window and viewport assignment is made only when the default window is used. Calls to SMPICT necessitate parallel calls to window and viewport routines to display the graphics material (refer to section 4).

SMPICT should not be called when a segment is open. A call to SMPICT closes the currently open picture. This also closes the currently open segment.

For SMPICT, if no picture **idpict** exists, one is created and a warning message is issued. For RAPICT, the call is ignored and an error message is issued.

SMPICT is identical to the EXTPIC subroutine. Refer to EXTPIC in this section for more information.

When moving a segment from one picture to another with RAPICT, the programmer should be aware that if the picture limits of the new picture are not equal to the picture limits of the old picture, improper scaling may result, and identification of points in the segment may be incorrect.

xxPLIx

Format

SMPLIM(**xll,yll,xur,yur**)

Modally set limits for 2-D picture coordinates.

TMPLIM(**xll,yll,xur,yur**)

Test the limits of the current modally-set 2-D picture.

TAPLIM(**idseg,xll,yll,xur,yur**)

Test the limits of the 2-D picture which contains the specified segment.

SMPLI3(**xllh,yllh,zllh,xury,yury,zury**)

Modally set limits for 3-D picture coordinates.

TMPLI3(**xllh,yllh,zllh,xury,yury,zury**)

Test the limits of the current modally-set 3-D picture.

TAPLI3(**idseg,xllh,yllh,zllh,xury,yury,zury**)

Test the limits of the 3-D picture which contains the specified segment.

Parameters

xll,yll,xur,yur	Input parameters (SMPLIM) or output parameters (TMPLIM and TAPLIM) specifying the user coordinates of the lower left and upper right corners of the picture. DEFAULT is: **xll=yll**=0. **xur=yur**=1.
xllh,yllh,zllh, xury,yury,zury	Input parameters (SMPLI3) or output parameters (TMPLI3 and TAPLI3) specifying the user coordinates of the lower left hither and upper right yon corners of the picture (refer to figure 3-1), such that lower< upper, left<right and hither< yon, although these terms are not necessarily meaningful for any observer position. The default picture (IDPICT=0) can be 2-D or 3-D, but not both. Default picture limits for 2-D pictures are given above. There are no default picture limits for 3-D; a programmer using the default picture must still establish 3-D picture limits by calling SMPLI3.
idseg	Input parameter identifying the segment whose attribute is to be tested.

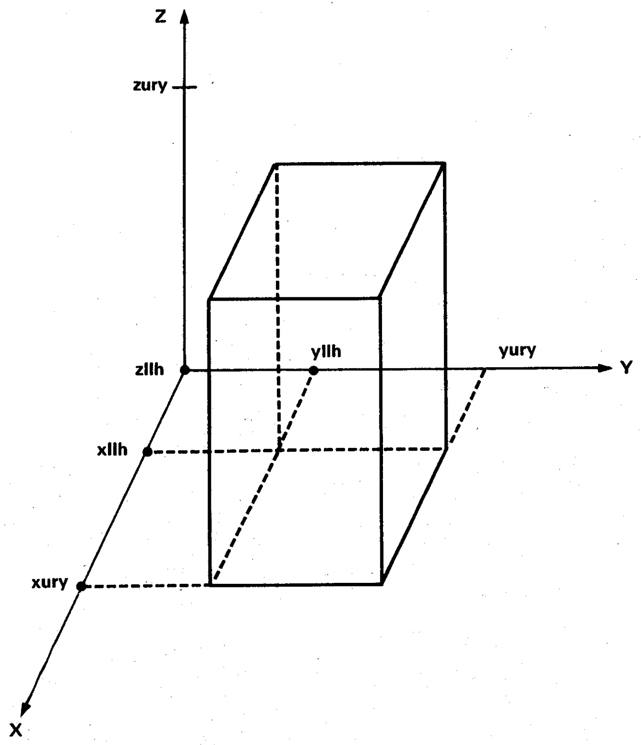


Figure 3-1. Three-Dimensional Picture Limits

Programming Notes

There are no test feature (TF) or reset attribute (RA) routines for two- or three-dimensional picture limits. TF routines are not applicable. RA routines are omitted because of the great complexity and potential for misinterpretation of such procedures.

Picture dimensionality (2-D or 3-D) is established by the selection of SMPLIM for two-dimensional pictures and SMPLI3 for three-dimensional pictures. If neither routine is selected, the picture will be two-dimensional with default picture limits. Either two- or three-dimensional picture limits can be in effect for a given picture, but not both. Any attempt to modify the picture limits of a picture, as when extending an existing picture, is an error.

Nondefault picture limits can be specified for the default picture; use of SMPLIM does not necessitate the use of OPNPIC. There are no default picture limits for three-dimensional pictures.

Picture limits must be declared before any routine is called in which user coordinates are specified; among these routines are primitives routines, window routines, the LOCATE subroutine, and transformation mode-setting routines. To be sure of unambiguously establishing desired picture limits, it is strongly recommended that if a picture is to have nondefault limits, these limits should be set immediately following the OPNPIC call.

As in the case of the window and viewport routines:

xll<xur

yll<yur

xllh<xury

yllh<yury

zllh<zury

The default 3-D viewing system is described in the windows and viewports chapter.

xx3D

Format

TM3D(**l3D**)

Test current mode for picture dimensionality.

TA3D(**idseg,l3D**)

Test dimensionality attribute of given segment.

Parameters

l3D	Output parameter; if L3D=.TRUE., the current modal setting or attribute is 3-D; otherwise, it is 2-D.
idseg	Input parameter identifying the segment whose attribute is to be tested.

Programming Notes

Test feature (TF), set mode (SM), and reset attribute (RA) routines for picture dimensionality do not exist. A TF routine is not applicable. The dimensionality of a picture or segment, once established by SMPLIM or SMPLI3, cannot be altered. Therefore, there are no SM or RA routines. Refer to SMPLIM and SMPLI3.

EXAMPLE OF SEGMENTS AND PICTURE USAGE

The following example uses many of the segment and picture calls documented in this section. It is not a complete program. Postprocessor-dependent portions of the program are not illustrated, nor are portions dealing with topics not covered in this section, such as the calls to graphics primitives subroutines and window and viewport routines. These topics are dealt with elsewhere in the manual.

The example uses three separate pictures, one of which is three-dimensional.

WARNING

Refer to appropriate postprocessor and operating system appendices for format of PROGRAM statement.

PROGRAM statement	
CALL INITIG (.TRUE.,.TRUE.,5LIFILE)	
CALL OPNPIC(1)	Open first picture, 2-D.
CALL OPNSEG(1)	
.	
.	
.	
2-D primitives	Open, define, and close first segment so it cannot be extended.
.	
.	
.	
CALL LCKSEG	
CALL OPNSEG(2)	
.	
.	
.	
2-D primitives	Open, define, and close second segment so it cannot be extended.
.	
.	
.	
CALL LCKSEG	
CALL CLSPIC	Close first picture.
CALL OPNPIC(2)	Open second picture, 3-D.
CALL SMPLI3(-100.,-100.,-100.,100.,100.,100.)	Set picture limits; there are no default 3-D picture limits. Picture limits should be established immediately after the picture is opened.
CALL OPNSEG(33)	
CALL SMSTYL(2)	
.	
.	
.	
3-D primitives	Open, define, and close 3-D segment; line style is long dashed lines, beam position at end is (100,100,100).
.	
.	
.	
CALL DRAWA3(100.,100.,100.)	
CALL CLSSEG	
CALL OPNPIC(3)	Open third picture; 3-D is current picture dimensionality until SMPLIM changes it to 2-D.
CALL SMPLIM(0.,0.,1.,1.)	
CALL COPY(1,11)	Copy segment 1 into new segment 11.
CALL RAPICT(11,3)	Even though picture 3 was the open picture when the segment was copied, the copy is in the original picture unless RAPICT is called.
CALL RAPICT(2,3)	Move segment 2 to picture 3.
CALL DELPIC(1)	Delete picture 1.

```
CALL EXTSEG(33)

CALL TMPICT(IDPICT)

IF(IDPICT.NE.2)STOP 66
```

Extend segment 33; all modes that were in effect when the segment was closed become current modes. The call to TMPICT and the conditional statement demonstrate that current picture modal setting has changed from picture 3 to picture 2; picture 3 is automatically closed. Current beam position is where the beam was left when segment 33 was closed, at (100,100,100). Line style is long dashed line (ISTYLE=2).

```
CALL SMSTYL(3)
    .
    .
    .
3-D primitives
    .
    .
    .
CALL CLSSEG
    .
    .
    .
Calls to window and viewport routines,
display routines, and so on.
```

Set line style mode to short dashed lines; line style attribute of segment 33 remains ISTYLE=2, long dashed. Since the segment now contains multiple modal settings for line style, the line style attribute of the segment can no longer be predictably reset.

WINDOWS AND VIEWPORTS

The following TIGS calls are documented in this section.

DELVUP	TMPORT	WINDOW
DELWIN	TMSVP	WINPER
SMPORT	VUPORT	WINPLN
SMSVP	VUPOR3	WINSIZ
TFPORT	WINCLP	WINUP
TFSVP	WINDIR	

This list includes routines which define and manipulate windows and viewports, and those mode/attribute/feature routines which affect windows and viewports. The mode/attribute/feature routines are placed in a separate group following the calls which define and manipulate windows and viewports.

Following the routines is an example in which window and viewport usage is demonstrated.

The programmer composes a graphics model through calls to primitives, segments, and picture routines. He must then determine how the model is to be viewed on the screen of the graphics terminal by using window and viewport subroutines. These window and viewport subroutines are described in this section.

A window is a subset of the picture with which it is associated. A viewport is a subset of the display screen. By means of window subroutines, the programmer controls what parts of the picture he has composed are displayed, and by means of viewport subroutines, where those parts are displayed on the terminal display screen. Window is to picture as viewport is to display screen. This relationship is illustrated in figure 4-1.

Viewports, like pictures, are modally set. Windows are not modally set; a new window must be defined each time one is required in the program.

A default window and viewport are provided for situations which do not require multiple windows and viewports. No calls to window or viewport routines are necessary to use the default window and viewport. However, the following restrictions on default window and viewport usage apply.

- The default window and viewport should not be used if nondefault pictures have been defined. Only the default picture can be displayed through the default window and viewport. A program should not contain calls to OPNPIC, SMPICT, or EXTPIC if the default window and viewport are to be used. However, nondefault windows and viewports can be used with the default picture.

- Nondefault windows cannot be used with the default viewport and vice versa.

- A program using the default window and viewport can draw either two- or three-dimensional pictures, but not both, because the default picture cannot be both two- and three-dimensional. For three-dimensional pictures, the programmer must specify the dimensionality of the picture by calling SMPLI3. A program using the default window and viewport for a two-dimensional picture need not declare picture dimensionality; however, the programmer can specify nondefault picture limits for the default picture by calling SMPLIM.

In the default case, the window includes the entire default picture; the window limits are the same as the picture limits. The default viewport covers the entire usable screen (refer to INITIG, section 8, for more information).

For nondefault windows and viewports, a window call defines what portion of the current modally set picture is displayed in the current modally set viewport. Thus, the series of calls

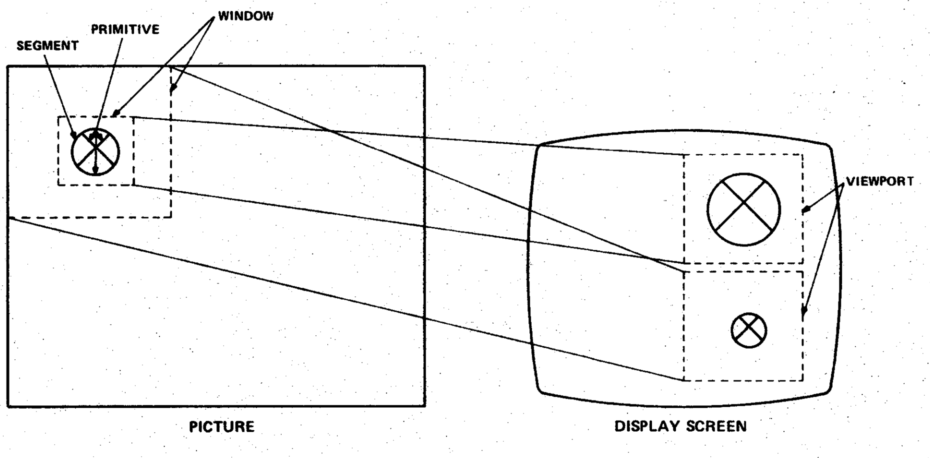


Figure 4-1. Relationship of TIGS Elements

```
CALL OPNPIC(1)

CALL SMPORT(5)

CALL WINDOW(63,XLL,YLL,XUR,YUR)
```

will result in assigning window 63 of picture 1 to viewport 5. In this example we assume that viewport 5 has already been defined, as it must be before it can be used.

If another window is defined without changing the modally set picture or viewport, two windows of the same picture will appear in the viewport, the second window superimposed on the first. Or, if the current picture is changed before the second window is defined

```
CALL OPNPIC(1)

CALL SMPORT(5)

CALL WINDOW(63,XLL,YLL,XUR,YUR)

CALL SMPICT(3)

CALL WINDOW(66,XLL,YLL,XUR,YUR)
```

window 66 of picture 3 will be displayed in viewport 5, on top of window 63 of picture 1. This technique is useful when composing a display from multiple sources to be used in various combinations, like overlay engineering drawings.

Remember that OPNPIC, SMPICT, EXTPIC, and EXTSEG all change the current modally set picture.

When the modally set viewport is changed, subsequent window calls are associated with that viewport. In this way it is possible to simultaneously display different views of the same object.

A number of useful effects are possible by properly manipulating window and viewport calls. If a window is defined at the picture center and the window limits are changed symmetrically with respect to the picture center, the picture can be zoomed in or out. Zooming in makes items appear larger, zooming out makes items appear smaller. For three-dimensional pictures, this effect can also be achieved by varying the position of the projection plane (refer to WINPLN).

A panning effect can be achieved by maintaining a constant window size but changing the window location with respect to the picture center, similar in effect to panning a movie camera over a scene. For three-dimensional pictures, the effect of circling around an object can be achieved by changing the eye position relative to the object (refer to WINDIR).

In all cases, if the window X:Y or X:Y:Z ratio is not equal to the viewport X:Y or X:Y:Z ratio, distortion will result because the scale factors of the display will not be equal. Some postprocessors may not be able to support this distortion.

Only two-dimensional windows and viewports can be used with two-dimensional pictures. Only three-dimensional windows and viewports can be used with three-dimensional pictures.

A given window can appear in only one viewport.

Following are capsule descriptions of the window and viewport routines covered in this section.

WINDOW defines two-dimensional windows. WINDIR, WINSIZ, WINPLN, WINUP, WINPER, and WINCLP define three-dimensional windows.

VUPORT and VUPOR3 define two- and three-dimensional viewports and viewport limits.

DELWIN and DELVUP delete the specified window or viewport.

The xxPORT mode/attribute/feature routines control viewport modal settings.

The xxSVP mode/attribute/feature routines control system viewport modal settings.

WINDOW AND VIEWPORT ROUTINES

DELVUP

Format

DELVUP(**idport**)

Delete the specified viewport.

Parameters

idport	Input parameter specifying ID of viewport to be deleted.

Programming Notes

When the specified viewport is deleted, all associated windows are also deleted. If the current modally set viewport is deleted, the default viewport is used (IDPORT=0).

Deleting the default viewport or a nonexistent viewport is an error; the DELVUP call is ignored.

An **idport** which has been deleted is reusable.

DELWIN

Format

DELWIN(**idwind**)

Delete the specified window.

Parameters

idwind	Input parameter specifying the ID of the window to be deleted.

Programming Notes

When the specified window is deleted, the portion of the associated picture within the window limits is no longer visible in the viewport. The picture itself and any other windows associated with it are unaffected.

If all window definitions for the default picture (IDPICT=0) are deleted, the default window is used.

Deleting the default window or a nonexistent window is an error; the DELWIN call is ignored.

An **idwind** which has been deleted is reusable.

VUPORx

Format

VUPORT(**idport,xll,yll,xur,yur**)

Define 2-D viewport ID and limits.

VUPOR3(**idport,xllh,yllh,zllh,xury,yury,zury**)

Define 3-D viewport ID and limits.

Parameters

idport	Input parameter specifying ID of viewport: $1 \leq$ **idport** $\leq 32{,}767$
xll,yll,xur,yur	Input parameters specifying screen coordinates of the lower left and upper right corners of the 2-D viewport. Refer to the following programming notes for defaults.
xllh,yllh,zllh, xury,yury,zury	Input parameters specifying the screen coordinates of the lower left hither and upper right yon corners of the viewport space (figure 4-2).

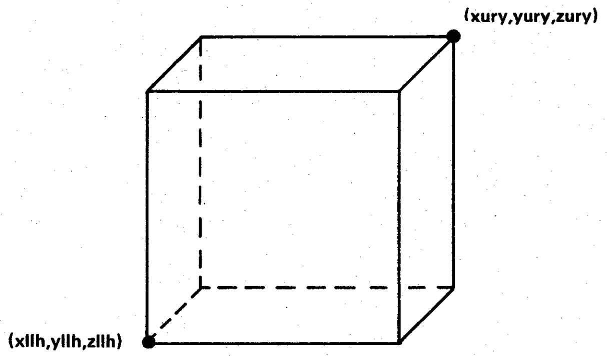


Figure 4-2. Three-Dimensional Viewport

Programming Notes

Viewport limits are supplied in terms of screen, not user, coordinates. The range of these coordinates depends upon the value for the **lsquar** parameter of the INITIG subroutine. If LSQUAR=.TRUE., the coordinate system ranges from (0.,0.) at the lower left to (1.,1.) at the upper right for a two-dimensional display. For a three-dimensional display, the values are (0.,0.,0.) for the lower left hither corner to (1.,1.,1.) at the upper right yon corner. All viewports must lie within these bounds. If LSQUAR= .FALSE., screen coordinates are dependent upon the postprocessor used. Refer to INITIG, section 8, and the appropriate postprocessor appendix for more information.

If no viewports are specified, a single default viewport covering all the usable screen is used. This is true for LSQUAR=.TRUE. and LSQUAR=.FALSE.. However, the default viewport can only be used with the default picture and window. In addition, the default picture, window, and viewport should not be used in any program which uses nondefault pictures, windows, and viewports.

Viewports can be overlapped and superimposed on the screen. For a three-dimensional display, depth overlapping is also possible: viewports may be defined behind other viewports.

Depth of a three-dimensional viewport is also used to implement depth queueing for postprocessors which support this feature. Depth queueing is the process of varying line intensity according to the distance from the front or hither plane defining the viewport.

As is the case in the window and picture limits routines:

xll<**xur**

yll<**yur**

xllh<**xury**

yllh<**yury**

zllh<**zury**

A viewport specification for a given **idport** may be redefined without affecting the modally set viewport.

WINxxx

Format

WINDOW(**idwind,xll,yll,xur,yur**)

Define 2-D window ID and limits.

WINCLP(**idwind,lclpnr,lclpfr,disner,disfar**)

Define 3-D window clipping.

WINDIR(**idwind,xeye,yeye,zeye,xat,yat,zat**)

Define 3-D line of vision.

WINPER(**idwind,lpersp**)

Define 3-D window perspective.

WINPLN(**idwind,distat**)

Define 3-D projection plane.

WINSIZ(**idwind,width,height**)

Define 3-D window size.

WINUP(**idwind,dxup,dyup,dzup**)

Define 3-D window up direction.

Parameters

idwind Input parameter specifying ID of window:

$1 \leq$ **idwind** $\leq 32{,}767$

xll,yll,xur,yur Input parameters specifying the user coordinates of the lower left and upper right corners of a 2-D window. The conditions

xll<xur

yll<yur

must be satisfied.

DEFAULT is picture limits; default applicable only to default picture (IDPICT=0).

lclpnr,lclpfr Input parameters specifying clipping planes; if LCLPNR=.TRUE., clipping is done to the near clipping plane; otherwise, it is not. If LCLPFR=.TRUE., clipping is done to the far clipping plane; otherwise, it is not.

DEFAULTs are:

LCLPNR=.FALSE.

LCLPFR=.FALSE.

disner,disfar Input parameters specifying clipping plane distances from the center of attention (refer to WINDIR). If LCLPNR=.FALSE., **disner** is meaningless; if LCLPFR=.FALSE., **disfar** is meaningless. Otherwise, these parameters specify the directed distance along the line of vision from the center of attention to the near and far clipping planes. Positive distances are away from the eye; negative distances are toward the eye. There are no defaults because the overall defaults for WINCLP specify no clipping at all.

xeye,yeye,zeye Input parameters specifying the coordinates of a point to be used as the viewer's eye position.

DEFAULTs are:

XEYE = **xury+(xury-xllh)** (refer to SMPLI3, section 3).

YEYE = center of y-axis picture limits.

ZEYE = center of z-axis picture limits.

xat,yat,zat Input parameters specifying the coordinates of the center of attention. This point and the eye position point define a line of vision.

DEFAULT is the center of the picture limits.

lpersp Input parameters specifying 3-D perspective type. If LPERSP=.TRUE., the window has a perspective projection; if LPERSP=.FALSE., the window has a parallel (axonometric) projection.

DEFAULT is LPERSP=.TRUE.

distat Input parameter specifying the directed distance from the center of attention to the projection plane. Positive distances are away from the eye, negative distances are toward the eye.

DEFAULT is DISTAT=0.

width Input parameter specifying the width of the viewing window on the projection plane, centered about the line of vision and perpendicular to the up direction (refer to WINUP).

DEFAULT is the width of the picture (refer to the following programming notes).

height Input parameter specifying the height of the viewing window on the projection plane, centered about the line of vision and parallel to the up direction (refer to WINUP).

DEFAULT is the height of the picture (refer to the following programming notes).

dxup,dyup,dzup Input parameters specifying the coordinates of a point relative to the center of attention. The directed line segment from the center of attention to this point defines an up direction for the window; this line segment becomes the vertical axis when the window is mapped to the viewport.

DEFAULT is:

DXUP=0.
DYUP=0.
DZUP=1.

making the up direction parallel to the positive z axis.

Programming Notes

Window subroutines define window limits which are applied to the current modally set (currently open) picture to control what is displayed in the current modally set viewport. All coordinates and distances are specified in user coordinates, established when picture limits were defined with SMPLIM or SMPLI3 or by default. Coordinate values outside the established range are permissible. For example, a three-dimensional eye position of (-1,-1,-1) where the picture ranges from (0,0,0) to (1,1,1) will be correctly interpreted. However, primitives defined outside of the picture limits are clipped when the default window is used.

A default window is provided for use with the default picture (IDPICT=0). Window limits are the same as the picture limits so that the entire default picture appears in the window. However, the default window can only be used with the default picture and viewport. In addition, the default picture, window, and viewport should not be used in any program which uses nondefault pictures, windows, and viewports. Refer to the General subsection in this section.

The TIGS default 3-D viewing system is right-handed, with Z as the default up direction. The TIGS user is required to define the 3-D picture limits subject to the constraints that "lower, left, hither" coordinates must be smaller than the "upper, right, yon" coordinates as was described in the Segments and Picture chapter. The default attention point is the center of the picture space and the default eye position places the eye outside of the picture space, on the positive x side at a distance from the box equal to the box width in the x direction. Figure 4-3 depicts the default situation. Note that the terms upper, lower, left, right and hither, yon are not necessarily meaningful with respect to the default (observer) eye position; rather, they are chosen strictly so that lower< upper, left< right and hither <yon.

A window specification for a given **idwind** may be redefined. This redefinition will be reflected in the next update of the display.

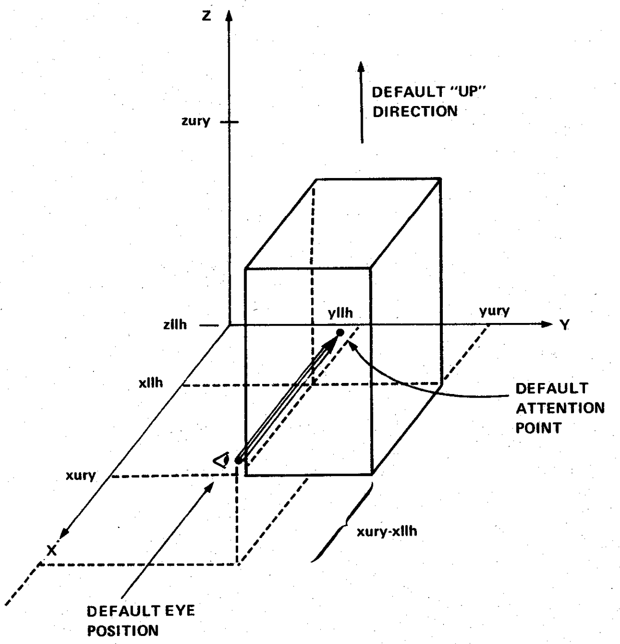


Figure 4-3. Default 3-D Viewing System

The two-dimensional window is defined in the same way as the two-dimensional picture; the parameters give the coordinates of the lower left and upper right corners of a rectangle which is a subset of the picture.

The three-dimensional window is also a subset of the three-dimensional picture but its specification is more complicated. The three-dimensional window is initially defined by combining the information given in three-dimensional window subroutine calls made in the program with default values for any three-dimensional window subroutine not called. Any subsequent window calls for that window identification modify the window as it is already defined. For example, the program sequence

```
CALL WINUP(1,0.,1.,0.)

CALL WINPER(1,.FALSE.)
```

accepts all three-dimensional window defaults except for the window up direction and the perspective default. For this window, the up direction will be in the direction of the positive y-axis and the projection will be axonometric (nonperspective, refer to WINPER description and to the example at the end of this section for more information). All other characteristics are as defaulted:

- Default line of vision, as defined by eye position, and center of attention.
- Default window covering the entire picture.
- Default projection plane located at the center of attention.
- No near or far clipping planes so that the window depth is the same as the picture depth.

The window can be modified later in the program, if desired:

```
CALL WINDIR(1,1.,1.,1.,.5,.5,.5)
```

This call changes the eye position while leaving the center of attention unchanged, resulting in a new above-and-to-the-right viewing position for a picture with limits of (0,0,0) to (1,1,1).

All other window specifications remain: a nondefault up direction and perspective specification, and all else as defaulted.

A programmer can accept all default specifications for a nondefault window. However, one call to a window routine must be made to establish the window ID. For example, the call

```
WINPER(12,.TRUE.)
```

establishes window 12 with all defaults, since .TRUE. is the default value for the WINPER routine.

No call to any window routine is necessary to use the default window (IDWIND=0) with the default picture and viewport.

Window proportions must match the viewport proportions, or distortion will result. For three-dimensional windows, window width and height and the picture depth must be proportionate to the three-dimensional viewport X:Y:Z ratio. The distance between clipping planes need not be proportional to the three-dimensional viewport depth.

Each of the three-dimensional window routines, WINCLP, WINDIR, WINPER, WINPLN, WINSIZ, and WINUP, controls a different aspect of the three-dimensional window. Each routine is examined separately in the following paragraphs. Figures 4-4 and 4-5 show the frustum of vision defined by these routines.

WINCLP – This subroutine specifies depth clipping for the three-dimensional window. The programmer can specify near and/or far clipping planes and their distances from the center of attention defined in WINDIR. Default conditions call for the entire front-to-back extent of the three-dimensional picture space to be included in the window. When the near clipping plane is defined,

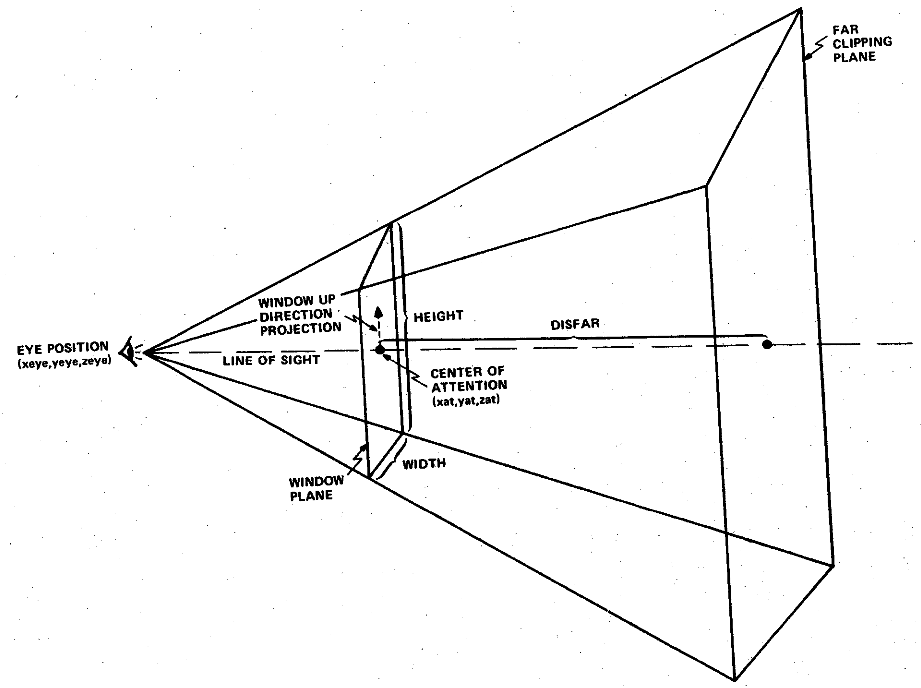

FAR CLIPPING PLANE
WINDOW UP DIRECTION PROJECTION
HEIGHT
DISFAR
EYE POSITION (xeye,yeye,zeye)
LINE OF SIGHT
CENTER OF ATTENTION (xat,yat,zat)
WINDOW PLANE
WIDTH

Figure 4-4. The Frustum of Vision for Three-Dimensional Perspective Window

everything lying in picture space between the plane and the eye position is deleted. When the far clipping plane is defined, everything lying in picture space beyond the plane is deleted. There is an implied near clipping plane at the eye position; anything lying behind the eye position in picture space is never included in a three-dimensional window. Both clipping planes must be in front of the eye position, and the near plane must be closer to the eye than the far plane. Clipping planes are always parallel to each other and perpendicular to the line of sight.

WINDIR – WINDIR defines a line of sight through picture space by describing a viewer's position and the direction in which he is looking. The line of sight is a directed line segment between the eye position **(xeye,yeye,zeye)** and the center of attention **(xat,yat,zat)**. There are no restrictions on this direction; the programmer is not constrained to any particular orientation toward picture space or to a line of vision parallel to one of the major axes. This line of sight is the line used for locating all major window planes (clipping planes and projection plane); all planes must be perpendicular to the line of sight and are specified as located a directed distance from the point defining the center of attention (figure 4-4).

When the **lpersp** parameter of WINPER has a value of .FALSE., the two points defining the line of sight do little more than give a viewing direction in space. When LPERSP=.TRUE., however, wide and narrow fields of vision can be simulated by the proper choice of eye position and center of attention points. When the eye position and center of attention points are chosen to lie relatively close together, a panoramic or fish-eye lens view is obtained. Lines drawn from the eye position through the corners of the window plane include a greater angle the closer the eye is to the window plane (refer to WINSIZ). This gives a wider field of view, just as a person sees more of the interior of a building the closer he stands to a window. Effects of perspective are also enhanced by placing the eye position closer to the window plane. When the wider angle of the frustum of vision thus produced is mapped to the three-dimensional viewport, which is a parallelepiped, objects nearer the eye are reduced in size less than objects farther from the eye (refer to WINPER).

Tunnel vision can be simulated by placing the eye position relatively farther away from the window plane. Perspective is also reduced thereby.

WINPER – WINPER specifies whether the window uses a perspective or axonometric projection. LPERSP=.TRUE. preserves perspective; objects farther away in picture space appear smaller than objects which are closer to eye position. When LPERSP=.FALSE., an axonometric view of everything contained in the window is obtained. That is, objects of equal size appear equal sized regardless of relative distance away from the eye; there is no perspective in an axonometric window. When the window up direction and line of sight are chosen so that the z-axis projects toward the lower left corner of the screen, the axonometric projection looks like the isometric projections used in engineering drafting. Figure 4-5 illustrates an axonometric projection; note that it is a rectangular parallelepiped, not a truncated pyramid like figure 4-4.

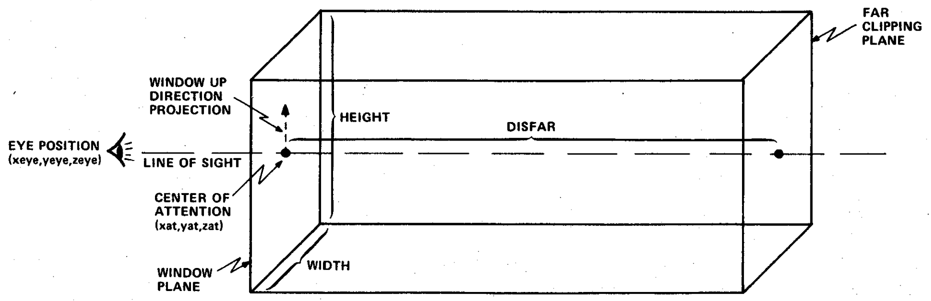

FAR CLIPPING PLANE
WINDOW UP DIRECTION PROJECTION
HEIGHT
DISFAR
EYE POSITION (xeye,yeye,zeye)
LINE OF SIGHT
CENTER OF ATTENTION (xat,yat,zat)
WINDOW PLANE
WIDTH

Figure 4-5. Parallelepiped of Vision for Three-Dimensional Axonometric Window

WINPLN – This subroutine specifies the distance of the projection plane from the center of attention. Default is a projection plane at the center of attention, making the projection plane identical with the window plane. When LPERSP=.FALSE. (refer to WINPER), WINPLN has no effect on the display. When LPERSP=.TRUE., varying the distance of the plane from the eye alters the size of the viewed objects. When **distat** values are positive, the projection plane moves away from the eye and objects get larger. When **distat** values are negative, the plane moves closer to the eye and objects get smaller (figure 4-6). The display is still clipped to the window defined in WINSIZ or as defaulted. The projection plane must be located in front of the eye position.

WINSIZ – This subroutine specifies the size of the window to be used by defining an aperture on the window plane. The window plane is the plane which is perpendicular to the line of sight at the center of attention. This subroutine defines a subset of this plane. The **width** parameter specifies the width of the viewing window centered at the center of attention and perpendicular to the window up direction (refer to WINUP). The **height** parameter specifies the height of the viewing window centered at the center of attention and parallel to the window up direction. Because of the way the default three-dimensional coordinate system is oriented (refer to WINUP), the programmer should use this routine when it is necessary to tailor a window to a non-square viewport. He should not attempt to define a non-square picture to match the viewport and then use the default window. Refer to the example at the end of this section.

WINUP – WINUP establishes an up direction to the window and gives meaning to the **width** and **height** parameters of the WINSIZ routine. The parameters define a directed line segment from the center of attention. When the window is mapped to a viewport, the projection of this line segment in the window plane is oriented vertically in the viewport. It is important to remember two things about this mapping:

- Because a projection of the line segment onto the window plane is used, the line segment cannot be colinear with the line of sight; if it were colinear, there would be no projection.

- This subroutine can only rotate a basic display around the center of attention. Because it is the projection of the line segment that is made perpendicular and not the line segment itself, the basic orientation of the viewer to the viewed object(s) does not change. The eye position and center of attention must be changed to alter the basic orientation.

In the default case, the positive z-axis is used as the window up direction.

MODE/ATTRIBUTE/FEATURE ROUTINES

xxPORT

Format

TFPORT(**nport**)

Test for maximum number of viewports supported by a postprocessor.

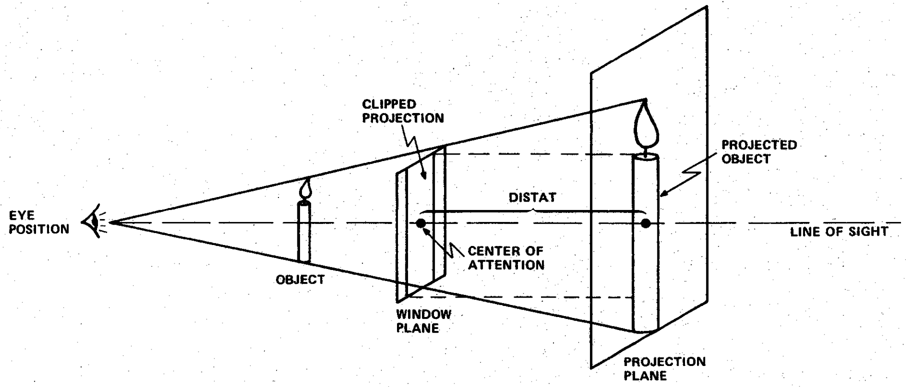


Figure 4-6. Projection Plane

SMPORT(**idport**)

Modally set viewport into which subsequent windows are mapped.

TMPORT(**idport**)

Test for current modally set viewport.

Parameters

nport	Output parameter specifying the maximum number of viewports the postprocessor supports: $1 \leq$ **nport** $\leq 32{,}767$
idport	Input parameter (SMPORT) or output parameter (TMPORT) specifying a viewport identifier as established by a call to the VUPORT or VUPOR3 routines.

Programming Notes

There are no reset attribute (RA) or test attribute (TA) routines because a segment, as part of a picture, can appear in more than one viewport at a time.

Like pictures, viewports are modally set. Thus, after a viewport is defined with VUPORT or VUPOR3, it must be made the current modally set viewport before windows can be assigned to it. Windows are always assigned to the current modally set viewport. In a program which mixes two- and three-dimensional pictures, the programmer should take care that the current modally set viewport is appropriate for the type of window being defined.

The default case provides for a single viewport covering the entire usable screen to be used only with the default picture and window. This viewport is either two- or three-dimensional, depending on the dimensionality of the default picture. No calls to SMPORT or VUPORx are necessary to use the default viewport. Refer to VUPORx in this section.

xxSVP

Format

TFSVP(**lsysvp**)

Test for a default system viewport separate from grahpics area.

SMSVP(**lscren,xll,yll,xur,yur**)

Modally set a viewport to be used for system messages and user entries.

TMSVP(**lscren,xll,yll,xur,yur**)

Test for viewport used for system messages and user entries.

Parameters

lsysvp	Output parameter; if LSYSVP=.TRUE., the postprocessor supports a default system viewport that is guaranteed separate from the area of the screen used for graphics displays. If LSYSVP=.FALSE., the default viewport is not guaranteed to be separate. Dependent on **lsquar** value given in INITIG routine.
lscren	Input parameter (SMSVP) or output parameter (TMSVP). If LSCREN=.TRUE., system viewport coordinates are given in a terminal independent coordinate system (refer to the following programming notes). If LSCREN=.FALSE., system viewport coordinates are given in terminal dependent coordinate system.
xll,yll,xur,yur	Input parameters (SMSVP) or output parameters (TMSVP) specifying the lower left and upper right corners of the screen area used as a system viewport; possible values are dependent on **lscren** value.

Programming Notes

There are no reset attribute (RA) or test attribute (TA) routines because the system viewport cannot become an attribute of a segment.

The system viewport is the area of the terminal display screen to be used for messages from and operator responses to the PROMPT and KEYBRD interactive routines (refer to section 7). Screen placement of operating system messages, such as FORTRAN diagnostics, is not controlled by TIGS; these messages are not affected by system viewport placement. The programmer can place the system viewport in any portion of the screen, depending on the value of the **lscren** parameter.

There is always a default system viewport defined for programmers who do not wish to define their own.

However, it is not always guaranteed to be separate from the graphics display area. The placement of the default system viewport is postprocessor dependent; refer to the appropriate postprocessor appendix.

The INITIG routine interacts with the system viewport routines in the following ways (refer to INITIG, section 8).

- If the INITIG **lsquar** parameter is set to LSQUAR=.FALSE. during initiation, the **lsysvp** parameter of TFSVP can never be .TRUE.. A separate area for a system viewport is dependent on LSQUAR=.TRUE..

- If the SMSVP **lscren** parameter is set to LSCREN=.FALSE., the programmer can take advantage of the separate area made available by setting LSQUAR=.TRUE. in a call to INITIG. If LSCREN=.TRUE., the system viewport must be located within the graphics display area. In the absence of other considerations, it is recommended that LSCREN=.FALSE. be specified so that the graphics display area is not overwritten by messages in the system viewport area.

Terminal independent screen coordinates range from (0.,0.) in the lower left corner of the usable screen area to (1.,1.) in the upper right corner of the usable screen. Terminal dependent screen coordinates vary from postprocessor to postprocessor; refer to the appropriate postprocessor appendix.

EXAMPLE OF WINDOW AND VIEWPORT USAGE

The example in figure 4-7 uses many of the window and viewport calls documented in this section. This is not a complete program; postprocessor-dependent portions of the program are not illustrated.

WARNING

Refer to appropriate postprocessor and operating system appendices for format of PROGRAM card.

This program combines two- and three-dimensional capabilities to illustrate differences in two- and three-dimensional window and viewport usage. The program uses one two-dimensional picture, window, and viewport, one three-dimensional picture, and two three-dimensional windows and viewports. A message is placed in the system viewport area to illustrate its positioning. Figure 4-8 illustrates how the final displav looks on the terminal screen.

PROGRAM statement	
DIMENSION ITEXT(4)	
DATA ITEXT/32HTHIS IS THE SYSTEM VIEWPORT AREA/	Establish array for use with PROMPT.
LOGICAL LSYSVP	
CALL INITIG(.TRUE.,.TRUE.,5LIFILE)	TIGS is initialized with **lsquar** parameter of INITIG set to .TRUE.. A value of .FALSE. precludes the possiblility of a default system viewport separate from the graphics display area.
CALL TFSVP(LSYSVP) IF(.NOT.LSYSVP) STOP 77	Test for default system viewport separate from display area; if not separate, the programmer must make other arrangements (not shown in this program) to divide the screen between system and other viewports.
CALL OPNPIC(1)	Define first picture; 2-D by default with default picture limits.
CALL VUPORT(1,0.,0.,.333,1.)	Define 2-D viewport covering first third of graphics display area of screen.
CALL SMPORT(1)	Make viewport 1 the current modally set viewport.
CALL OPNSEG(1) CALL MOVEA(.05,.4) CALL DRAWA(.25,.4) CALL DRAWA(.25,.6) CALL DRAWA(.05,.6) CALL DRAWA(.05,.4) CALL CLSSEG	Define 2-D box.
CALL WINDOW(1,0.,0.,.333,1.)	Create window on portion of picture containing 2-D box to appear in current modally set viewport; ratio of width to height is the same for window and viewport to avoid distortion.
CALL OPNPIC(2) CALL SMPLI3(0.,0.,0.,1.,1.,1.)	Define 3-D picture; SMPLI3 specifies it as 3-D.
CALL VUPOR3(2,.333,0.,0.,.666,1.,1.)	Define 3-D viewport covering second third of graphics display area.
CALL SMPORT(2)	Make viewport 2 the current modally set viewport.

Figure 4-7. Sample Program with Viewport end Window Calls

```
CALL OPNSEG(2)
CALL MOVEA3(.4,.05,.4)
CALL DRAWA3(.6,.05,.4)
CALL DRAWA3(.6,.25,.4)
CALL DRAWA3(.4,.25,.4)
CALL DRAWA3(.4,.25,.6)
CALL DRAWA3(.6,.25,.6)
CALL DRAWA3(.6,.05,.6)
CALL DRAWA3(.4,.05,.6)
CALL DRAWA3(.4,.05,.4)
CALL DRAWA3(.4,.25,.4)
CALL MOVEA3(.6,.05,.4)
CALL DRAWA3(.6,.05,.6)

CALL MOVEA3(.6,.25,.4)
CALL DRAWA3(.6,.25,.6)
CALL MOVEA3(.4,.05,.6)
CALL DRAWA3(.4,.25,.6)
CALL CLSSEG
```

Define 3-D box.

```
CALL WINPER(2,.FALSE.)
CALL WINSIZ(2,.333,1.)
CALL WINDIR(2,1.,.27,.7,.5,.167,.5)
```

Create axonometric 3-D window, a subset of current modally set 3-D picture, to appear in current modally set 3-D viewport; distortion is avoided by ensuring that window and viewport width, height, and depth are proportional. Line of sight is chosen to give an above-and-to-the-right viewing position. Balance of window characteristics are as defaulted.

```
CALL VUPOR3(3,.666,0.,0.,1.,1.,1.)
```

Define second 3-D viewport covering last third of graphics display area.

```
CALL SMPORT(3)
```

Make viewport 3 the current modally set viewport.

```
CALL WINPER(3,.TRUE.)
CALL WINSIZ(3,.333,1.)
CALL WINDIR(3,1.,.27,.7,.5,.167,.5)
```

Create 3-D window (with depth perspective) as subset of current modally set 3-D picture, to appear in current modally set 3-D viewport. All other characteristics are the same as in window 2.

```
CALL PROMPT(32,ITEXT)
    .
    .
    .
```

Use PROMPT routine (section 7) to display message in default system viewport area.

Figure 4-7. Sample Program with Viewport and Window Calls (Contd)

THIS IS THE SYSTEM VIEWPORT AR
EA

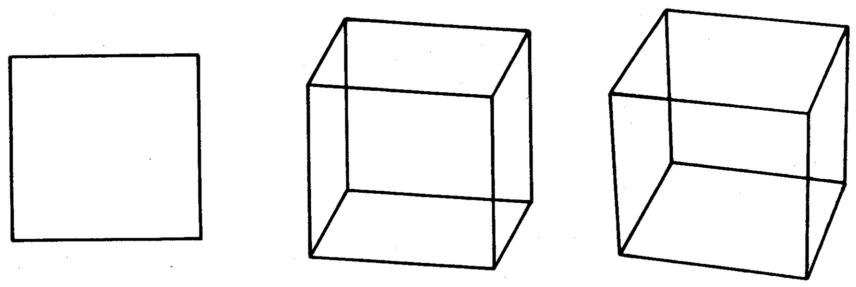


Figure 4-8. Sample Viewport and Window Usage Display

TRANSFORMATION MATRICES

The following TIGS calls are documented in this section.

CLRSTK	SMXFL3	TMXFA	XROTR3
CLRST3	SMXFR	TMXFA3	XSCLA
POP	SMXFR3	TMXFL	XSCLA3
POP3	TAXFA	TMXFL3	XSCLL
PUSH	TAXFA3	TMXFR	XSCLL3
PUSH3	TAXFL	TMXFR3	XSCLR
RAXFA	TAXFL3	XIDNT	XSCLR3
RAXFA3	TAXFR	XIDNT3	XTRNA
RAXFL	TAXFR3	XINVR	XTRNA3
RAXFL3	TFXFA	XINVR3	XTRNL
RAXFR	TFXFA3	XROTA	XTRNL3
RAXFR3	TFXFL	XROTA3	XTRNR
SMXFA	TFXFL3	XROTL	XTRNR3
SMXFA3	TFXFR	XROTL3	
SMXFL	TFXFR3	XROTR	

This list includes routines which perform matrix manipulations, and the mode/attribute/feature routines which affect transformation matrix operations. The mode/attribute/feature routines are placed in a separate group following the matrix building and manipulation routines.

Following the routine descriptions are examples of transformation matrix usage.

GENERAL

With the transformation matrix routines, the TIGS programmer can alter his two- or three-dimensional display by translating, rotating, and scaling it up or down. In addition, TIGS allows the programmer to supply his own matrix to the matrix multiplication routines for other effects.

There are two steps in using TIGS transformation routines. First, the required matrix is built using matrix utility routines. This group of routines includes XIDNTx, XINVRx, XROTxx, XSCLxx, and XTRNxx. Second, subroutines from the xxXFxx group are selected to effect the desired transformation.

Transformation matrices are modally set, and affect all segments defined while a given transformation is in effect. Each segment receives a transformation attribute from the modally set transformation. This transformation attribute can subsequently be changed with attribute resetting calls which specify a new transformation matrix.

The current modally-set transformation is called the current transformation matrix (CTM)†. This is a central concept of TIGS transformation capability. In the default case, the CTM is the identity matrix. No translation, rotation, or scaling effects are produced by the identity matrix: segments appear as originally defined.

The CTM is changed each time a call is made to a transformation matrix mode-setting subroutine (SMXFxx). The programmer has a choice of absolute or relative setting of a new CTM. What the terms absolute and relative mean in the context of transformation subroutines is as follows: the absolute mode-setting subroutines specify that the supplied matrix is to be used as is for the transformation operations through simple replacement of the old CTM; the relative mode-setting subroutines specify that the supplied matrix is to be concatenated (multiplied) with the CTM, with the resultant matrix becoming the new CTM. In other words:

- Absolute routines imply no matrix concatenation.
- Relative routines imply matrix concatenation, and hence, cumulative transformations.

Absolute mode-setting subroutines have -A suffixes for two-dimensional transformations and -A3 suffixes for three-dimensional transformations, such as SMXFA and SMXFA3. Relative mode-setting routines have -R and -L suffixes for two-dimensional transformations and -R3 and -L3 suffixes for three-dimensional transformations, such as SMXFR and SMXFL3. The -R and -L suffixes imply right and left side matrix multiplication, respectively. The significance of each type of multiplication is discussed later in this section.

The situation is analogous for the transformation attribute-resetting subroutines (RAXFxx), except that the supplied matrix is concatenated with the transformation matrix already associated with the segment, which is not necessarily the CTM. Again, the programmer has a choice of using left or right side matrix multiplication when using relative routines, as, for example, RAXFL and RAXFR3.

Absolute and relative have no meaning for the test mode (TMXFxx) and test attribute (TAXFxx) routines; absolute and relative routines are supplied here for consistency.

Remember the following things about absolute versus relative transformation routine usage.

- For an initial transformation to be applied to a segment, an absolute routine is used. The programmer could get the same effect by using a relative routine which concatenates the desired transformation matrix with the identity matrix.
- If cumulative transformations are desired, the relative routines are used.
- If each transformation of a series of transformations is to be applied to the segment as the segment was originally defined, the absolute routines are used.

† CTM3 for three-dimensional transformations; collectively referred to as CTM, except where distinguished for clarity.

Illustrations of cumulative transformations are found in the examples at the end of this section.

When the CTM is changed, the programmer has the option of saving the old CTM for later use. This is accomplished by calling the PUSHx subroutine which places a copy of the CTM on a stack. The CTM can then be changed without losing the old matrix. When the old matrix is needed, a call to POPx fetches the old matrix from the stack and establishes it as the CTM for subsequent transformations. CTMs are collected in the stack on a first-in, last-out basis; if six calls to PUSHx were made in creating the stack, six calls to POPx must be made to reestablish the original matrix as CTM.

The programmer should keep in mind that EXTSEG restores all modes that were in effect when the segment to be extended was closed. This means the CTM in effect when the segment was closed becomes the new CTM when EXTSEG is called.

Before transformations are done, the matrix to be supplied to the mode/attribute routines must be built. This can be done by the programmer or by TIGS transformation utility routines. If done by the programmer, the matrix is supplied in a 2 x 3 array (for two-dimensional pictures) or 3 x 4 array (for three-dimensional pictures), which is then specified by the mode/attribute routines. To help in constructing these matrices, the mathematical basis for TIGS transformation routines is described in appendix D.

TIGS transformation utility routines will construct transformation matrices for translating, rotating, and scaling displays, or a combination of any of the preceding. A transformation building matrix (the B matrix) is constructed by these utilities. The programmer again has the option of using absolute or relative utility calls. Absolute utility routines, like XTRNA, replace the current B matrix with the matrix built by the XTRNA call. Relative utility routines, like XTRNR or XTRNL, supply a matrix which is concatenated with the current B matrix to form a new B matrix for cumulative transformations. In its turn, this B matrix can be specified in a mode/attribute call, where it either replaces the CTM or current segment transformation matrix in absolute mode/attribute calls, or it is concatenated with the CTM or current segment transformation matrix to form a new matrix in relative mode/attribute calls, as discussed previously. Refer to the examples at the end of this section for more information.

Any transformation operation which involves matrix concatenation can specify that the multiplication take place on either the right or left side. This is true of the matrix utility subroutines which build matrices, such as XTRNR and XTRNL; the mode-setting subroutines such as SMXFR3 and SMXFL3; and the attribute-resetting subroutines such as RAXFR and RAXFL. The mathematical details of this process are contained in appendix D. It is the purpose of this introduction only to guide the programmer in choosing the matrix multiplication procedure appropriate to his needs. Failure to choose the correct one may have unexpected results.

As a guide to the proper choice of relative transformation routines, it is useful to consider what the effect of each step in a cumulative transformation process is for both the left and right types of multiplication routines.

The left type, like SMXFL and RAXFL3, implies that each successive step in a cumulative transformation series occurs after the preceding steps. If a programmer has a series of transformation calls for rotating, translating, and scaling a segment, the segment will be first, rotated; second, translated; and third, scaled. Each new step occurs after the previous step has been applied.

The right multiplication type, like SMXFR and RAXFR3, implies that each successive step in a cumulative transformation series occurs before the preceding steps. If the series previously mentioned for rotating, translating, and scaling a segment employs right multiplication routines, the segment will be first, scaled; second, translated; and third, rotated. Each new step occurs before the previous step has been applied. The right multiplication routines invert the series order given in the program.

In summary, left multiplication routines mean that each step in a cumulative transformation series occurs after the preceding step; right multiplication routines mean that each step occurs before the preceding step. Refer to appendix D for more information.

The before/after effects of the matrix multiplication routines have important implications for the type of graphics tasks the left and right types of routines are suited for. Simply stated, the left side multiplication routines are used in straightforward displacement and manipulation of graphics displays, while the right side multiplication routines are used in transformation modeling tasks. Both types of tasks are discussed in the following paragraphs.

The majority of instances in which transformation routines are used involve simple manipulation of a display by translating, rotating, and scaling elements of that display. Routines with an -L or -L3 suffix are selected for these purposes. The following example illustrates the sort of task for which -L or -L3 routines are best suited.

Assume we wish to rotate a triangle about a point in the interior of the triangle. The center of the triangle is at (10,10) relative to the origin of the picture (figure 5-1). TIGS routines perform all transformations relative to the picture origin (not necessarily the center of the terminal screen). Thus, if we simply used the matrix utility routine XROTA, the triangle would be rotated around the origin, not around the interior point we want. To accomplish the desired rotation, the triangle must be:

1. Moved to the origin.

2. Rotated the desired amount.

3. Moved back to its original position.

This involves building a B matrix with three utility calls, as follows:

1. A call to

   XTRNA (-10.0,-10.0,B)

   to move the interior point to the origin (0,0).

2. A call to

   XROTL (45.0,B)

   to rotate the triangle 45°.

   Note that we specify the same array to place the matrix in.

3. A call to

   XTRNL (10.0,10.0,B)

   to move the triangle back to its original position.

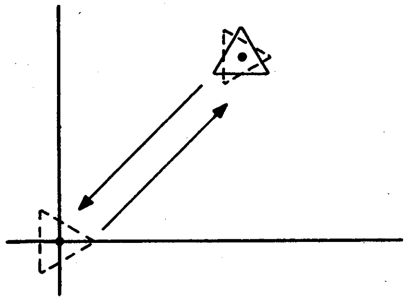


Figure 5-1. Rotation of Triangle About Interior Point

These three calls will achieve the desired effect. Note that the first call, to XTRNA, was to an absolute routine because the first matrix is not concatenated with any other matrix but applied directly to the segment as the segment was originally defined. The other two calls, to XROTL and XTRNL, have the -L suffix because the ultimate effect we want is to be cumulative and in the order specified, the result of matrix concatenation.

Similar steps are required in scaling a display about a point other than the origin.

In the special case in which the matrix utility routines build a matrix to be used in resetting the attribute of a segment which already has a transformation attribute (other than the identity matrix), further steps are necessary. The first example at the end of this section deals with this situation.

Transformation modeling is an extension of the concept of modeling introduced in section 1. The twin objectives of defining and spatially organizing a graphics model are generally accomplished together in defining the primitives, segments, and pictures that compose the model. The example at the end of section 2 employs this method.

The two objectives may, however, be separated. Some models are constructed by defining each segment independently of the others; in effect, each in its own coordinate system. Then, transformations can be applied to each segment to organize the total model. Transformation modeling is the process of orienting segments each drawn in terms of its own coordinate system within another coordinate system. Many engineering drawings are constructed this way: for example, a drawing of a transistor in a subcircuit in an overall circuit. The transistor, the subcircuit, and the overall circuit can be drawn each in its own coordinate system. In regard to the total model, the programmer only knows the positions of each component relative to the larger circuit or coordinate system in which it fits. He knows that the transistor goes at a certain location in the subcircuit, and the subcircuit is situated at a certain location in the larger circuit. Unless he wants to calculate it himself, he does not know the position of the transistor relative to the overall circuit. He employs transformation routines to link each component in the proper overall relationship. From the information he has regarding the relative positions of these components, he decides to use right side multiplication routines to accomplish this. It is clear that before he moves the subcircuit into the overall circuit, he must move the transistor into the subcircuit. If he were to move the transistor after the subcircuit, his knowledge of the position of the transistor relative to the subcircuit would be useless and he would have to calculate its final position himself. Thus, he moves the transistor into the subcircuit, and by cumulative transformations, moves the transistor along with the subcircuit into the overall circuit.

The key to his choice of matrix multiplication routines is the use of the word before. When a transformation must be done before another transformation to preserve overall relationships, the right side routines are used. All tasks analogous to the one described above employ -R or -R3 suffix routines to achieve the desired effect. This process is demonstrated in the second example at the end of this section.

To summarize the discussion of the important task of selecting the proper transformation matrix routine, remember the following:

- Absolute routines, whether matrix utility routines like XTRNA, mode-setting routines like SMXFA, or attribute-resetting routines like RAXFA, imply no matrix concatenation. They are used in the initial transformation of a segment and any time successive transformations are not to be cumulative, but each is applied to the segment as it was originally defined.

- Relative routines with an -L suffix, whether utility routines like XROTL3, mode-setting routines like SMXFL3, or attribute-resetting routines like RAXFL3, imply matrix concatenation with multiplication done on the left. For these routines, the effect of each step in a cumulative transformation series is felt after the effects of the preceding steps. They are used in graphics tasks which involve simple rearranging of elements in a display by translating, rotating, and scaling the segments which make up the display. Refer to the first example at the end of this section.

- Relative routines with an -R suffix, whether utility routines like XSCLR, mode-setting routines like SMXFR, or attribute-resetting routines like RAXFR, imply matrix concatenation with multiplication done on the right. For these routines, the effect of each step in a cumulative transformation series is felt before the effects of the preceding steps. They are used in transformation modeling tasks where objects drawn in one coordinate system are to be oriented in another coordinate system. Refer to the second example at the end of this section.

Following are capsule descriptions of the transformation routines covered in this section.

XTRNxx utility routines build two- and three-dimensional translation matrices.

XROTxx utility routines build two- and three-dimensional rotation matrices.

XSCLxx utility routines build two- and three-dimensional scaling matrices.

XIDNT and XIDNT3 build two- and three-dimensional identity matrices.

XINVR and XINVR3 build two- and three-dimensional inverse matrices.

PUSHx pushes the current transformation matrix (CTM or CTM3) onto the transformation matrix storage stack. POPx replaces the CTM or CMT3 with the matrix on top of the matrix storage stack. CLRSTx clears the transformation matrix storage stack.

The xxXFxx mode/attribute/feature routines control the transformation matrix mode or attribute settings.

TRANSFORMATION MATRIX ROUTINES

CLRSTx

Format

CLRSTK

Clear the 2-D transformation matrix storage stack.

CLRST3

Clear the 3-D transformation matrix storage stack.

Parameters

None.

Programming Notes

These routines clear all entries from the two- and three-dimensional transformation matrix storage stacks. Also, the CTM or CTM3 is set to the identity matrix following a call to CLRSTK or CLRST3.

POPx

Format

POP

Replace the CTM with the matrix on the top of the 2-D transformation matrix storage stack.

POP3

Replace the CTM3 with the matrix on the top of the 3-D transformation matrix storage stack.

Parameters

None.

Programming Notes

POPx modally sets the CTM or CTM3 from the top of the transformation matrix storage stack and moves the other matrices in the stack up one position. If the stack is empty when POPx is called, the CTM or CTM3 is set to the identity matrix and an error message is issued.

Two- and three-dimensional matrices are maintained on separate stacks.

Refer to the general heading in this section for more information on PUSHx and POPx interaction.

PUSHx

Format

PUSH

Place a copy of the CTM on the 2-D transformation matrix storage stack.

PUSH3

Place a copy of the CTM3 on the 3-D transformation matrix storage stack.

Parameters

None.

Programming Notes

PUSHx places a copy of the CTM or CTM3 on the appropriate stack; the CTM or CTM3 is unchanged by a call to PUSHx.

Unpredictable results can occur if the programmer pushes two-dimensional matrices onto the three-dimensional stack or vice versa.

Refer to the general heading in this section for more information on PUSHx and POPx interaction.

XIDNTx

Format

XIDNT(**bmat23**)

Build 2-D identity matrix.

XIDNT3(**bmat34**)

Build 3-D identity matrix.

Parameters

bmat23	Input parameter; 2 x 3 array used for 2-D building matrix.
bmat34	Input parameter; 3 x 4 array used for 3-D building matrix.

Programming Notes

When applied to a segment, the identity matrix results in no transformations. Refer to the general heading in this section for more information.

bmat23 and **bmat34** are the arrays in which the identity matrices are assembled. The arrays must be specified to other utility routines or mode/attribute routines which use the matrices.

XINVRx

Format

XINVR(**bmat23,binv23**)

Build 2-D inverse matrix.

XINVR3(**bmat34,binv34**)

Build 3-D inverse matrix.

Parameters

bmat23	Input parameter; 2 x 3 array used for 2–D building matrix.
binv23	Output parameter; inverse matrix of **bmat23**; 2 x 3 array.
bmat34	Input parameter; 3 x 4 array used for 3–D building matrix.
binv34	Output parameter; inverse matrix of **bmat34**; 3 x 4 array.

Programming Notes

XINVRx accepts the matrix in array **bmat23** or **bmat34**, creates the inverse of that matrix and places it in **binv23** or **binv34**.

XINVRx is useful in reversing the effect that a previous matrix has had on a segment. That is, if matrix X has moved a segment from point A to point B, matrix X^{-1} will move the segment back to A.

binv23 and **binv34** are the arrays in which the inverse matrices are placed. The arrays must be specified to other utility routines or mode/attribute routines which use the matrices.

XROTxx

Format

XROTA(**deg,bmat23**)

2-D absolute rotation.

XROTL(**ddeg,bmat23**)

2-D relative rotation, left multiplication.

XROTR(**ddeg,bmat23**)

2-D relative rotation, right multiplication.

XROTA3(**idaxis,deg,bmat34**)

3-D absolute rotation.

XROTL3(**idaxis,ddeg,bmat34**)

3-D relative rotation, left multiplication.

XROTR3(**idaxis,ddeg,bmat34**)

3-D relative rotation, right multiplication.

Parameters

deg	Input parameter; absolute number of degrees to rotate segment; positive values indicate counterclockwise rotation.
ddeg	Input parameter; relative number of degrees to rotate segment; positive values indicate counterclockwise rotation.
idaxis	Input parameter; axis about which to perform the rotation for 3-D; specify as 1HX, 1HY, or 1HZ.
bmat23	Input parameter (relative rotation) and output parameter (all); 2 x 3 array used for 2–D building matrix.
bmat34	Input parameter (relative rotation) and output parameter (all); 3 x 4 array used for 3–D building matrix.

Programming Notes

Rotation transformations rotate segments about the picture origin by the specified number of degrees. To rotate a segment about any other point, refer to the general heading in this section and to the examples at the end of the section.

For the **deg** and **ddeg** parameters, positive values result in counterclockwise rotations. Negative values result in clockwise rotations. For three-dimensional pictures, rotation can be done only around one of the major coordinate axes specified in FORTRAN Hollerith format. Direction of three-dimensional rotation is as viewed from the positive side of the axis of rotation. Note that this is not necessarily the direction of rotation that will appear on the screen. For example, the call

```
XROTA3(1HZ,45.,B)
```

results in a matrix to effect counterclockwise rotation of 45° about the z-axis, as viewed in figure 5-2.

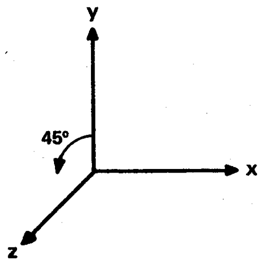


Figure 5-2. Three-Dimensional Rotation, Front View

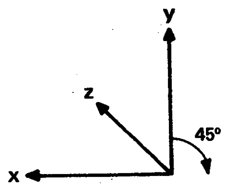


Figure 5-3. Three-Dimensional Rotation, Back View

If, however, the line of vision of the three-dimensional window were chosen so that the display is viewed from the back (figure 5-3), the rotation appears to be in a clockwise direction.

The absolute rotation routines imply that the B matrix is to be built as specified in the call. The relative translation routines imply that the B matrix is to be built by concatenating the specified matrix with the existing B matrix to form the new B matrix. The programmer has a choice of right or left matrix multiplication for relative routines. Refer to the general heading in this section for more information on relative and absolute transformation.

Only the beginning character position of a text string is altered by XROTxx; the basic orientation of the string to the display screen does not change. Refer to xxROT in section 2 for more information on rotation of text.

bmat23 and **bmat34** are the arrays in which the rotation matrices are assembled. The arrays must be specified to other utility routines or mode/attribute routines which use the matrices.

XSCLxx

Format

XSCLA(**sx,sy,bmat23**)

2-D absolute scale.

XSCLL(**sdx,sdy,bmat23**)

2-D relative scale, left multiplication.

XSCLR(**sdx,sdy,bmat23**)

2-D relative scale, right multiplication.

XSCLA3(**sx,sy,sz,bmat34**)

3-D absolute scale.

XSCLL3(**sdx,sdy,sdz,bmat34**)

3-D relative scale, left multiplication.

XSCLR3(**sdx,sdy,sdz,bmat34**)

3-D relative scale, right multiplication.

Parameters

sx,sy,sz	Input parameters; absolute scale factors in x, y, and z axes.
sdx,sdy,sdz	Input parameters; relative scale factors in x, y, and z axes.
bmat23	Input parameter (relative scale) and output parameter (all); 2 x 3 array used for 2-D building matrix.
bmat34	Input parameter (relative scale) and output parameter (all); 3 x 4 array used for 3-D building matrix.

Programming Notes

Scaling transformations are done relative to the picture origin only. It may be useful to think of two-dimensional scaling transformations as occurring on a sheet of elastic material. The material is stretched or shrunk the appropriate amounts in each direction while remaining anchored at the origin. Segments on the fabric are scaled up or down as the fabric is stretched or shrunk. It can be seen that segments that are not centered at the origin will not only be scaled up or down but also generally displaced by this process. Scaling a segment about a point other than the origin is done in a way analogous to rotating a segment about a point other than the origin. Refer to the general heading in this section for more information.

sx, sy, sz, sdx, sdy, and **sdz** represent the absolute and relative scale factors in the x, y, and z directions. Scale factors > 1 result in enlargement. Positive scale factors < 1 result in shrinking. For example, SX=2 results in an effective doubling of the distance between units on the x axis, and an elongation in the x direction of an affected segment. Unequal scale factors (for example, SX=2, SY=3) result in distortion of the original display. A negative scale factor in any axis results in the mirroring of the segment about the other axes (figure 5-4).

The absolute scaling routines imply that the B matrix is to be built as specified in the call. The relative scaling routines imply that the B matrix is to be built by concatenating the specified matrix with the existing B matrix to form the new B matrix. The programmer has a choice of right or left matrix multiplication for relative routines. Refer to the general heading in this section for more information on relative and absolute transformations.

Text segments are not scaled; the beginning character position of a text string may be altered by a scaling transformation.

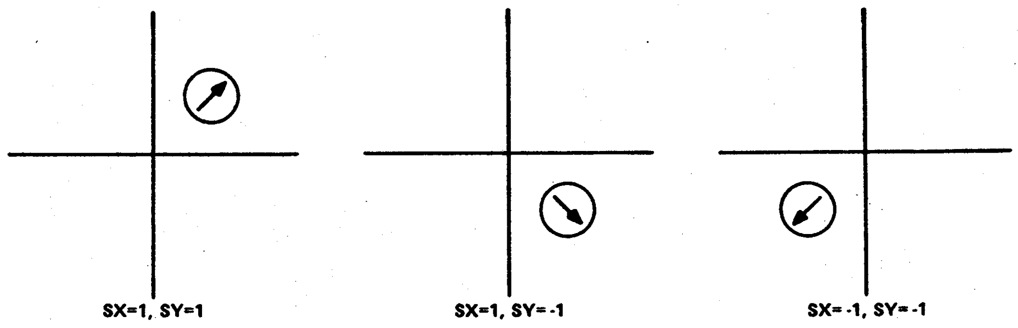


Figure 5-4. Mirroring of Display

bmat23 and **bmat34** are the arrays in which the scaling matrices are assembled. The arrays must be specified to other utility routines or mode/attribute routines which use the matrices.

XTRNxx

Format

XTRNA(**x,y,bmat23**)

2-D absolute translation.

XTRNL(**dx,dy,bmat23**)

2-D relative translation, left multiplication.

XTRNR(**dx,dy,bmat23**)

2-D relative translation, right multiplication.

XTRNA3(**x,y,z,bmat34**)

3-D absolute translation.

XTRNL3(**dx,dy,dz,bmat34**)

3-D relative translation, left multiplication.

XTRNR3(**dx,dy,dz,bmat34**)

3-D relative translation, right multiplication.

Parameters

x,y,z	Input parameters; absolute displacements in x, y, and z directions.
dx,dy,dz	Input parameters; relative displacements in x, y, and z directions.
bmat23	Input parameter (relative translation) and output parameter (all); 2 x 3 array used for 2-D building matrix.
bmat34	Input parameter (relative translation) and output parameter (all); 3 x 4 array used for 3-D building matrix.

Programming Notes

Translation transformations displace segments in a straight line by the specified distances in the x, y, and z directions.

The absolute translation routines imply that the B matrix is to be built as specified in the call. The relative translation routines imply that the B matrix is to be built by concatenating the specified matrix with the existing B matrix to form the new B matrix. The programmer has a choice of left and right matrix multiplication for relative routines. Refer to the general heading in this section for more information on relative and absolute transformations.

bmat23 and **bmat34** are the arrays in which the translation matrices are assembled. The arrays must be specified to other utility routines or mode/attribute routines which use the matrices.

MODE/ATTRIBUTE/FEATURE ROUTINES

xxXFxx

Format

TFXFA(**lxlat,lscal,lrot**)

TFXFL(**lxlat,lscal,lrot**)

TFXFR(**lxlat,lscal,lrot**)

Test for terminal hardware capability to perform 2-D transformations; there is no difference among TFXFA, TFXFL, and TFXFR.

TFXFA3(**lxfm3,lpersp,lpyram**)

TFXFL3(**lxfm3,lpersp,lpyram**)

TFXFR3(**lxfm3,lpersp,lpyram**)

Test for terminal hardware capability to perform 3-D transformations; there is no difference among TFXFA3, TFXFL3, and TFXFR3.

SMXFA(**bmat23**)

Modally set absolute 2-D transformation.

SMXFL(**bmat23**)

Modally set relative 2-D transformation, left multiplication.

SMXFR(**bmat23**)

Modally set relative 2-D transformation, right multiplication.

SMXFA3(**bmat34**)

Modally set absolute 3-D transformation.

SMXFL3(**bmat34**)

Modally set relative 3-D transformation, left multiplication.

SMXFR3(**bmat34**)

Modally set relative 3-D transformation, right multiplication.

TMXFA(**bmat23**)

TMXFL(**bmat23**)

TMXFR(**bmat23**)

Test the current modally set 2-D transformation matrix; a copy of the CTM is placed in **bmat23.** There is no difference among TMXFA, TMXFL, and TMXFR.

TMXFA3(**bmat34**)

TMXFL3(**bmat34**)

TMXFR3(**bmat34**)

Test the current modally set 3-D transformation matrix; a copy of the CTM3 is placed in **bmat34.** There is no difference among TMXFA3, TMXFL3, and TMXFR3.

RAXFA(**idseg,bmat23**)

Absolutely reset the transformation attribute of a 2-D segment.

RAXFL(**idseg,bmat23**)

Relatively reset the transformation attribute of a 2-D segment, left multiplication.

RAXFR(**idseg,bmat23**)

Relatively reset the transformation attribute of a 2-D segment, right multiplication.

RAXFA3(**idseg,bmat34**)

Absolutely reset the transformation attribute of a 3-D segment.

RAXFL3(**idseg,bmat34**)

Relatively reset the transformation attribute of a 3-D segment, left multiplication.

RAXFR3(**idseg,bmat34**)

Relatively reset the transformation attribute of a 3-D segment, right multiplication.

TAXFA(**idseg,bmat23**)

TAXFL(**idseg,bmat23**)

TAXFR(**idseg,bmat23**)

Test the transformation attribute of a 2-D segment; a copy of the matrix is placed in **bmat23.** There is no difference among TAXFA, TAXFL, and TAXFR.

TAXFA3(**idseg,bmat34**)

TAXFL3(**idseg,bmat34**)

TAXFR3(**idseg,bmat34**)

Test the transformation attribute of a 3-D segment; a copy of the matrix is placed in **bmat34.** There is no difference among TAXFA3, TAXFL3, and TAXFR3.

Parameters

lxlat	Output parameter; if LXLAT=.TRUE., 2-D translation is hardware supported; otherwise, it is not.
lscal	Output parameter; if LSCAL=.TRUE., 2-D scaling is hardware supported; otherwise, it is not.
lrot	Output parameter; if LROT=.TRUE., 2-D rotation is hardware supported; otherwise, it is not.
lxfm3	Output parameter; if LXFM3=.TRUE., all 3-D transformation functions are hardware supported; otherwise, they are not.
lpersp	Output parameter; if LPERSP=.TRUE., perspective preservation during 3-D transformations is hardware supported; otherwise, it is not.
lpyram	Output parameter; if LPYRAM=.TRUE., clipping to 3-D window pyramid during transformations is hardware supported; otherwise, it is not.
bmat23	Input parameter (SMXFA, SMXFL, SMXFR, RAXFA, RAXFL, RAXFR) or output parameter (TMXFA, TMXFL, TMXFR, TAXFA, TAXFL, TAXFR) which is the name of the 2 x 3 array used in 2-D transformations.
bmat34	Input parameter (SMXFA3, SMXFL3, SMXFR3, RAXFA3, RAXFL3, RAXFR3) or output parameter (TMXFA3, TMXFL3, TMXFR3, TAXFA3, TAXFL3, TAXFR3) which is the name of the 3 x 4 array used in 3-D transformations.
idseg	Input parameter identifying the segment whose attribute is to be tested or reset.

Programming Notes

Absolute (-A suffix) and relative (-R and -L suffix) test routines perform identical functions; they are included only for consistency.

The test feature (TF prefix) routines test only to see if transformation capabilities are hardware or software supported; all capabilities are supported for all postprocessors. If capabilities are software supported, however, execution time is greatly increased.

For background information on the **lpersp** and **lpyram** parameters, refer to the three-dimensional window routines in section 4.

bmat23 and **bmat34** are the names of arrays in which the transformation matrices are placed. The matrices may be supplied by the programmer or assembled by the matrix utility routines described earlier in this section. The array containing the matrix is then specified in the call to the mode/attribute routine.

EXAMPLES OF TRANSFORMATION ROUTINE USAGE

The following examples use many of the transformation routines documented in this section. They are not complete programs.

WARNING

Refer to appropriate postprocessor and operating system appendices for format of PROGRAM statement.

The first example illustrates the use of TIGS transformation routines in simple translation, rotation, and scaling of segments. In this program, a two-dimensional picture is used. The coordinate axes are drawn in for reference markings. Then, a triangle, circle, and square are drawn in quadrants I, II, and III respectively (figure 5-5). In three steps, the square is rotated and placed in the circle (figure 5-6), the triangle is scaled up and placed in the square (figure 5-7), and then all three segments are moved to the origin (figure 5-8). Each step is shown separately on the screen, using the TIGS DSPLAY call (section 7) and the FORTRAN PAUSE command.

The second example illustrates the use of both left and right matrix multiplication routines. A house, chimney, and flag are defined, each without regard for the final relationship between them. However, before each is defined, a matrix is built to move that segment into the proper relationship with the model unit, or coordinate system, which will contain it - the house with the overall picture, the chimney with the house, and the flag with the chimney. Cumulative transformations are effected by concatenating the old CTM with a new matrix. Each segment is oriented without regard for what eventually happens to the segment, or coordinate system, in which it is oriented. For example, the fact that the house will eventually double in size does not matter to the programmer when he scales the chimney down to 1/8 scale in order to fit it to the originally defined house. The doubling in size, through cumulative transformations, will apply to the now properly oriented chimney as well as to the house. Program output is reproduced in figures 5-9 and 5-10.

```
PROGRAM statement
DIMENSION A(2,3),B(2,3),C(2,3)           Dimension arrays to be used for matrices.
CALL INITIG(.TRUE.,.TRUE.,5LIFILE)
CALL SMPLIM(-50.,-50.,50.,50.)   }
CALL OPNSEG(1)                   }
CALL MOVEA(0.,50.)               }
CALL DRAWA(0.,-50.)              }       Draw coordinate axes.
CALL MOVEA(-50.,0.)              }
CALL DRAWA(50.,0.)               }
CALL CLSSEG                      }
CALL OPNSEG(2)                   }
CALL MOVEA(15.,15.)              }
CALL DRAWA(20.,25.)              }       Draw triangle in first quadrant.
CALL DRAWA(25.,15.)              }
CALL DRAWA(15.,15.)              }
CALL CLSSEG                      }
CALL OPNSEG(3)
CALL MOVEA(-15.,15.)
CALL ARCDR(-5.,5.,360.)
CALL CLSSEG
```

Draw circle in second quadrant.

```
CALL OPNSEG(4)
CALL MOVEA(-15.,-15.)
CALL DRAWA(-15.,-25.)
CALL DRAWA(-25.,-25.)
CALL DRAWA(-25.,-15.)
CALL DRAWA(-15.,-15.)
CALL CLSSEG
```

Draw square in third quadrant.

```
CALL DSPLAY
PAUSE
```

Display picture on terminal screen and then wait for operator response.

```
CALL XTRNA(20.,20.,A)
CALL XROTL(45.,A)
CALL XTRNL(-20.,20.,A)
CALL RAXFA(4,A)
CALL DSPLAY
PAUSE
```

Construct matrix to move square to origin, rotate it 45° counterclockwise and then move it onto the circle. Note that first call is to an absolute routine but subsequent calls to build matrix are relative routines for concatenation. Call to reset attribute routine could have been to RAXFL instead of RAXFA since transformation matrix attribute of square was the identity matrix (no transformations). Modified picture is then displayed and program pauses.

```
CALL XTRNA(-20.,-20.,B)
CALL XSCLL(1.5,1.5,B)
CALL XTRNL(-20.,20.,B)
CALL RAXFA(2,B)
CALL DSPLAY
PAUSE
```

Construct matrix to move triangle to origin, scale it up 1.5 times and move it onto the circle. Other observations from last step apply here.

```
CALL XTRNA(20.,-20.,C)
CALL RAXFA(3,C)
```

For the final step, each segment is moved to the origin. Each segment has to be moved separately. For segment 3 (circle) which has not been previously moved, this is accomplished by setting up the translation matrix C and resetting segment 3's transformation attribute.

```
CALL RAXFL(2,C)
CALL RAXFL(4,C)
CALL DSPLAY
PAUSE
```

For segments 2 and 4 (triangle and square) the procedure is identical except that a relative routine using left matrix multiplication, namely RAXFL, is used. The same matrix is used for all three segments.

```
CALL QUITIG(.TRUE.)
STOP
END
```

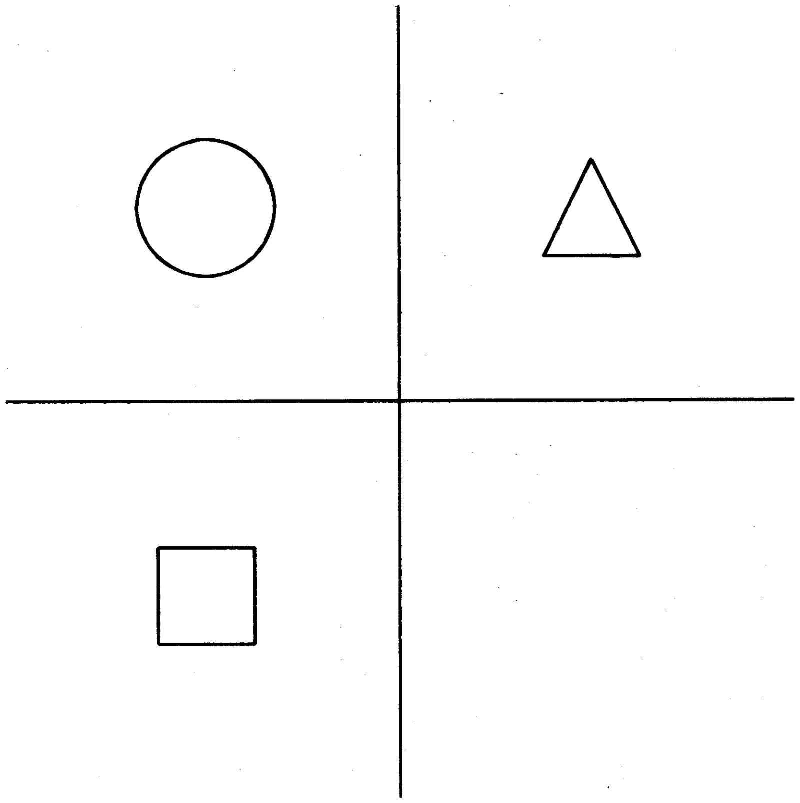


Figure 5-5. Output from First Example, Step 1

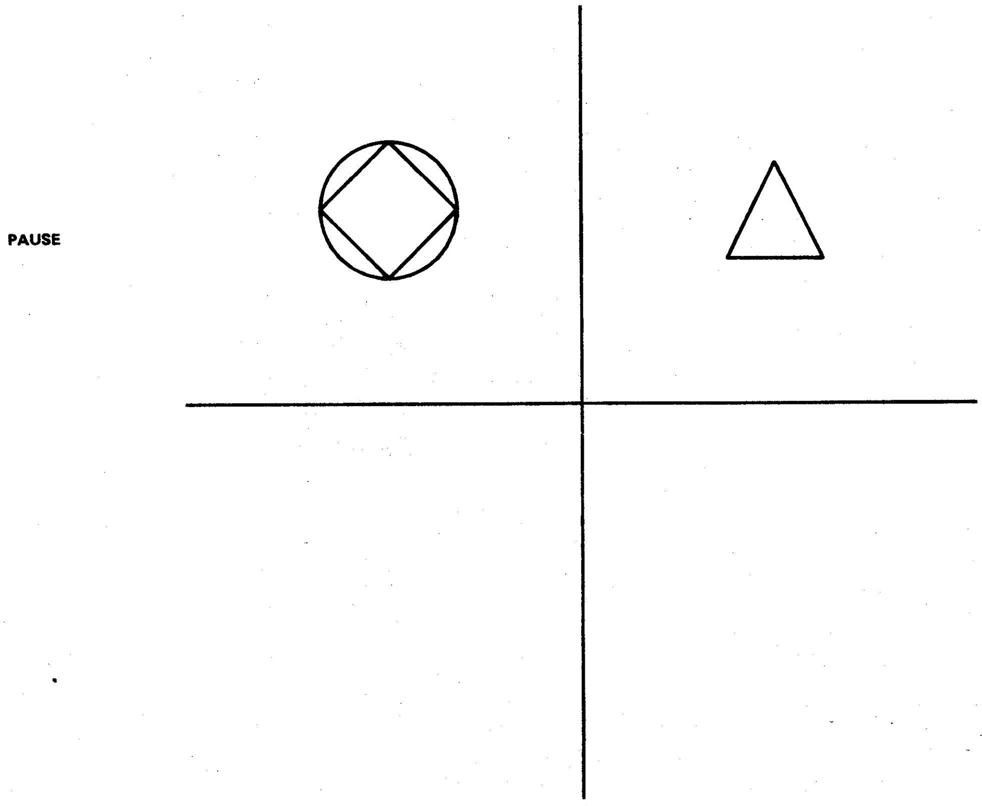


Figure 5-6. Output from First Example, Step 2

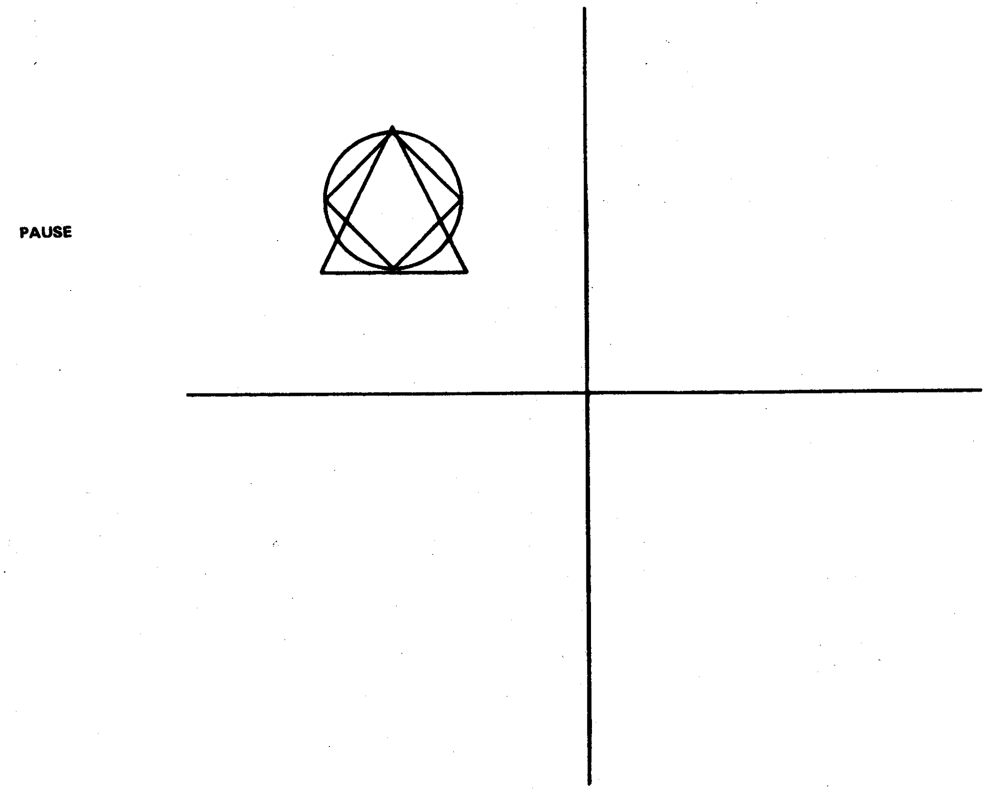


Figure 5-7. Output from First Example, Step 3

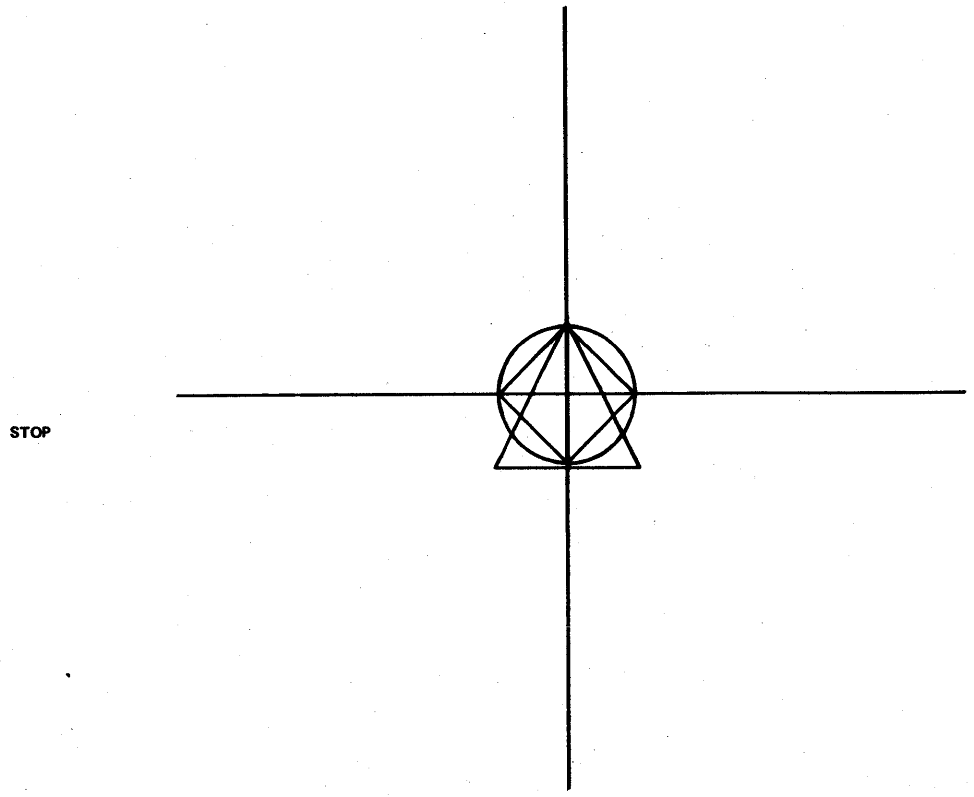


Figure 5-8. Output from First Example, Step 4

PROGRAM statement	
DIMENSION A(2,3)	Dimension array in which to build transformation matrices. Only one array is necessary because CTM is saved by TIGS; the programmer does not have to do so.
CALL INITIG(.TRUE.,.TRUE.,5LIFILE)	
CALL SMPLIM(-100.,-100.,100.,100.)	
CALL XTRNA(-30.,0.,A) CALL XSCLL(2.,2.,A)	Build transformation matrix for house segment; it is to be centered in the picture and scaled up two times. Note first call (XTRNA) is to an absolute routine because it is the first transformation to be applied to the segment. This applies to the first step of building the matrix for each segment. Second call is to left multiplication routine because the house is to be scaled after the house is moved.
CALL SMXFA(A)	Set transformation mode for house segment to be drawn in next section of program. An absolute routine is used because it is the first CTM and does not involve matrix multiplication with any other CTM. Note that this CTM will eventually be concatenated with the matrices for the chimney and flag segments.
CALL OPNSEG(1) CALL MOVEA(0.,0.) CALL DRAWA(60.,0.) CALL DRAWA(30.,30.) CALL DRAWA(0.,0.) CALL DRAWA(0.,-30.) CALL DRAWA(60.,-30.) CALL DRAWA(60.,0.) CALL CLSSEG	Draw house.
CALL XSCLA(.125,.125,A) CALL XTRNL(30.,30.,A)	Build matrix for chimney segment. It is to be scaled down to 1/8 size and moved to the right 30 units and up 30 units. This will place it on top of the house as the house was originally defined. Left routines are used because this step involves straightforward scaling and translating. Do not be concerned that the house is subsequently altered by transformations; the next step takes care of that problem.
CALL SMXFR(A)	Set transformation mode for chimney segment to be drawn in next section of program. A relative routine is used because the final position of the chimney is a result of cumulative transformations; a right multiplication routine is used because the chimney must first be moved to the top of the house and then house and chimney moved together to their final location. Only the relationship of chimney to house is known, not the relationship of the chimney to the overall picture. The chimney is moved before the house transformation is applied.

```
CALL OPNSEG(2)
CALL MOVEA(0.,0.)
CALL DRAWA(20.,-20.)
CALL DRAWA(20.,40.)
CALL DRAWA(-40.,40.)
CALL DRAWA(-40.,-40.)
CALL DRAWA(0.,0.)
CALL CLSSEG
```

Draw chimney.

```
CALL XTRNA(-30.,-30.,A)
CALL XROTL(-90.,A)
CALL XTRNL(20.,100.,A)
```

Build matrix for flag segment. It is to be rotated 90° and placed on top of the chimney. Since the flag is not defined at the origin, it is moved to the origin before rotation and then moved to the top of the chimney as the chimney was originally defined. Left routines are used in this step because we want to move the flag to the origin, then rotate it, and then move it to the chimney. Each step happens after the preceding step.

```
CALL SMXFR(A)
```

Set transformation mode for flag segment to be drawn in next section of program. The relative routine states that the final position of the flag is a result of cumulative transformations. The right routine is used so that when the new matrix is concatenated with the old CTM, the transformation to move the flag to the top of the chimney is applied before the transformation to move flag, chimney, and house to their final locations.

```
CALL OPNSEG(3)
CALL MOVEA(30.,30.)
CALL DRAWR(0.,30.)
CALL DRAWR(25.,0.)
CALL DRAWR(0.,-30.)
CALL MOVER(-25.,0.)
CALL DRAWR(60.,0.)
CALL CLSSEG
```

Draw flag; relative draw routines are used (refer to section 2).

```
CALL DSPLAY
```

Display picture.

```
PAUSE
CALL XIDNT(A)
```

Set identity matrix to move all segments back to original location.

```
CALL RAXFA(1,A)
CALL RAXFA(2,A)
CALL RAXFA(3,A)
```

Reset transformation attributes of all three segments to show position of segments as they were originally defined.

```
CALL DSPLAY
PAUSE
CALL QUITIG(.TRUE.)
STOP
END
```

PAUSE

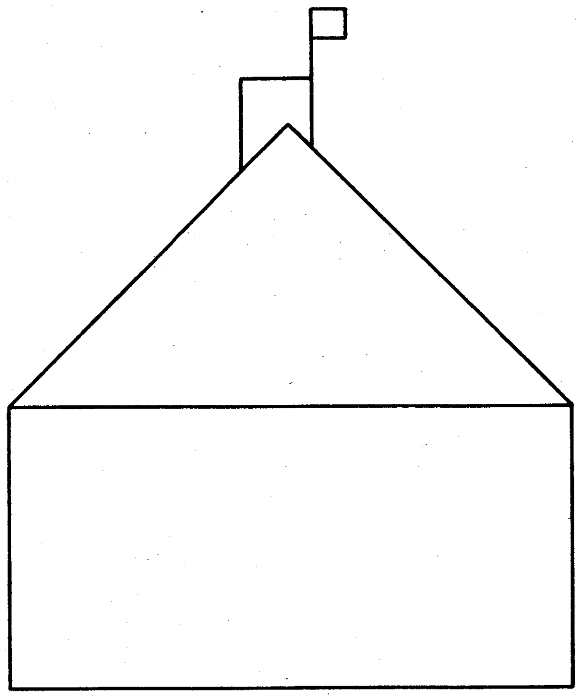


Figure 5-9. Output from Second Example, Step 1

PAUSE

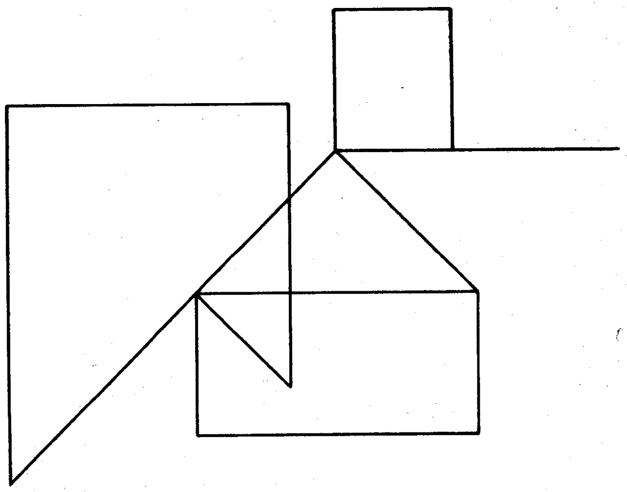


Figure 5-10. Output from Second Example, Step 2

GEOMETRY UTILITIES

The following TIGS calls are documented in this section.

ENDPAR
ENDPLN
ENDPL3
RTANGL
RTANG3

There are no mode/attribute/feature calls specifically associated with TIGS geometry utilities.

An example of geometry utility usage follows the documentation of the calls listed above.

GENERAL

Before calling graphic primitives routines for drawing arcs or lines, the programmer may want to investigate how much of an arc or line lies within a specific rectangular area. This can be useful information in calculating how much of a picture will appear in a given window, for example. He can in this way determine in advance how window clipping will affect his display.

There are separate routines for two- and three-dimensional pictures. The RTANGx routine sets up the boundaries of the rectangle or rectangular parallelepiped of interest. Then the ENDPLx routine calculates the points of intersection, if any, of a line with the area or volume described by RTANGx. The ENDPAR routine calculates the points of intersection, if any, of an arc with the area of interest.

GEOMETRY UTILITY ROUTINES

ENDPAR

Format

ENDPAR(**cx,cy,x1,y1,x2,y2,narcs,xn1,yn1,xn2,yn2**)
Determine endpoints of specified 2-D arc.

Parameters

cx,cy	Input parameters; coordinates of the center of the arc to be checked against the boundaries specified by RTANGL.
x1,y1 **x2,y2**	Input parameters; coordinates of the endpoints of the arc.
narcs	Output parameter; contains the number of arc sections that are inside the area specified by RTANGL: $0 \leq \textbf{narcs} \leq 5$ If NARCS=0, then **xn1,yn1,xn2,yn2** are meaningless (refer to the following programming notes).
xn1,yn1 **xn2,yn2**	Output parameters; coordinates of intersections of line with boundaries of area. Each is dimensioned as an array of five words (refer to the following programming notes).

Programming Notes

This subroutine determines whether or not the specified arc crosses the boundaries of the area defined by RTANGL. If the arc intersects any boundary, the intersection point is calculated. An arc can intersect a rectangle such that up to five sub-arcs (ten endpoints) are produced (figure 6-1). Thus, output parameters **xn1,yn1, xn2,yn2** must each be dimensioned as five-word arrays to contain the possible endpoints of the sub-arcs. The endpoints resulting from this operation are returned to the application program for subsequent use in the appropriate primitives.

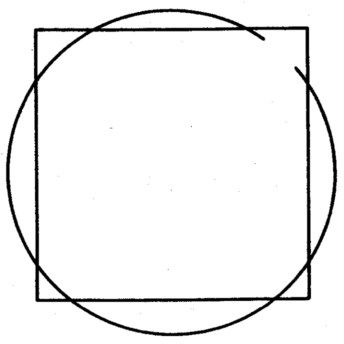


Figure 6-1. Intersection of Arc With Clipping Rectangle

Coordinates are specified and returned as user coordinates (that is, in terms of the coordinate system specified by SMPLIM, section 3).

ENDPLx

Format

ENDPLN(**x1,y1,x2,y2,ishow,xn1,yn1,xn2,yn2**)

Determine endpoints of specified 2-D line.

ENDPL3(**x1,y1,z1,x2,y2,z2,ishow,xn1,yn1,zn1, xn2,yn2,zn2**)

Determine endpoints of specified 3-D line.

Parameters

x1,y1,z1 **x2,y2,z2**	Input parameters; endpoints of line to be checked against boundaries specified by RTANGx.
ishow	Output parameter; specifies the result of the boundary check.

If ISHOW=0, line lies outside area or volume and **xn1,yn1,zn1,xn2,yn2,zn2** are meaningless.

If ISHOW=1, line lies totally within area and new endpoints are the same as original endpoints:

x1=xn1	**x2=xn2**
y1=yn1	**y2=yn2**
z1=zn1	**z2=zn2**

If ISHOW=2, first endpoint is clipped and at least one of the following relationships is true:

x1 ≠ xn1
y1 ≠ yn1
z1 ≠ zn1

If ISHOW=3, second endpoint is clipped and at least one of the following relationships is true:

x2 ≠ xn2
y2 ≠ yn2
z2 ≠ zn2

If ISHOW=4, both endpoints are clipped and at least one of the relationships in each column is true:

x1 ≠ xn1	**x2 ≠ xn2**
y1 ≠ yn1	**y2 ≠ yn2**
z1 ≠ zn1	**z2 ≠ zn2**

xn1,yn1,zn1 **xn2,yn2,zn2**	Output parameters; coordinates of intersections of line with boundaries of area/volume (new endpoints).

Programming Notes

These subroutines determine whether or not the specified line crosses the boundaries of the area/volume defined by RTANGx. If the line intersects any boundary, the intersection point is calculated. The endpoints resulting from this operation are returned to the application program for subsequent use in the appropriate primitives.

Coordinates are specified and returned as user coordinates (that is, in terms of the coordinate system specified by SMPLIx, section 3).

RTANGx

Format

RTANGL(**xll,yll,xur,yur**)

Define limits for 2-D endpoint calculations.

RTANG3(**xllh,yllh,zllh,xury,yury,zury**)

Define limits for 3-D endpoint calculations.

Parameters

xll,yll,xur,yur	Input parameters; coordinates of the lower left and upper right corners of the 2-D rectangle used by the ENDPLN and ENDPAR subroutines.
xllh,yllh,zllh, xury,yury,zury	Input parameters; coordinates of the lower left hither and upper right yon corners of the 3-D rectangular parallelepiped used by the ENDPL3 subroutine (refer to the following programming notes).

Programming Notes

The parameters for these routines are specified the same as the parameters for SMPLIx, section 3. However, these routines have no effect on picture limits as established by SMPLIx. They simply specify the area or volume to be used when subsequent ENDPLx and ENDPAR calls are made.

Coordinates are specified as user coordinates established by a prior call to SMPLIx.

EXAMPLE OF GEOMETRY UTILITY USAGE

The following example uses the geometry utilities calls documented in this section. It is not a complete program. Postprocessor dependent portions of the program are not illustrated, nor are portions not dealing with topics covered in this section.

The program tests for the portions of a proposed line and arc which will fit in a defined rectangle, then draws those portions. Figure 6-2 illustrates the situation (this figure is not program output). The rectangle and the parts of the line and arc which lie outside the rectangle are drawn with dotted lines. The parts of the line and arc lying within the rectangle are drawn with solid lines.

WARNING

Refer to postprocessor and operating system appendices for format of PROGRAM statement.

```
      PROGRAM Statement

      DIMENSION X1(5),Y1(5),X2(5),Y2(5)
```

Dimension 4 arrays of 5 words each for ENDPAR routine, because the intersection of an arc with a rectangle may yield up to 5 subarcs (refer to figure 6–1).

```
      CALL RTANGL(.23,.3,.7,.7)
```

Define rectangle of interest; program uses default picture with units of (0.,0.) to (1.,1.).

```
      CALL ENDPLN(.6,.9,.8,.1,ISHOW,XN1,YN1,XN2,YN2)
```

Define line; on return, ISHOW will be set to 4, and XN1,YN1,XN2,YN2 will contain the coordinates of the endpoints of the line segment contained in the rectangle.

```
      CALL MOVEA(XN1,YN1)

      CALL DRAWA(XN2,YN2)
```

Draw the line segment.

```
      CALL ENDPAR(.5,.5,.5,.8,.5,.2,NARCS,X1,Y1,X2,Y2)

      IF (NARCS.EQ.0) GO TO 200
```

Define arc; on return, NARCS is set to 2 which indicates 2 subarcs are contained in the rectangle. If NARCS had been 0 (no subarcs in rectangle), the arc drawing routine would have been skipped.

```
      DO 150 I=1,NARCS

      CALL MOVEA(X1(I),Y1(I))

      CALL ARCA(.5,.5,X2(I),Y2(I))

  150 CONTINUE

  200 CONTINUE
          .
          .
          .
```

The arrays X1 and Y1 contain the starting point of the two subarcs and the arrays X2 and Y2 contain the endpoints. There are two significant words in each array (NARCS=2). The program loops twice to draw the subarcs from starting point to endpoint.

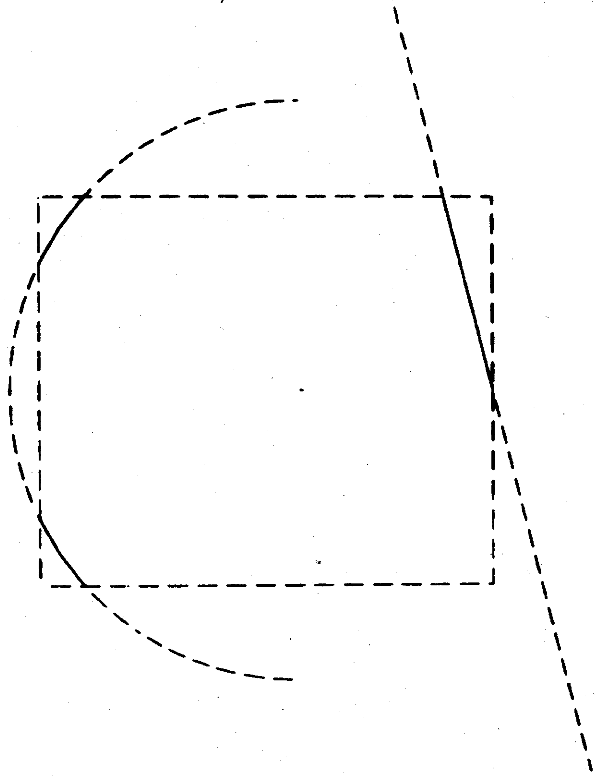


Figure 6–2. Example of Geometry Utility Usage

INTERACTION

The following TIGS routines are documented in this section.

EVENT	PROMPT	TAINFO
KEYBRD	RAAC	TFAC
KYAC	RAINFO	TFID
KYOFF	SMAC	TFLOCR
KYON	SMID	TMAC
LOCATE	SMINFO	TMID
PREEVN	SMLOCR	TMINFO
PRELOC	TAAC	TMLOCR

This list includes routines which perform interactive functions, and those mode/attribute/feature routines which affect interaction routines. The mode/attribute/feature routines are placed in a separate group following the interaction routines.

Following the description of the routines is a comprehensive example which illustrates interactive programming techniques.

GENERAL

TIGS may be used in either passive or interactive mode. In the passive mode, the terminal operator is essentially a spectator observing the display generated by the application program. In the interactive mode, the terminal operator responds to the graphics display with terminal input which influences and modifies the display. Calls to TIGS interaction routines permit this interaction. Only TIGS interaction routines are described in this section; a limited form of interaction is possible through the operating system (such as with the FORTRAN PAUSE statement) but this is interaction between the operator and the operating system, not the operator and the TIGS application program.

The TIGS interaction subroutines are used by the application program to obtain reports about events that occur when the terminal operator responds to the graphics display. Possible events are the picking of a segment with a locator device and the pressing of a function key. The interaction routines have been defined independently of the physical input devices to which they relate. Thus, the programmer can write his program for virtual devices (such as the locator device) defined for the preprocessor and the postprocessor will link these virtual devices to the equivalent physical input devices available at a given terminal. The programmer interrogates the postprocessor as to available virtual input devices with the TFLOCR subroutine. The programmer should consult the appropriate postprocessor appendix for specific information on linkage of virtual input devices to physical input devices.

These terminal events can be assigned by the application program to action types which dictate what action will be performed when an action occurs. There are three action types as follows:

- Ignore - No action is taken by the system on the terminal event. This ignore capability is included so that specific segment pick and function key events may be temporarily deactivated during an interactive session. The ignore action can be permanently assigned to segments which are not meant to be involved in interaction.

- Recognize - When a segment or function key which has been assigned the recognize action is selected by the terminal operator, that selection is recognized by the system. When an event is recognized, the information about that event is reported to the application program by the subroutine EVENT. In addition, if the event is a segment pick, the event is echoed at the terminal. The implementation of the echo is postprocessor dependent. For example, the segment may be redrawn or made to blink to echo a pick.

- Terminate - The terminate action includes all the functions of the recognize action. In addition, when a segment or function key has been assigned to the terminate action type, the selection of that segment or function key flags the event as the last event in the queue for a given call to the subroutine EVENT. If no segment or function key has been assigned to the terminate action type, there is no way to return control to the main program from the subroutine EVENT. This is the default action type.

Action types are assigned via the xxAC mode/attribute routines for segments. The action type to which a segment is assigned becomes an attribute of that segment, and thus may be reset. For example, a segment may initially be assigned the recognize action and subsequently reset to the ignore action.

Action types are assigned to the function keys with the KYAC, KYON, and KYOFF routines.

TIGS interaction routines can be divided into three main groups. The first group includes the TIGS main interactive capability centered on the EVENT subroutine. The second group is composed of routines for dealing with interactive text items. The third is composed of routines which allow the reporting of screen locations to the application program.

The first group is composed of the following interaction routines.

EVENT
PREEVN
KYON
KYOFF
KYAC
xxAC
xxID
xxINFO

Calling the EVENT subroutine enables the segment pick and function key press events. The program pauses while the terminal operator selects one or more segments or function keys. The information associated with each segment or function key is not reported to the application program until the terminate action is performed by the selection of a segment or function key that has been assigned to the terminate action type. Then the EVENT subroutine is called repeatedly until all (or as much as the programmer wants) of the event information is processed. EVENT is called once for each event which has been placed on the queue.

PREEVN is called to prepare the mode set locator (refer to SMLOCR) for use as a segment pick device. The locator device is not displayed until a subsequent call to EVENT. If there is a current EVENT queue, it is cleared.

KYAC, KYOFF, and KYON assign action types to function keys. Depending on the terminal, function keys may be a separate set of keys or function keys may be simulated by certain keys on the alphanumeric keyboard. (Refer to the appropriate postprocessor appendix for details.) KYAC assigns individual function keys; KYOFF and KYON assign all keys at once.

xxAC assigns and tests action types for individual segments. TFAC tests for postprocessor support of basic interaction via the EVENT subroutine.

xxID assigns and tests intrasegment identifiers for subsegments. For example, a single segment may be composed of a triangle and a ball. Although a segment can only be picked as a whole, intrasegment IDs can be used to inform the application program of whether the locator was on the triangle or the ball when the segment was picked. Note that when the event is echoed at the terminal, however, the entire segment, not just the portion picked, is echoed.

xxINFO is used to store information about a segment which can be reported to the application program. Its use is not restricted to interactive graphics, but will most often be used in connection with interactive graphics.

The second group is composed of the following interaction routines.

KEYBRD
PROMPT

KEYBRD is used to inform the application program of keyboard input from the terminal in the form of an alphanumeric character string. PROMPT is used by the application program to display text strings on the display screen. The system viewport area of the screen is used by both routines.

The third group is composed of the following interaction routines.

LOCATE
PRELOC
xxLOCR

The LOCATE subroutine is used to obtain the coordinates of the locator device for use in the application program. The locations are stored on a queue analogous to the EVENT queue. The program pauses while the operator selects one or more locations. Registration and termination of location selections is postprocessor defined. After termination the LOCATE subroutine is called repeatedly until all (or as many as the programmer wants) of the locations are reported to the application program.

PRELOC prepares the mode set locator for use in reporting locations. The locator device is not displayed until a subsequent call to LOCATE. If there is a current LOCATE queue, it is cleared.

The xxLOCR mode/attribute/feature routines control testing and setting of the virtual locator device used with the EVENT and LOCATE routines. Assignment of physical input device to the virtual device is postprocessor determined.

INTERACTION ROUTINES

EVENT

Format

EVENT(**lky,ids,coords,iremng**)

Report terminal input to application program.

Parameters

lky Output parameter; indicates whether event was a function key press or segment pick.

If LKY=.TRUE., event was a function key press.

If LKY=.FALSE., event was a segment pick.

ids Output parameter; five-word array containing information about the event reported by the current call to EVENT.

If event was a function key press (LKY=.TRUE.), IDS(1) contains the function key ID; 0≤ID≤255.

The other elements of array **ids** have no meaning, and the **coords** parameter has no meaning.

If event was a segment pick (LKY=.FALSE.), array **id** has the following significance.

Array Element	Contents
1	ID of picked segment
2	ID of window in which segment is displayed
3	ID of picture containing segment
4	ID of viewport in which segment is displayed
5	Intrasegment identifier (zero if no intrasegment ID)

coords	Output parameter; three-word array containing the best effort coordinate values of the location of the segment pick. COORDS(1) = x coordinate COORDS(2) = y coordinate COORDS(3) = z coordinate (zero for 2-D picks)
iremng	Output parameter; the number of events remaining on the event queue for the current set of events.

Programming Notes

When a call to EVENT is encountered and the event queue is empty:

- The application program pauses to permit the terminal operator to make segment picks or function key presses. A postprocessor-defined prompt is initiated (crosshairs, cursor, etc.) to inform the operator that the program is waiting for input.
- Selections can be made until a segment or function key in the terminate action type is selected.
- Control is then returned to the calling program.

Events on the event queue are processed by repeated calls to EVENT, one call per event on the event queue. When IREMNG=0, **lky**, **ids**, and **coords** contain the information about the last event on the queue.

A function key press event is registered by pressing the key. A segment pick event is registered by moving the locator device until it is on some line of the desired segment (not a space enclosed by the segment). For one-shot locators (refer to SMLOCR), the operator must also register the pick.

If no segment or function key has the terminate action associated with it, the subroutine can never return control to the calling program.

If there are events already on the event queue when EVENT is called, there is no program prompt. The events already on the queue are processed. The **iremng** parameter informs the programmer of the number of events remaining on the queue. The PREEVN subroutine is used to clear the event queue when necessary.

The exact coordinate values returned for the **coords** parameter are postprocessor-dependent, but are typically the endpoint of a picked line, arc, arc chord or segment, a point, and so on. The coordinate system used in reporting the values is the one specified in the most recent call to PREEVN.

KEYBRD

Format

KEYBRD(**maxchr,nchrs,itext**)

Return to application program the text string entered from the terminal alphanumeric keyboard.

Parameters

maxchr	Input parameter; maximum number of characters permitted in string.
nchrs	Output parameter; number of characters contained in array **itext** up to and including the last nonblank character: **nchrs** ≤ **maxchr**
itext	Output parameter; first word of array containing the text string.

Programming Notes

KEYBRD reports text strings entered by the terminal operator. When a call to KEYBRD is encountered, the cursor appears in the system viewport area and the operator enters the desired character string. The character string is terminated in the same way normal terminal input is (RETURN, SEND, ETX key, and so on) and control is returned to the calling program.

If a text string larger than **maxchr** is entered at the terminal, only **maxchr** characters are reported to the application program. If a text string shorter than **maxchr** is entered at the terminal, the remaining character positions are blank filled.

As many characters as possible are packed into a word; the number of characters depends on the processor word length. Characters are in the character set of the processor.

The maximum number of characters which can be input with one KEYBRD call depends on the postprocessor. However, each postprocessor will allow at least 50 characters.

When numerical data is input via KEYBRD, the data must be reformatted before any calculations can be made. The FORTRAN DECODE statement, although non-ANSI standard, will reformat the data as needed. (Refer to the FORTRAN Manual for information.)

KYAC

Format

KYAC(**idky,iactn**)

Assign individual function key to an action type.

Parameters

idky	Input parameter; number of the function key to be assigned:

$0 \leq$ **idky** ≤ 255

Number of available function keys is postprocessor-dependent.

iactn	Input parameter; action to be performed when **idky** is pressed:

IACTN = 1	Ignore.
IACTN = 2	Recognize.
IACTN = 3	Terminate.

DEFAULT is IACTN = 3, terminate.

Programming Notes

KYAC assigns individual function keys to action types for use with the EVENT subroutine. KYON and KYOFF can be used to assign all keys at one time.

NOTE

So that application programs can be to the greatest practical extent terminal-independent, it is strongly recommended that the programmer use only function keys 0 to 9 which always exist or will be simulated for any terminal supporting the EVENT subroutine.

Refer to the general heading in this section for more information on action types.

KYOFF

Format

KYOFF

Assign all function keys to ignore action type.

Parameters

None.

Programming Notes

KYOFF assigns all function keys to the ignore action type. The number of function keys is postprocessor-dependent.

KYON assigns all function keys to the terminate action type. In the default case (no calls to KYON or KYOFF), all function keys are assigned to the terminate action type.

For example, the following sequence of calls turns off all keys except 9, which is assigned to the terminate action type.

```
CALL KYOFF

CALL KYAC (9,3)
```

Refer to KYON.

KYON

Format

KYON

Assign all function keys to terminate action type.

Parameters

None.

Programming Notes

KYON assigns all function keys to the terminate action type. The number of function keys is postprocessor-dependent.

KYOFF assigns all function keys to the ignore action type. In the default case (no calls to KYON or KYOFF), all function keys are assigned to the terminate action type.

Refer to KYOFF.

LOCATE

Format

LOCATE(**x,y,iremng**)

Report one or more sets of locator symbol coordinates.

Parameters

x,y	Output parameters; coordinates of the locator symbol.
iremng	Output parameter; number of locations left on location queue.

Programming Notes

This subroutine reports one or more sets of locator symbol screen coordinates. When a call to LOCATE is encountered and the location queue is empty:

- The mode set locator is enabled (refer to SMLOCR).
- The locator symbol is displayed at the center of the window or viewport specified by the most recent call to PRELOC, or at the coordinates of the location when LOCATE was last terminated, if there is no intervening PRELOC call.†
- Locations can be entered until terminated by the operator; control is then returned to the calling program.

Locations on the location queue are then processed by repeated calls to LOCATE, one call per location on the location queue. When IREMNG=0, **x** and **y** contain information about the last set of coordinates picked.

Locations are selected at the terminal by moving the postprocessor-determined locator symbol (crosshairs, tracking cross, etc.) to the desired location. One-shot locators require that each location be separately registered (refer to SMLOCR). The current set of locations is terminated by the postprocessor-determined terminator.

If the set of locations is not terminated, control cannot be returned to the calling program.

If there are locations already on the location queue when LOCATE is called, the mode set locator is not enabled; the locations already on the queue are processed. The **iremng** parameter informs the programmer of the number of locations remaining on the queue. The PRELOC subroutine is used to clear the location queue when necessary.

LOCATE returns locator coordinates based on the **lucord** parameter of the PRELOC subroutine. If LUCORD=.TRUE., coordinates are returned as user coordinates. User coordinate ranges are established by calling SMPLIM. Only two-dimensional window IDs can be specified in PRELOC when LUCORD=.TRUE.; three-dimensional windows result in diagnostics. If LUCORD=.FALSE., coordinates are returned as screen coordinates. Screen coordinate range is established when INITIG is called. When LUCORD=.FALSE., the locator is associated with a viewport, not a window. Either two- or three-dimensional windows may be associated with a viewport. Only when LUCORD=.FALSE. can LOCATE be used with three-dimensional pictures.

PREEVN

Format

PREEVN(**lucord**,**idvuwi**)

Specify coordinate system and window/viewport ID for EVENT processing.

Parameters

lucord	Input parameter; specify coordinate system for returned coordinates and the type of ID specified by **idvuwi**. If LUCORD=.TRUE., coordinates are returned as user coordinates; **idvuwi** specifies the ID of a window. If LUCORD=.FALSE., coordinates are returned as screen coordinates; **idvuwi** specifies the ID of a viewport.
idvuwi	Input parameter; the ID of a window or viewport in which initially to display the locator symbol. The ID is for a window or viewport depending on the value given for **lucord**.

Programming Notes

This subroutine is used to prepare the locator device for a subsequent call to EVENT. PREEVN specifies the window or viewport in which initially to display the locator symbol. (For some postprocessors, some locator devices cannot be assigned in this way; their initial location is determined solely by conditions at the terminal and are not influenced by TIGS software. Refer to the appropriate postprocessor appendix for more information.) The coordinate system in which pick coordinates are returned to the calling program is specified by PREEVN: screen coordinates if LUCORD=.FALSE., user coordinates if LUCORD=.TRUE..

PREEVN clears any events remaining on the event queue.

The parameters apply to the mode set locator used as the pick device when it is next enabled by a call to EVENT.

If PREEVN is not called prior to the first call to EVENT, defaults are as follows:

LUCORD=.FALSE.

IDVUWI=0 (default viewport)

PRELOC

Format

PRELOC(**lucord**,**idvuwi**)

Specify coordinate system and window/viewport ID for LOCATE processing.

Parameters

lucord	Input parameter; specify coordinate system for returned coordinates and the type of ID specified by **idvuwi.** If LUCORD=.TRUE., coordinates are returned as user coordinates; **idvuwi** specifies the ID of a window. If LUCORD=.FALSE., coordinates are returned as screen coordinates; **idvuwi** specifies the ID of a viewport.

†Some locators (for example, thumbwheels) cannot be software-positioned; they appear where last positioned by terminal locator controls.

idvuwi	Input parameter; the ID of a window or viewport in which initially to display the locator symbol. The ID is for a window or viewport depending on the value given for **lucord**.

Programming Notes

This subroutine is used to prepare the locator device for a subsequent call to LOCATE. PRELOC specifies the window or viewport in which initially to display the locator symbol. (For some postprocessors, some locator devices cannot be assigned in this way; their initial location is determined solely by conditions at the terminal and are not influenced by TIGS software. Refer to the appropriate postprocessor appendix for more information.) The coordinate system in which locations are returned to the calling program is specified by PRELOC: screen coordinates if LUCORD=.FALSE., user coordinates if LUCORD=.TRUE..

PRELOC clears any locations remaining on the location queue.

The parameters apply to the mode set locator when it is next enabled by a call to LOCATE.

If PRELOC is not called prior to the first call to LOCATE, defaults are as follows:

LUCORD=.FALSE.

IDVUWI=0 (default viewport)

PROMPT

Format

PROMPT(**nchar,itext**)

Display a message in the system viewport area of the screen.

Parameters

nchar	Input parameter; number of characters including blanks in the message: $0 \leq$ **nchar** ≤ 256
itext	Input parameter; name of array in which prompting message is stored.

Programming Notes

For more information on the system viewport area, refer to SMSVP, section 4. Messages in the system viewport area are not redrawn when the screen is redrawn, as segments containing text are.

Before numerical information can be output via PROMPT, the data must be reformatted. The FORTRAN ENCODE statement, although non-ANSI standard, will reformat the data as needed. (Refer to the FORTRAN Reference Manual.)

MODE/ATTRIBUTE/FEATURE ROUTINES

xxAC

Format

TFAC(**lactn**)

Test for interaction support by postprocessor.

SMAC(**iactn**)

Modally set action type.

TMAC(**iactn**)

Test the current modally set action type.

RAAC(**idseg,iactn**)

Reset the action type attribute of a segment.

TAAC(**idseg,iactn**)

Test the action type attribute of a segment.

Parameters

lactn	Output parameters; if LACTN=.TRUE., then the EVENT subroutine is supported by the postprocessor; otherwise, it is not.
iactn	Input parameter (SMAC and RAAC) or output parameter (TMAC and TAAC) specifying the action type: IACTN = 1 Ignore. IACTN = 2 Recognize. IACTN = 3 Terminate. DEFAULT is IACTN = 3, terminate.
idseg	Input parameter identifying the segment whose attribute is to be tested or reset.

Programming Notes

TFAC tests for postprocessor support of the EVENT subroutine for basic interaction; it does not imply anything regarding LOCATE support.

SMAC modally sets the action type to which subsequently defined segments are assigned. The action type modal setting cannot be changed while a segment is open; that is, from the time the first primitive in a segment is defined until the segment is closed.

xxID

Format

TFID(**nid**)

Test for postprocessor support of return of intrasegment identifier.

SMID(**idintr**)

Modally set intrasegment identifier.

TMID(**idintr**)

Test for current modally set intrasegment identifier.

Parameters

nid	Output parameter; the largest number supported by the postprocessor for intrasegment identifiers; if NID=0, the feature is not supported.
idintr	Input parameter (SMID) or output parameter (TMID) specifying the intrasegment identifier: $0 \le$ **idintr** $\le 32{,}767$ If **idintr** > **nid**, **nid** is used.

Programming Notes

There are no reset attribute (RA) or test attribute (TA) routines because intrasegment identifiers are not attributes of segments; a segment may not have a single intrasegment identifier which could be tested or reset.

The xxID routines are primarily for use in conjunction with the EVENT subroutine.

An intrasegment identifier is an ID assigned to a part of a segment. A single segment may be composed of several component parts, each of which may be assigned a separate intrasegment identifier. Although a segment can only be picked as a whole, intrasegment identifiers can inform the application program of which component the locator device was on when the segment was picked.

SMID is a mode-setting routine which, necessarily, can be set during a segment definition.

xxINFO

Format

SMINFO(**ninfo,info**)

Modally set information stored with segments.

TMINFO(**ninfo,info**)

Test current modal setting for information stored with segment.

RAINFO(**idseg,ninfo,info**)

Reset the application information attribute of a segment.

TAINFO(**idseg,ninfo,info**)

Test the application information attribute of a segment.

Parameters

ninfo	Input parameter; the number of words in the **info** array: $0 <$ **ninfo** < 4 DEFAULT, NINFO=0
info	Input parameter (SMINFO and RAINFO) or output parameter (TMINFO and TAINFO); name of array containing or to contain the application information.
idseg	Input parameter; ID of the segment whose attribute is to be tested or reset.

Programming Notes

There is no test feature (TF) routine; all postprocessors support this feature.

Application-related information is contained in a free-field storage area that becomes an attribute of a segment and part of its definition. There are no restrictions on what the programmer places in this storage, other than the physical ones outlined in the following paragraphs. It need not contain interactive information, but this subroutine will be most often used to store information regarding interaction.

ninfo cannot be greater than four words and cannot subsequently be increased above the number specified when the segment was defined.

For purposes of terminal independence, it is strongly suggested that the info array be constructed of 16-bit, right-justified data parcels. Although some postprocessors may handle words of greater length than 16 bits, all will handle at least 16-bit words.

xxLOCR

Format

TFLOCR(**maxloc,nlocrs,descrp,lone**)

Test for number of locators (if any) supported and what their characteristics are.

SMLOCR(**ilocr**)

Modally set the locator device to be used.

TMLOCR(**ilocr**)

Test for current modally set locator device.

Parameters

maxloc	Input parameter; size of the arrays **descrp** and **lone.**
nlocrs	Output parameter; the total number of locator devices supported by the post-processor up to **maxloc**: $0 \leq$ **nlocrs** $\leq$ **maxloc**
descrp	Output integer array of size **nlocrs** describing the **nlocrs** locators for this postprocessor. The information for a given entry describes the type of locator.
lone	Output logical array of size **nlocrs**; if LONE(I)=.TRUE., then DESCRP(I) is a one-shot locator (refer to the following programming notes).
ilocr	Input parameter (SMLOCR) or output parameter (TMLOCR); specifies locator. Ordinals specify corresponding members of **descrp** array; that is, ILOCR=1 specifies locator described in DESCRP(1). If **ilocr** > **nlocrs**, the highest numbered locator is used. DEFAULT is ILOCR=1.

Programming Notes

There are no reset attribute (RA) or test attribute (TA) routines because the locator cannot become an attribute of a segment.

The TFLOCR routine returns to the application program essentially the same information as is contained in the locator section of the postprocessor appendix for a given terminal. The choice of a terminal locator device from within an application program is postprocessor-dependent because the programmer must know the various codes returned by the various postprocessors to select the desired locator device.

TFLOCR informs the application program of whether or not location operations are possible; if NLOCRS=0, no such operations are possible. Also, TFLOCR informs the application program of the current locator device status of a variable configuration terminal. For example, for a terminal with an optional tablet input device, **nlocrs** and **descrp** report whether or not this device is in use in a given terminal session.

The **maxloc** parameter establishes a maximum array size for **descrp** and **lone.** If the terminal has a greater number of locator devices than there are words in **maxloc**, only a portion of the total locator device information is returned to the application program; that is, only **maxloc** locators will be described.

Each significant word of array **descrp** contains the right-justified code equivalent for a locator device available at a given terminal. These codes are explained in the appropriate postprocessor appendix.

lone is an array of the same size as **descrp.** For each entry in **descrp**, the corresponding entry in **lone** specifies whether the locator is a one-shot type. A one-shot locator is a locator in which the terminal operator must manually register each location as he selects it as opposed to a stream of points based on continuous movement of the locator (for example, the thumbwheel/crosshair locator is a one-shot locator device; most lightpens are not). The programmer should keep in mind that such locators are cumbersome when they must be used to select a large number of points. The human interaction characteristic should be a factor weighed in the decision to use or not use a particular locator device.

ilocr must be in the range:

$0 \leq$ **ilocr** ≤ 63

If ILOCR=0, all calls to LOCATE will produce errors.

EXAMPLE OF INTERACTION ROUTINE USAGE

The following example uses many of the interaction routines documented in this section. It is not a complete program.

WARNING

Refer to appropriate postprocessor and operating system appendices for format of PROGRAM statement.

In this example, six circles are initially displayed on the screen, in addition to the light buttons (text segments) ADD, DELETE, and STOP (figure 7-1). Selection of the ADD button enables the operator to select a location for a new circle. If the operator selects DELETE, any or all of the circles can be deleted from the picture. When DELETE is selected, an ACCEPT lightbutton appears so that the operator can inform the program that he has selected the circles to be deleted. The STOP button stops this sample program.

PROGRAM statement	
DIMENSION IDS(5), CORDS(3)	
LOGICAL LVIS, LUSER, LKY	
DATA LVIS/.TRUE./,LUSER/.TRUE./	
CALL INITIG(.TRUE.,.TRUE.,5LIFILE)	
CALL SMPLIM(-6.,-6.,6.,6.)	
CALL PRELOC(LUSER,0)	Prepare locator; it is assigned to window 0 (default window) and coordinates are user coordinates.
CALL CIRCLE(-2.,4.)	Call program subroutine to draw 6 circles. Each circle is assigned to the recognize action type (2) in the subroutine. Note that when control is returned to the main program, the modally set action type is still 2.
CALL CIRCLE(-2.,0.)	
CALL CIRCLE(-2.,-4.)	
CALL CIRCLE(4.,4.)	
CALL CIRCLE(4.,0.)	
CALL CIRCLE(4.,-4.)	
CALL SMAC(3)	Create ADD, DELETE, STOP, and ACCEPT light buttons; assign each light button to the terminate action type.
CALL OPNSEG(1)	
CALL MOVEA(-5.75,5.)	
CALL TEXT(3,3HADD)	
CALL CLSSEG	
CALL OPNSEG(2)	
CALL MOVEA(-5.75,3.)	
CALL TEXT(6,6HDELETE)	
CALL CLSSEG	
CALL OPNSEG(3)	
CALL MOVEA(-5.75,1.)	
CALL TEXT(4,4HSTOP)	
CALL CLSSEG	
CALL SMVIS(.NOT.LVIS)	Create ACCEPT but make it invisible until after the operator selects DELETE.
CALL OPNSEG(4)	
CALL MOVEA(-5.75,-1.)	
CALL TEXT(6,6HACCEPT)	
CALL CLSSEG	
CALL SMVIS(LVIS)	Visibility mode must be set back to visible so subsequent segments are visible.

```
10      CALL EVENT(LKY,IDS,CORDS,NREM)
```

Wait for event; successive IF statements ensure that ADD, DELETE, or STOP must be selected for program to proceed.

```
        IF(LKY) GO TO 10
        IF (IDS(1).LT.1.OR.IDS(1).GT.3) GO TO 10
        GO TO (100,200,300) IDS(1)
100     CONTINUE
110     CALL LOCATE(X,Y,NREM)
```

Select locations for additional circles. Operator must terminate location queue with postprocessor-determined termination procedure. Then the program loops to process all locations on location queue, one call to LOCATE per location. Program returns for next event.

```
        CALL CIRCLE (X,Y)
        IF (NREM.NE.0) GO TO 110
        GO TO 10
200     CONTINUE
        CALL RAVIS (4,LVIS)
```

ACCEPT light button is made visible.

```
210     CALL EVENT (LKY,IDS,CORDS,NREM)
```

Operator selects circles to be deleted. Note that since circles are associated with the recognize action, a segment associated with the terminate action must be picked to terminate the event queue and return control to the program. ACCEPT is used for this, although in this program any light button (or function key) could be used.

```
        IF (NREM.EQ.0) GO TO 220
        CALL DELSEG (IDS(1))
        GO TO 210
220     CONTINUE
        CALL RAVIS (4,.NOT.LVIS)
        GO TO 10
300     CALL QUITIG(.TRUE.)
        END
        SUBROUTINE CIRCLE (X,Y)
```

Subroutine to draw circles.

```
        DATA ID/100/
```

New segment IDs start at 100.

```
        CALL SMAC(2)
        CALL OPNSEG(ID)
        ID=ID+1
        IF (ID.EQ. 32767) STOP
        CALL MOVEA (X+.5, Y+.5)
        CALL ARCA (X,Y,X+.5, Y+.5)
        CALL CLSSEG
        RETURN
        END
```

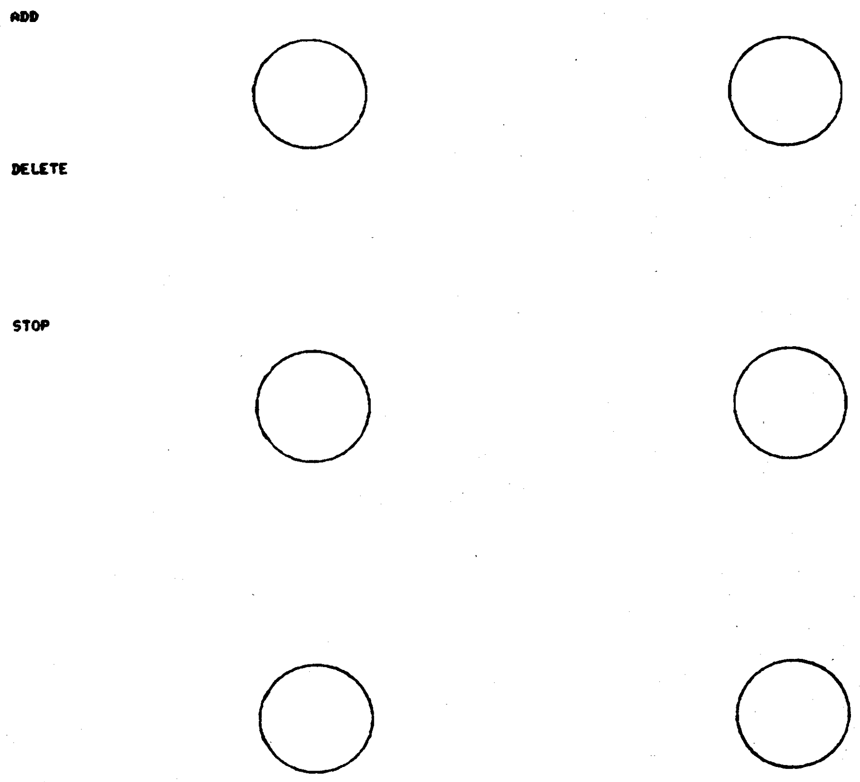


Figure 7-1. Output from Interaction Routines Usage Example

TERMINAL FUNCTIONS

The following TIGS calls are documented in this section.

ALARM	QUITIG	TFSCRN
CLRSCR	REMSCR	UDATA
DSPLAY	SCRNUR	UNISCR
INITIG	TFHARD	WHERE
		WHERE3

The two feature testing routines, TFHARD and TFSCRN, are documented following the terminal function routines.

GENERAL

The routines in this section can be treated in several groups. The first group is composed of INITIG and QUITIG. These two routines are used to initiate and terminate a graphics program and should be the first and last graphics calls, respectively, in the program.

In the second group are the routines DSPLAY, CLRSCR, REMSCR, SCRNUR, UNISCR, WHERE, and WHERE3. These routines affect or report on displays at the terminal screen.

In the third group are TFHARD and TFSCRN which test terminal features.

A last miscellaneous group is composed of ALARM and UDATA. ALARM is used by the application program to alert the terminal operator. UDATA is used to include non-TIGS data in the neutral display file.

TERMINAL FUNCTION ROUTINES

ALARM

Format

ALARM(**lon**)

Turn terminal alarm indicator on or off.

Parameters

lon	Input parameter; turn alarm on or off.
	If LON=.TRUE., turn alarm on.
	If LON=.FALSE., turn alarm off.

Programming Notes

The minimum terminal configuration is assumed to have a bell which is activated when the terminal receives the ASCII bell character as part of the program output. For this terminal, ALARM (.TRUE.) produces a single ASCII bell character; ALARM (.FALSE.) does nothing. This type of terminal requires a series of calls to ALARM to produce a sustained tone. For other terminals capable of a sustained alarm, the alarm must be turned on and off. For some terminals, lights may be used instead of an audible alarm.

CLRSCR

Format

CLRSCR

Clear all displays from terminal screen.

Parameters

None.

Programming Notes

This routine clears the display screen of all graphics and system output. Nothing in the neutral display file is deleted; the material can be redisplayed, if desired.

DSPLAY

Format

DSPLAY

Display all pictures.

Parameters

None.

Programming Notes

This subroutine causes all currently defined pictures to be displayed on the terminal screen. The only exceptions are for segments which have a visibility attribute of .FALSE. (refer to xxVIS, section 2) and pictures with blinds down (refer to BLINDS, section 3).

The programmer must make provisions to display his pictures. Any of the following routines force the terminal display to be updated to reflect the current contents of the neutral display file.

DSPLAY
EVENT
KEYBRD
LOCATE

If a graphics program does not contain a call to any of these subroutines, nothing will be displayed on the terminal screen.

INITIG

Format

INITIG(**lsquar**,**lnwfil**,**filnam**)

Set all TIGS and terminal conditions to initial values.

Parameters

lsquar	Input parameter; define mapping of screen coordinates onto terminal display surface. If LSQUAR=.TRUE., screen coordinates will be mapped onto a square portion of the terminal display surface and there will be a separate area of the screen for the system viewport. If LSQUAR=.FALSE., screen coordinates will be mapped onto the entire terminal display surface. In this case the system viewport will overlap with the graphics display area. DEFAULT is LSQUAR=.TRUE.
lnwfil	Input parameter; specifies whether the neutral display file is new or old. If LNWFIL=.TRUE., a new neutral display file will be created by this graphics program. If LNWFIL=.FALSE., this program will use an old neutral display file (refer to the following programming notes). DEFAULT is LNWFIL=.TRUE.
filnam	Input parameter; specifies the name of the neutral display file. DEFAULT is postprocessor-dependent.

Programming Notes

INITIG must be the first graphics routine called. All TIGS modes are set to their default values by INITIG.

INITIG can be called at any point in a program. It is useful in any situation in which it is desirable to reset all modal settings to default values (refer to QUITIG).

The **lsquar** parameter specifies whether the display will use a square portion of the terminal display surface, or all available surface. When all available terminal display surface is used, portions of the screen coordinate system may liew outside the limits of the terminal display surface resulting in the possible loss of part of the display (for example, when the mapping is done to a round display surface). When LSQUAR=.TRUE., a terminal-independent screen coordinate system is used when specifying viewports, returning screen coordinates to the application program, and so on. This coordinate system ranges from (0,0) at the lower left corner to (1,1) at the upper right. When LSQUAR=.FALSE., the screen coordinate system is postprocessor-dependent; refer to the appropriate postprocessor appendix for the specific range. For reasons of terminal independence, it is urged that **lsquar** be set to .TRUE..

Only when LSQUAR=.TRUE. can the system viewport be guaranteed to be separate from the graphics working area (refer to SMSVP, section 4).

The **lnwfil** parameter specifies whether or not a new neutral display file is to be created by the graphics application program. If LNWFIL=.TRUE., the **filnam** parameter gives the name of the new neutral display file. A new neutral display file must not have the same name as an existing local file or conflicts will occur. If LNWFIL=.FALSE., the **filnam** parameter gives the name of an existing neutral display file (created by a previous graphics application program). It is the operator's responsibility to ensure that this neutral display file has been attached to the terminal before the application program is executed. It is also the operator's responsibility to extend, if necessary, the permanent file if an old neutral display file is used and modifications have been made. (Refer to the appropriate operating system reference manual for details.) All picture, window, segment, and viewport definitions are preserved in an old neutral display file, but all modal settings are reset to default values by the INITIG call. This capability to use existing neutral display files is especially useful where many application programs use a complex picture, window, and viewport scheme. It is only necessary to define this scheme once and preserve the resulting neutral display file. Then succeeding programs specify that file in the INITIG call and display new segments in a standardized picture/window/viewport arrangement. It is the programmer's reponsibility to be familiar with the segment, picture, window, and viewport definitions in an existing neutral display file.

The **filnam** parameter specifies the name of the neutral display file, either new or old. The name of the file is specified in one of two ways.

The programmer may simply specify an integer in the range 1 through 60 and TIGS will interpret it as a file of the form TAPE(integer), which is an ANSI standard file name.

For example, the call

```
INITIG(LSQUAR,.TRUE.,58)
```

results in the creation of a neutral display file named TAPE58. The call

```
INITIG(LSQUAR,.FALSE.,58)
```

results in TIGS searching for a neutral display file named TAPE58.

The programmer can also specify a file name, 1 to 7 characters, left-justified and zero-filled. The call

```
INITIG(LSQUAR,.FALSE.,7LOLDFILE)
```

results in TIGS searching for a neutral display file named OLDFILE.

With each neutral display file, TIGS associates the version number of the TIGS program library under which it was created (refer to appendix E). If a program specifies that an old neutral display file is to be used but the library version number associated with the old file does not correspond to the version number of the TIGS program library in use, TIGS issues an error message and creates a new file with the old file name.

NOTE

If a program aborts prematurely or if it has not called QUITIG, the operator must return the neutral display file before running a TIGS application with the same neutral display file name.

Refer to QUITIG for information on how to save neutral display files for future use.

NOTE

Neutral display files created with TIGS v.1.0 must undergo a conversion. Refer to the operating system dependencies section for information on the control language statements needed to convert a 1.0 file to 1.1 format.

QUITIG

Format

QUITIG(**ldelet**)

Ensure orderly shutdown of terminal at end of program.

Parameters

ldelet	Input parameter; specifies whether neutral display file used by program is to be saved or discarded when QUITIG is called. If LDELET=.TRUE., neutral display file is discarded. If LDELET=.FALSE., neutral display file is not discarded. DEFAULT is LDELET=.TRUE..

Programming Notes

QUITIG is called to terminate a graphics program in an orderly fashion. QUITIG can be called at any point in a graphics program: the combination of calls

CALL QUITIG(**ldelet**)

CALL INITIG(**lsquar,lnwfil,filnam**)

is used mid-program to clear all modal settings in effect and reestablish default values for them. By specifying the proper parameter values in the above sequence (LNWFIL=.TRUE.), all current segment, picture, window, and viewport definitions may also be cleared.

Neutral display files are saved for future use by specifying LDELET=.FALSE. when QUITIG is called. This means that the file space occupied by the neutral display file is not returned to the system. It is the operator's responsibility to make provisions to save the file at job end.

The neutral display file is discarded (file space is returned to the system) by specifying LDELET=.TRUE.. However, if the neutral display file is not in the proper format, it is not discarded, even when LDELET=.TRUE.. Thus, for example, if the programmer should mistakenly specify a source file rather than an existing neutral display file in the INITIG routine, the source file is not discarded when the program terminates.

REMSCR

Format

REMSCR

Copy contents of screen on remote hardcopier.

Parameters

None.

Programming Notes

REMSCR causes a copy of the terminal display screen to be made at a remote hardcopier device. Implementation is of course dependent on the presence of such a device. Whether or not the system viewport area is copied is postprocessor-dependent.

SCRNUR

Format

SCRNUR(**idwind,xscrn,yscrn,xuser,yuser**)

Convert screen coordinates to user coordinates; 2-D pictures only.

Parameters

idwind	Input parameter; ID of 2-D window to whose coordinate system conversion is to be made.
xscrn,yscrn	Input parameters; screen coordinates to be converted.
xuser,yuser	Output parameters; user coordinates converted from screen coordinates.

Programming Notes

This subroutine will extrapolate user coordinates outside the specified window if necessary. For example, in figure 8-1, a window with IDWIND=1 is displayed in a viewport on the left half of the terminal screen. If a screen location on the right half of the screen (the x) is specified, with the ID of the window on the left:

```
CALL SCRNUR (1,.9,.9,XUSER,YUSER)
```

the values for XUSER and YUSER will fall outside the window limits.

This subroutine can be useful in calculating user coordinates from values returned by LOCATE, when LOCATE has been set up to return screen coordinates.

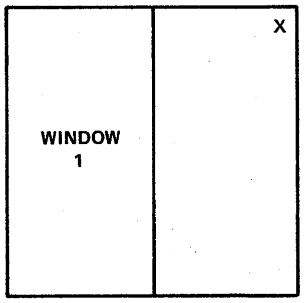


Figure 8-1. SCRNUR Calculation

UDATA

Format

UDATA(**nwords,idat**)

Place user data in neutral display file.

Parameters

nwords	Input parameter; number of words in the idat array: $0 \leq$ **nwords** ≤ 26
idat	Input parameter; array containing user data.

Programming Notes

This subroutine permits the programmer to place program-defined data in the neutral display file to be used for some non-TIGS function. The data is ignored by TIGS. The programmer must make his own arrangements to use the data once it has been placed in the neutral display file. The data is stored in the currently open segment.

When this UDATA data is encountered by the postprocessor in processing the NDF for creating terminal displays, the postprocessor hands control back to the application program, passes the UDATA information to the application program, and waits while the application program processes the information. Control is passed to a user subroutine which must be called USRDAT. Its format is as follows:

```
SUBROUTINE USRDAT(NWORDS,ARRAY)
```

where ARRAY is the array into which the data from UDATA will be placed, and NWORDS is the number of words in this array. The application program then processes the data and returns control to the postprocessor.

Note that care should be exercised when using USRDAT. It is not advisable to call other TIGS routines from USRDAT particularly if the segment or overlay loader is being used.

In the interest of terminal independence, it is strongly suggested that the programmer use only the rightmost 16 bits of each word of **idat**. All postprocessors will support a minimum of 16 bits per word.

UNISCR

Format

UNISCR	Create picture on UNIPLOT NPFILE to reflect current state of neutral display file.

Parameters

None.

Programming Notes

Upon completion of the TIGS run a hardcopy plot can be made of all pictures on the NPFILE (neutral picture file) via UNIPOST. Refer to the UNIPLOT Reference Manual for information on UNIPLOT, UNIPOST and the NPFILE. Of course, this routine can only be used on systems which have the UNIPLOT package.

Since UNIPLOT creates a local file named NPFILE, the user should not have a local file named NPFILE at the time the UNISCR call is made.

TIGS does not update the screen in conjunction with a UNISCR call.

When running an application which uses the UNISCR call, declare the TIGS library first and the UNIPLOT library second. Refer to the operating systems dependencies appendix for information on the TIGS library, and refer to the UNIPLOT Reference Manual for information on the UNIPLOT library.

WHEREx

Format

WHERE(**x,y**)

Obtain current beam position for 2-D picture.

WHERE3(**x,y,z**)

Obtain current beam position for 3-D picture.

Parameters

x,y,z	Output parameters; coordinates of drawing beam.

Programming Notes

WHEREx returns the current drawing beam position to the calling program. Coordinates returned are user coordinates established by calling SMPLIx (section 3).

TEST FEATURE ROUTINES

TFHARD

Format

TFHARD(**lremot**)

Test for remote hardcopier availability.

Parameters

lremot	Output parameter; if LREMOT=.TRUE., a remote hardcopier exists.

Programming Notes

If LREMOT=.TRUE., a REMSCR call will copy the contents of the screen to the hardcopy device.

TFSCRN

Format

TFSCRN(**lrtang,xll,yll,xur,yur,resltn**)

Test the size, shape, and resolution of the terminal screen.

Parameters

lrtang,xll,yll,xur,yur	Output parameters describing screen shape and dimensions. If LRTANG=.TRUE., the terminal screen is rectangular and **xll,yll,xur,yur** give the lower left and upper right corners of the screen, expressed in a terminal-dependent coordinate system. If LRTANG=.FALSE., the screen is assumed to be circular and **xll,yll** give the screen center, **xur** the radius of the largest viewable circle on the screen, and **yur** is meaningless. All values are expressed in a terminal-dependent coordinate system.
resltn	Output parameter; a value expressing the ability of the terminal screen to resolve images (refer to the following programming notes).

Programming Notes

The **lrtang**,**xll**,**yll**,**xur**, and **yur** parameters give the size and shape of the terminal screen. The only general use of these values which is compatible with the terminal-independent aim of TIGS is in setting up a system viewport (refer to SMSVP, section 4). All terminals used with TIGS can make use of the terminal independent screen coordinate system established in a call to INITIG. Specific values for the various terminal types can also be found in the appropriate postprocessor appendixes.

The **resltn** parameter indicates the ability of the terminal screen to resolve images; that is, the ability of the terminal screen to distinguish portions of a graphics display which are close to each other. The higher the value for **resltn**, the better the ability of the terminal to resolve images. For example, if RESLTN=780., the screen may be thought of as divided into 780 units; portions of the user's display lying less than 1/780 of the screen width of each other will not be distinguished from each other. If RESLTN=4095., portions of the display lying less than 1/4095 of the usable width from each other will be displayed as one item; resolution is much greater in this case.

The value for **resltn** depends on the value given for the **lsquar** parameter of INITIG as well as on the terminal type. Specific values for the various terminals for either value of **lsquar** can also be found in the appropriate postprocessor appendix.

ERROR PROCESSING 9

The following TIGS routines are documented in this section.

IERROR
SMERR
TFERR
TMERR

This list includes the routine which checks error status, and the mode/feature routines which set and test a user error routine. The mode/feature routines are placed in a separate group following the error status routine.

This section does not include a compilation of error messages produced by TIGS. Error messages are found in appendix C.

GENERAL

Part of the design philosophy of TIGS is to continue program execution despite TIGS programming errors. TIGS logs all errors on an error file and attempts to continue execution. (Program execution is terminated if errors 4104-4106 are generated.) The programmer has the option of providing more selective error processing capabilities in his program by using the routines in this section.

SMERR allows the programmer to provide the address of a programmer-written error routine to which TIGS jumps any time it encounters an error in processing. TMERR tests for the address of this routine. TFERR tests for postprocessor support of user error routines.

IERROR is called to check current error status. It returns the error number of the error (if any) produced by the last encountered TIGS call.

ERROR STATUS ROUTINE

IERROR

Format

IERROR(**ierr**)

Check current error status.

Parameters

ierr	Output parameter; TIGS error number.
	If IERR=0 [IERROR(IERR)=0 if called as function], no error was detected.
	If IERR≥1[IERROR(IERR)≥1 if called as function], IERR contains the error number of the detected error.

Programming Notes

IERROR can be called either as a subroutine or as a function. In either case, the error detected (if any) by the most recently called TIGS routine is returned to the application program. Based on the information returned to the program, a choice can be made regarding further error processing (refer to SMERR).

Whether or not this routine is called, all error messages are logged on the error file and then the modally set error routine is called (refer to SMERR). The name and format of the error file is postprocessor-dependent; refer to the appropriate postprocessor appendix for more information.

MODE/FEATURE ROUTINES

xxERR

Format

TFERR(**lroutn**)

Test for postprocessor support of a user-supplied error routine.

SMERR(**routin**)

Modally set error processing routine to be used when an error is detected.

TMERR(**routin**)

Test the current modally set error processing routine.

Parameters

lroutn	Output parameter; if LROUTN=.TRUE., then a user-supplied error routine is supported; otherwise, it is not.
routin	Input parameter (SMERR) or output parameter (TMERR) specifying the address of the error routine.
	DEFAULT is an error routine named NULL.

Programming Notes

There are no reset attribute (RA) or test attribute (TA) routines because an error routine cannot become an attribute of a segment.

SMERR allows the programmer to supply the address of his own error routine to be used whenever TIGS encounters an error. In conjunction with IERROR, this routine allows the programmer to respond selectively to encountered errors. If the programmer does not supply an error routine, TIGS attempts to resume execution at the next statement after the call that detected the error (default error processing).

When the programmer uses either the overlay loader or the segment loader for his TIGS job, the name of the user-supplied error routine must be ERROR.

The following program fragment illustrates a user error routine which tolerates TIGS error number 901 but stops the program on all other errors. Error 901 indicates that the default segment was extended because no segment was open when primitives were encountered. Each time a TIGS routine encounters an error, control is passed to the error routine. If the error number is 901, control is passed right back to the main program; otherwise, the program stops.

```
   .
   .
   .
EXTERNAL ERROR
   .
   .
   .
CALL SMERR (ERROR)
   .
   .
   .
SUBROUTINE ERROR
IF [IERROR (IERR) .NE.901] STOP 11
RETURN
END
```

Note that the name of the error routine must be declared in a FORTRAN EXTERNAL statement.

In the example above, IERROR was called as a function in subroutine ERROR. It could have been called as a subroutine, as follows:

```
   .
   .
   .
SUBROUTINE ERROR
CALL IERROR (IERR)
IF (IERR.NE.901) STOP 11
RETURN
END
```

It is also possible to use the user-defined error routine for only part of the time, and then return to default error processing.

```
   .
   .
   .
EXTERNAL ERROR, NULL
   .
   .
   .
CALL SMERR (ERROR)
   .
   .
   .
CALL SMERR (NULL)
   .
   .
   .
```

The programmer should exercise caution when developing user-supplied error routine(s). With the exception of IERROR, TIGS routines should not be called by the user's routine because recursive subroutine calls may result. Also, if the programmer is using either the overlay loader or the segment loader, any user-supplied error routine(s) must be included in the zero-level overlay or the root segment.

CHARACTER SET

TIGS FORTRAN programs, running under either NOS/BE or NOS operating systems, use the character set shown in table A-1. The display code values shown in the table are translated to Hollerith values for all FORTRAN usages, including input to the TEXT and PROMPT routines.

For applications requiring the extended ASCII 128 character set for READ and WRITE input/output, refer to the appropriate operating system reference manual. The extended character set cannot be used with the TIGS TEXT and PROMPT routines.

TABLE A-1. TIGS CHARACTER SET

CHAR.	CODE (7-BIT OCTAL)	CODE (7-BIT HEXADECIMAL)	INTERNAL DISPLAY CODE (6/12-BIT OCTAL)	CHAR.	CODE (7-BIT OCTAL)	CODE (7-BIT HEXADECIMAL)	INTERNAL DISPLAY CODE (6/12-BIT OCTAL)
:	072	3A	00†	5	065	35	40
A	101	41	01	6	066	36	41
B	102	42	02	7	067	37	42
C	103	43	03	8	070	38	43
D	104	44	04	9	071	39	44
E	105	45	05	+	053	2B	45
F	106	46	06	–	055	2D	46
G	107	47	07	*	052	2A	47
H	110	48	10	/	057	2F	50
I	111	49	11	(	050	28	51
J	112	4A	12	)	051	29	52
K	113	4B	13	$	044	24	53
L	114	4C	14	=	075	3D	54
M	115	4D	15	(SPACE)	040	20	55
N	116	4E	16	,	054	2C	56
O	117	4F	17	.	056	2E	57
P	120	50	20	#	043	23	60
Q	121	51	21	[	133	5B	61
R	122	52	22	]	135	5D	62
S	123	53	23	%	045	25	63†
T	124	54	24	"	042	22	64
U	125	55	25	_	137††	5F	65
V	126	56	26	!	041	21	66
W	127	57	27	&	046	26	67
X	130	58	30	'	047	27	70
Y	131	59	31	?	077	3F	71
Z	132	5A	32	<	074	3C	72
0	060	30	33	>	076	3E	73
1	061	31	34	@	100	40	74
2	062	32	35	\	134	5C	75
3	063	33	36	^	136	5E	76
4	064	34	37	;	073	3B	77

† IN THE 63-CHARACTER SET, THIS DISPLAY CODE REPRESENTS A COLON (:), 7-BIT ASCII CODE 072, 7-BIT HEXADECIMAL CODE 3A.

†† ON TTY MODELS HAVING NO UNDERLINE, THE BACKARROW (←) TAKE ITS PLACE.

GLOSSARY

Action	Response to an event. A function key or segment can be assigned to one of three action categories. • IGNORE – No action is taken by TIGS when a segment in this action category is picked or when a function key in this action category is pressed. • RECOGNIZE – Information about the event is reported to the application program; if the event is a segment pick, it is echoed at the terminal. • TERMINATE – This category encompasses all the functions of the RECOGNIZE action. Additionally, TIGS treats the event associated with this action category as the last event in a sequence of events to be reported to the application program.
Attribute	A characteristic of a segment. When the first primitive of a segment is defined, the current modal settings become the attributes of the segment.
Drawing Beam	A logical pointer. Current beam position is defined as the position, in absolute user coordinates, of the beam in the segment being defined. It should not be confused with the beam that draws on the terminal display screen: the current positions of the logical and physical drawing beams are not necessarily the same.
Event	A segment pick or function key press operation performed by the terminal operator.
Event Queue	Function key presses and segment picks are queued up until an event with a TERMINATE action is detected. The application program can then take events off the queue one at a time.
Font	Character style to be used for text generation (e.g., italics).
Function Keys	Keys which can be selected by the operator to inform the application about the next operation to be performed. Terminals without physical function keys use numeric keys 0-9 to simulate function keys.
Highlighting	A method of emphasizing certain geometry or text strings. The implementation of highlighting is terminal dependent and could be done by blinking or by using a reserved intensity, line style, color, and so forth.
Intrasegment Identifier	An identifier associated with a portion of a segment which is returned upon picking that portion of the segment.
Left-Hand Relative Transformation	A transformation in which the current transformation matrix is left-multiplied by the input transformation matrix, that is, the input matrix is on the left.
Locate	To choose a point on the screen and have its coordinates returned to the calling program.
Locator Device	The hardware device used to perform segment picks or choose locations.
Location Queue	Locations are queued up until the appropriate terminal-dependent TERMINATE action is detected. The application program can then process locations from the location queue.
Mode	The current state of a TIGS system parameter; for example, line style, character size, plotting symbol. Defaults exist for all modes.
Modeling	The process of using primitives, segments, and pictures to define objects in space.
Neutral Display File (NDF)	File containing the data required to generate the display. NDF data is generated by the TIGS preprocessor and interpreted by the TIGS postprocessor.
Picture	A collection of segments.
Plotting Symbols	A set of centered symbols which serve as markers on user-generated plots.
Postprocessor	The terminal-dependent, or device driver, portion of TIGS.

Preprocessor	The terminal-independent portion of TIGS, that is, that part of TIGS which remains the same regardless of the type of terminal being used.
Primitives	The products of the line drawing, beam positioning, dot generation, arc generation, plot generation, and text generation functions of TIGS.
Right-Hand Relative Transformation	A transformation in which the current transformation matrix is right-multiplied by the input transformation matrix; that is, the input matrix is on the right.
Screen Coordinates	Virtual coordinates defining the area on the screen where the graphical entities are to be viewed. Range of default screen coordinates is (0.,0.) to (1.,1.).
Segment	A collection of primitives which has attributes associated with it. One or more segments belong to a picture. A segment belongs to only one picture at a time.
System Viewport	A portion of the physical screen where operator prompts are to appear.
User Coordinates	A coordinate system defined by the programmer with the SMPLIx routine or by default. Limits on user coordinates are the largest and smallest numbers representable on the processor being used.
Viewing	The process of developing instructions, using window and viewport routines, on how the basic graphics model is to be viewed on the terminal screen.
Viewport	That part of the terminal screen in which the associated window is to appear. One or more windows from one or more pictures may appear in a viewport.
Window	That part of a picture to be viewed. One or more windows may be defined on a picture. A window is assigned to a viewport.

ERROR MESSAGES

It is a part of the TIGS design philosophy to attempt to continue program execution despite TIGS programming errors. Thus there are no fatal TIGS programming errors (although TIGS errors may eventually result in conditions which will cause job step abort). The error messages in this appendix are warning messages and serious error diagnostics. The consequences of a programming error which results in a warning message are generally trivial; either TIGS can compensate for the error, or ignoring the statement containing the error will not hamper program execution unduly. Serious error diagnostics indicate conditions in which TIGS can only ignore the statement containing the error and attempt to resume program execution at the next statement.

All error messages issued by TIGS are logged on an error file. The means of access to this file is operating system dependent (refer to appendix E).

WARNING MESSAGES

ILLEGAL ID ERRORS (1-100)

Error Number	Cause	Action Taken by TIGS	Routines Affected
1	Attempt to use an **idseg** that is undefined or open when it should not be.	Call is ignored.	RAxxxx,TAxxxx,COPY RENAME,EMPTY,DELSEG
2	Attempt to change the picture attribute of a segment to a nonexistent **idpict**.	Call is ignored.	RAPICT
3	Attempt to delete or put blinds on a nonexistent picture.	Call is ignored.	DELPIC,BLINDS
4	ID of window is not defined.	Call is ignored.	SCRNUR,LOCATE,EVENT, DELWIN
5	ID of viewport is not defined.	Call is ignored.	LOCATE,EVENT,DELVUP, SMPORT
6	Attempt to open or extend a segment when one is already open.	The open segment is closed.	OPNSEG,EXTSEG
10	Attempt to extend a picture that does not exist.	Picture is opened.	SMPICT(≡EXTPIC)
11	Attempt to extend a segment that does not exist.	Segment is opened.	EXTSEG

CHARACTER OR WORD COUNT ERRORS (101-200)

Error Number	Cause	Action Taken by TIGS	Routines Affected
102	**npoint**≤0 on a plot call.	Call is ignored.	PLOTA,PLOTR,PLOTA3, PLOTR3
103	**maxchr**≤0 on a request for keyboard input.	Call is ignored.	KEYBRD
104	**nwords**<0 or **nwords**>26 on "User Data" specification.	Call is ignored.	UDATA
105	Number of characters in prompting message or text string is out of range (**nchars**<0 or **nchars**>256).	If **nchars**<0, call is ignored. If **nchars**>256, the first 256 characters are processed.	TEXT,PROMPT,TEXT3

CLIPPED AND UNCLIPPED BEAM POSITION ERRORS (201-300)

Error Number	Cause	Action Taken by TIGS	Routines Affected
201	Clipped (to picture limits) and unclipped beam positions unequal when picture was closed and now programmer wishes to extend picture.	Picture will be extended and unclipped position set to the clipped position at the previous close of the picture.	SMPICT(≡ EXTPIC)
202	Clipped and unclipped beam positions unequal when segment was closed and now programmer wishes to extend segment.	Segment will be extended and unclipped position set to clipped position at the previous close of the segment.	EXTSEG

UNSUPPORTED POSTPROCESSOR FEATURES (301-400)

Error Number	Cause	Action Taken by TIGS	Routines Affected
301	Request for unsupported font.	Font number NFONT is used where NFONT is the number of fonts supported.	SMFONT,RAFONT
302	Request for unsupported intensity.	Default intensity is used.	SMINT,RAINT
303	Request for unsupported plotting symbol.	Plotting symbol number NSYM is used where NSYM is the number of plotting symbols supported.	SMSYM,RASYM
304	Request for highlighting when highlighting is not supported.	Call is ignored.	SMHILT,RAHILT
305	Request for continuous characters when not supported.	Call is ignored.	SMCSIZ

RESERVED WARNING NUMBERS (401-900)

MISCELLANEOUS ERRORS (901-1000)

Error Number	Cause	Action Taken by TIGS	Routines Affected
901	Primitive encountered and no open segment.	Segment 0 is extended.	ARCA,ARCR,ARCDA,ARCDR, DOTA,DOTR,DRAWA,DRAWR, MOVEA,MOVER,PLOTA, PLOTR,TEXT,ARCA3,ARCR3, ARCDA3,ARCDR3,DOTA3, DOTR3,DRAWA3,DRAWR3, MOVEA3,MOVER3,PLOTA3, PLOTR3
902	Mode set locator is 0 and user wishes to specify locations.	Call is ignored.	LOCATE
903	Attempt to pop an empty 2-D transformation stack.	CTM (Current Transformation Matrix) is set to the identity matrix.	POP
904	Attempt to pop an empty 3-D transformation stack.	CTM3 (Current 3-D Transformation Matrix) is set to the identity matrix.	POP3

SERIOUS ERROR DIAGNOSTICS

PRIMITIVE SPECIFICATION ERRORS (1001-1100)

Error Number	Cause	Action Taken by TIGS	Routines Affected
1001	Arc center outside picture limits.	Call is ignored.	ARCA,ARCR,ARCDA,ARCDR, ARCA3,ARCR3,ARCDA3, ARCDR3
1002	Radii not approximately equal.	Call is ignored.	ARCA,ARCR,ARCDA,ARCDR, ARCA3,ARCR3,ARCDA3, ARCDR3
1003	Radius ≤ 0.	Call is ignored.	ARCDR,ARCDA,ARCR,ARCA, ARCDR3,ARCDA3,ARCR3, ARCA3
1004	Relative degrees of arc approximately 0.	Call is ignored.	ARCDA,ARCDR,ARCDR3, ARCDA3
1005	Sum of squares of direction cosines $\neq 1$.	Call is ignored.	ARCA3,ARCR3,ARCDA3, ARCDR3
1006	Direction cosine vector is coincident with or parallel to the vector from the start point to the center point.	Call is ignored.	ARCA3,ARCR3,ARCDA3, ARCDR3

SEGMENT, PICTURE, WINDOW, OR VIEWPORT ERRORS

Error Number	Cause	Action Taken by TIGS	Routines Affected
1101	Attempt to extend a locked segment.	Return without extending the segment. (If there was an open segment, it is closed.)	EXTSEG
1102	**idwind** out of range (**idwind**< 0 or **idwind**>32767).	Call is ignored.	WINDOW,SCRNUR,LOCATE, EVENT,DELWIN,WINxxx (3-D)
1103	**idport** out of range (**idport**≤ 0 or **idport**>32767).	Call is ignored.	VUPORT,SMPORT,LOCATE, EVENT,DELVUP,VUPOR3
1104	**idpict** out of range (**idpict**≤ 0 or **idpict**>32767). For BLINDS legal range is 0≤**idpict**≤32767.	Call is ignored.	SMPICT(≡ EXTPIC),OPNPIC, DELPIC,BLINDS
1105	**idseg** out of range (**idseg**≤ 0 or **idseg**>32767).	Call is ignored.	RENAME,RAxxxx,TAxxxx, COPY,EMPTY,DELSEG, EXTSEG,OPNSEG
1106	Viewport or system viewport limits are invalid.	Call is ignored.	VUPORT,SMSVP,VUPOR3
1107	Window limits outside range of picture limits or lower left corner not less than upper right corner.	Call is ignored.	WINDOW
1108	Attempt to define a window on the default viewport.	Call is ignored.	WINDOW,WINxxx (3-D)
1109	Attempt to delete the modally set picture.	Call is ignored.	DELPIC
1110	New segment ID is already defined.	Call is ignored.	RENAME,COPY
1113	**idseg** already defined.	Call is ignored.	OPNSEG
1114	**idpict** already defined.	Picture 0 is extended.	OPNPIC
1115	Eye position is the same as the center of attention.	Call is ignored.	WINDIR
1116	**width** or **height** of window< 0.	Call is ignored.	WINSIZ
1117	Zero-length up direction vector specified.	Call is ignored.	WINUP
1118	Far clipping plane is closer to eye than near clipping plane or clipping planes are coincident.	Call is ignored.	WINCLP
1119	Up direction is parallel to the line of sight.	Call is ignored.	WINUP,WINDIR
1120	Near or far clipping plane is not in front of the eye.	Call is ignored.	WINCLP,WINDIR
1121	Projection plane is not in front of the eye.	Call is ignored.	WINPLN,WINDIR

TWO-/THREE-DIMENSIONAL INTERMIXING ERRORS (1201-1300)

Error Number	Cause	Action Taken by TIGS	Routines Affected
1201	Attempt to put a 2-D window on a 3-D picture.	Call is ignored.	WINDOW
1202	Attempt to put a 2-D primitive or 2-D transformation in a 3-D segment.	Call is ignored.	RAXFA, RAXFL, RAXFR, ARCA, ARCR, ARCDA, ARCDR, DOTA, DOTR, DRAWA, DRAWR, MOVEA, MOVER, PLOTA, PLOTR, SMXFA, SMXFR, SMXFL, TEXT, SMROT, RAROT
1203	2-D locations requested when location queue is 3-D.	Call is ignored.	LOCATE
1205	Window or viewport is 3-D and both should be 2-D.	Call is ignored.	SCRNUR
1206	Attempt to put a 3-D primitive or 3-D transformation in a 2-D segment.	Call is ignored.	RAXFA3, RAXFL3, RAXFR3, ARCA3, ARCR3, ARCDA3, ARCDR3, DOTA3, DOTR3, DRAWA3, DRAWR3, MOVEA3, MOVER3, PLOTA3, PLOTR3, SMXFA3, SMXFR3, SMXFL3, TEXT3, SMROT3, RAROT3
1207	Attempt to put a 3-D segment in a 2-D picture or attempting to put a 2-D segment in a 3-D picture.	Call is ignored.	RAPICT
1208	Attempt to put a 3-D window on a 2-D picture.	Call is ignored.	WINxxx(3-D)
1209	Attempt to put a 3-D window in a 2-D viewport.	Call is ignored.	WINxxx(3-D)
1210	Attempt to modify a 2-D window with 3-D information.	Call is ignored.	WINxxx(3-D)
1211	Attempt to "LOCATE" (2-D) in a 3-D window; conversion to user coordinates is not possible.	Call is ignored.	LOCATE
1212	Segment picked was in a 3-D window and EVENT (2-D) was called.	Coordinates are returned as zero.	EVENT
1213	Attempt to modify a 3-D window with 2-D information.	Call is ignored.	WINDOW
1214	Attempt to modify a 2-D viewport with 3-D information.	Call is ignored.	VUPOR3
1215	Attempt to modify a 3-D viewport with 2-D information.	Call is ignored.	VUPORT
1216	3-D routine was called where a 2-D call should have been made.	Information will be returned, but z-coordinate will be meaningless.	TMROT3, TAROT3 TMPLI3, TAPLI3, WHERE3, TMXFA3, TMXFL3, TMXFR3, TAXFA3, TAXFL3, TAXFR3
1217	2-D routine called where a 3-D call should have been made.	Information will be returned for x and y only, or for transformation routines, 2-D transformation matrix is returned.	TMPLIM, TAPLIM, WHERE, TMXFA, TMXFL, TMXFR, TAXFA, TAXFL, TAXFR TMROT, TAROT

SET MODE OR RESET ATTRIBUTE ERRORS (1301-1400)

Error Number	Cause	Action Taken by TIGS	Routines Affected
1301	Intensity out of range (**finten**<0. or **finten**>1.)	Call is ignored.	SMINT,RAINT
1302	Font value out of range (**ifont**<0 or **ifont**>63).	Call is ignored.	SMFONT,RAFONT
1303	Action type out of range (**iactn**≤0 or **iactn**>3).	Call is ignored.	RAAC,SMAC,KYAC
1304	Function key number out of range (**idky**<0 or **idky**>255).	Call is ignored.	KYAC
1305	Plot symbol number is out of range (**isym**<0 or **isym**> 32767).	Call is ignored.	SMSYM,RASYM
1306	Line thickness percentage is out of range (**percnt**<0. or **percnt**>100.)	Call is ignored.	SMTZOM,RATZOM
1307	Intrasegment ID is out of range (**idintr**<0 or **idintr**>32767).	Call is ignored.	SMID
1308	Number of words of application-related information is out of range (**ninfo**<0 or **ninfo**>4).	Call is ignored.	SMINFO,RAINFO
1309	Locator code is out of range (**ilocr**<0 or **ilocr**>63).	Call is ignored.	SMLOCR
1310	Line style value is out of range (**istyle**≤0 or **istyle**>4095).	Call is ignored.	RASTYL,SMSTYL
1320	Attempt to change action attribute of a segment after a primitive has been defined.	Call is ignored.	SMAC
1321	Attempt to set picture limits after a primitive or closed segment has been defined.	Call is ignored.	SMPLIM,SMPLI3
1322	Attempt to modify the visibility attribute of a segment after the first primitive has been defined.	Call is ignored.	SMVIS
1323	Attempt to change highlighting attribute of a segment after a primitive has been defined.	Call is ignored.	SMHILT
1324	Base and plane vectors are colinear.	Call is ignored.	SMROT3,RAROT3
1325	Character height or width≤0.	Call is ignored.	SMCSIZ,RACSIZ

SYMBOL TABLE (IDLIST) OVERFLOW (1401-1500)

Error Number	Cause	Action Taken by TIGS	Routines Affected
1401	Symbol Table overflow on opening a picture.	Picture 0 is extended.	OPNPIC
1402	Symbol Table overflow on opening a segment.	No segment is opened and subsequent primitive calls are not processed.	OPNSEG,COPY,RENAME

Error Number	Cause	Action Taken by TIGS	Routines Affected
1403	Symbol Table overflow on defining a viewport.	Call is ignored.	VUPORT,VUPOR3
1404	Symbol Table overflow on defining a window.	Call is ignored.	WINDOW,WINxxx(3-D)

RESERVED SERIOUS ERROR NUMBERS (1501-3900)

MISCELLANEOUS ERRORS (3901-4000)

Error Number	Cause	Action Taken by TIGS	Routines Affected
3901	**xllh≥xury or yllh≥yury or zllh≥zury**	Call is ignored.	RTANGL,RTANG3,SMPLIM, SMPLI3
3902	Requested locator does not exist.	Use highest number locator that does exist.	EVENT,LOCATE
3903	Attempt to hardcopy on a non-existent hard copier.	Call is ignored.	REMSCR
3904	Attempt to rotate about an undefined axis.	Call is ignored.	XROTA3,XROTR3,XROTL3
3905	Attempt to invert a 2-D singular matrix.	Call is ignored.	XINVR
3906	Attempt to invert a 3-D singular matrix.	Call is ignored.	XINVR3
3907	Attempt to reset continuous character size attribute of a segment with a discrete character size attribute.	Call is ignored.	RACSIZ
3908	Attempt to reset discrete character size attribute of a segment with a continuous character size attribute.	Call is ignored.	RADSIZ
3909	Mode set or attribute character size is discrete and a continuous character size test routine was called.	0.,0. is returned.	TMCSIZ,TACSIZ
3910	Mode set or attribute character size is continuous and a discrete character size test routine was called.	0.,0. is returned.	TMDSIZ,TADSIZ

DATA MANAGER DETECTED ERRORS (4001-5000)

Error Number	Cause	Action Taken by TIGS	Routines Affected
4001	No data file or in-core block space available.	No allocation is made.	All 2-D and 3-D primitives, SMxxxx,CLSSEG,CLSPIC, INITIG,OPNPIC,OPNSEG, VUPORT,VUPOR3,WINDOW, WINxxx(3-D)

Error Number	Cause	Action Taken by TIGS	Routines Affected
4002	The version number of the old file does not match the current TIGS version number.	A new file is opened with the old-file name.	INITIG

NOTE

The following error numbers, 4100-4109, will occur if an application is run when a neutral display file from a previous run has not been returned. QUITIG called with LDELET=.TRUE. will return the NDF. Refer to Data Handler Reference Manual for details.

Error Number	Cause	Action Taken by TIGS	Routines Affected
4100	Illegal bead address passed to internal routine DMDMP.	Next address passed to DMDMP is processed.	None.
4101	The file to be dumped by internal routine DMDMP has not been initialized.	Next address passed to DMDMP is processed.	None.
4102	File to be accessed has not been initialized.	No allocation is made.	SMxxxx,COPY,INITIG, OPNPIC,LCKSEG,CLSSEG, OPNSEG,CLSPIC,SMPICT, QUITIG,WINDOW,WINxxx(3-D), PUSH,PUSH3
4103	Block count limit exceeded.	No bead is allocated.	Same as for 4102.
4104	File is not in the correct format to be processed by the TIGS Data Manager.	The error is reported on the error file and on the terminal screen. Program execution terminates.	INITIG
4105	Internal Data Handler array, IBLK, is too short.	The error is reported on the error file and on the terminal screen. Program execution terminates.	INITIG
4106	Internal common block/ZLDMTB/ loaded after internal array IBLK.	The error is reported on the error file and on the terminal screen. Program execution terminates.	INITIG
4107	Illegal bead address passed to internal routine DMRLBD.	Next address passed to DMRLBD is processed.	CLRSTK,QUITIG,CLRST3, DELPIC,DELSEG,DELVUP, DELWIN,EMPTY,POP, POP3
4108	Illegal bead address passed to internal routines DMSET or DMGET.	Call is ignored.	SMxxxx,COPY,CLSPIC, QUITIG,CLSSEG,OPNSEG, CLRSTK,CLRST3,DELPIC, DELSEG,DELVUP,DELWIN, EMPTY,INITIG,OPNPIC, EXTSEG,LCKSEG,BLINDS, DSPLAY,EVENT,KEYBRD, LOCATE,RAxxxx,TAxxxx, SCRNUR,WINxxx(3-D), VUPORT,WINDOW,VUPOR3, POP,POP3,PUSH,PUSH3, RENAME
4109	Illegal component type code passed to internal routines DMSET or DMGET.	Call is ignored.	Same as for 4108.

TIGS RESERVED ERROR NUMBERS (5001-10000)

TRANSFORMATION MATRICES IN TIGS

MATRIX MULTIPLICATION

Recall that if $A=(a_{ij})$ and $B=(b_{ij})$ are matrices of the appropriate dimension, the product $AB=C=(c_{ij})$ is defined by:

$$c_{ij}=\sum_{k=1}^{n} a_{ik}b_{kj}$$

where n is the number of columns of A and the number of rows of B. If A is an n x n matrix and B is an n x 1 matrix, the definition above includes the notion of a matrix operating on a vector.

If A and B are both n x n matrices, AB and BA are both defined, but in general, AB does not equal BA.

Examples:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 12 & 1 & -2 \\ 3 & 4 & 6 \\ 7 & 11 & 15 \end{bmatrix}$$

AB = B; A is the identity matrix

$$A = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$AB = \begin{bmatrix} -Y \\ X \\ 1 \end{bmatrix}$$ BA is undefined

$$A = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & 3 \\ 0 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$AB = \begin{bmatrix} 1 & 0 & 1 \\ 0 & -1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \qquad BA = \begin{bmatrix} -1 & 0 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

TRANSFORMATION OPERATIONS IN TIGS

TIGS treats all transformation operations (translation, rotation, shearing, and so forth) as matrices operating on vectors. To implement this, TIGS augments vectors by adding an additional coordinate which is always 1, and augments matrices by adding an additional row which is all zeroes except in the last column which is 1. The augmentation is done internally by TIGS.

Matrices are specified by the programmer. The vectors are implied by declaring the segment to be operated upon by the transformation matrix.

The programmer specifies the transformation matrix either by building a matrix with TIGS matrix utility calls or by supplying the matrix directly to the TIGS transformation routines SMXFxx and RAXFxx for effects TIGS matrix utility routines do not produce, such as shearing. In either case, the programmer must reserve storage space for the matrix by dimensioning an array in which to place it. These arrays are 2 x 3 arrays for two-dimensional transformations and 3 x 4 arrays for three-dimensional transformations. As mentioned earlier, matrix augmentation is done internally by TIGS; the array is not dimensioned to include the final row even when the programmer supplies his own matrix.

Given below are the matrices built by TIGS matrix utility routines. The last row of each matrix is enclosed in parentheses to indicate that this row is supplied by TIGS.

$$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ (0 & 0 & 1) \end{bmatrix}$$

2-D translation matrix, where T_x and T_y are the displacements in the x and y directions.

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ (0 & 0 & 1) \end{bmatrix}$$

2-D rotation matrix, where α is the angle through which to rotate the segment.

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ (0 & 0 & 1) \end{bmatrix}$$

2-D scaling matrix, where S_x and S_y are the x and y axis scale factors.

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ (0 & 0 & 0 & 1) \end{bmatrix}$$

3-D translation matrix, where T_x, T_y, and T_z are the displacements in the x, y, and z directions.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ (0 & 0 & 0 & 1) \end{bmatrix}$$

3-D rotation matrix for rotation about the x axis where α is the angle through which to rotate the segment; refer to section 5 for direction-of-rotation conventions.

$$\begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ (0 & 0 & 0 & 1) \end{bmatrix}$$

3-D rotation matrix for rotation about the y axis where α is the angle through which to rotate the segment.

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ (0 & 0 & 0 & 1) \end{bmatrix}$$

3-D rotation matrix for rotation about the z axis where α is the angle through which to rotate the segment.

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ (0 & 0 & 0 & 1) \end{bmatrix}$$

3-D scaling matrix where S_x, S_y, and S_z are the x, y, and z scale factors.

For two-dimensional transformations, the vector used is:

$$\begin{bmatrix} X \\ Y \\ (1) \end{bmatrix}$$

For three-dimensional, it is:

$$\begin{bmatrix} X \\ Y \\ Z \\ (1) \end{bmatrix}$$

Initially, the transformation matrix for a given segment is the identity matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ (0 & 0 & 1) \end{bmatrix}$$

For 2-D segments

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ (0 & 0 & 0 & 1) \end{bmatrix}$$

For 3-D segments

TIGS matrix utility routines build matrices which are applied to a segment by using the SMXFxx or RAXFxx transformation routines. Both utility and transformation routines can be specified as absolute or relative routines (refer to section 5). Remember that the relative routines imply matrix concatenation and are available in right or left multiplication versions. The right multiplication routines perform matrix multiplication with the supplied matrix on the right of the existing matrix; the left multiplication routines perform multiplication with the supplied matrix on the left. The following example will make this clearer.

Suppose a two-dimensional segment has an existing transformation attribute represented by the matrix

$$\begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ (0 & 0 & 1) \end{bmatrix}$$

which moves the segment 3 units in the negative x direction and 3 units in the negative y direction. Now the programmer wishes to apply an additional transformation to the segment by rotating it 90° counterclockwise. A call to XROTA produces the following matrix:

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ (0 & 0 & 1) \end{bmatrix}$$

The programmer now has a choice of using right or left multiplication routines in resetting the transformation attribute of that segment. RAXFL produces the following multiplication:

$$\underset{\text{new matrix}}{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ (0 & 0 & 1) \end{bmatrix}} \text{ X } \underset{\text{existing matrix}}{\begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ (0 & 0 & 1) \end{bmatrix}} = \underset{\text{resultant matrix}}{\begin{bmatrix} 0 & -1 & 3 \\ 1 & 0 & -3 \\ (0 & 0 & 1) \end{bmatrix}}$$

while RAXFR produces:

$$\underset{\text{existing matrix}}{\begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ (0 & 0 & 1) \end{bmatrix}} \text{ X } \underset{\text{new matrix}}{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ (0 & 0 & 1) \end{bmatrix}} = \underset{\text{resultant matrix}}{\begin{bmatrix} 0 & -1 & -3 \\ 1 & 0 & -3 \\ (0 & 0 & 1) \end{bmatrix}}$$

The considerations governing the choice of right or left multiplication routines are detailed in section 5.

In either case, when the matrix has been built, it is applied to the segment by multiplying vectors with the vector to the right of the matrix.

$$\begin{bmatrix} 0 & -1 & \pm 3 \\ 1 & 0 & -3 \\ (0 & 0 & 1) \end{bmatrix} \text{ X } \begin{bmatrix} X \\ Y \\ (1) \end{bmatrix} = \begin{bmatrix} -Y\pm 3 \\ X-3 \\ (1) \end{bmatrix}$$

Refer to section 5 for further information.

OPERATING SYSTEM DEPENDENCIES AND LOADING PROCEDURES E

TIGS supports applications running on both NOS/BE and NOS operating systems. There are aspects of TIGS usage that vary depending on which operating system is being used. Loading procedures, however, are the same on both systems. This appendix describes both operating system dependencies and loading procedures and is divided into the following subsections: field Length considerations, TIGS under NOS/BE, and TIGS under NOS.

FIELD LENGTH CONSIDERATIONS

The field length of a program using TIGS depends upon the program's:

- Complexity
- Data structures
- Operating system requirements
- Utilization of TIGS features

Simple programs using TIGS require approximately 50K words, hence advanced loading procedures such as overlaying or segmenting become mandatory for sophisticated programs. If a program is designed with overlaying or segmenting in mind, field length can be dramatically reduced without adding signficant overhead due to overlay or segment swapping. In fact the reduced field length often outweighs the additional overhead thus yielding both better response time and less cost.

OVERLAY LOADING

TIGS routines contain no overlay calls, thus unless a programmer is willing to go into the internals of TIGS, the greatest potential for saving memory results from overlaying the user's routines. This scheme indirectly partitions the TIGS routines into the various overlays realizing up to 10K words of savings above that saved by overlaying the user routines.

CAUTION

When using the overlay loader the following rules must be observed.

- The TIGS library must have been built with UPDATE directive *DEFINE SEGLDR; consult a systems analyst for this information.
- INITIG, NULL and any user-supplied error routines (refer to SMERR) must be located in the zero-level overlay.

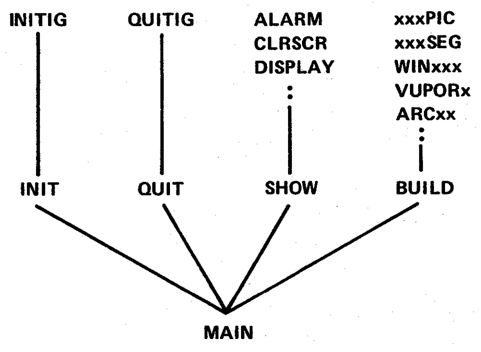
SEGMENT LOADING

The execution of a segmented program does not differ greatly from the execution of an overlayed program. The segment loader does however provide more flexible overlay structure (via trees and levels) and is directive driven so that explicit overlay calls do not have to be inserted within the source code. If the programmer structures his application to correspond to the general structure of TIGS, the segment loader can save considerable field length.

The TIGS application should be partitioned into at least four subroutines: initialization, termination, display, and picture creation/alteration. In addition, any application functions not interfacing with TIGS directly should be partitioned into subroutines. TIGS calls should be distributed among the four primary partitions as follows:

Initialization	INITIG
Termination	QUITIG
Display	ALARM,CLRSCR,DSPLAY, EVENT,KEYBRD, LOCATE,PROMPT, REMSCR,UNISCR
Picture creation/ alteration	All other TIGS routines

A segment tree for the preceding structure would appear as follows (note the similarity).



The corresponding segment directives are:

```
        TREE     MAIN-(INIT,QUIT,SHOW,BUILD)
MAIN    INCLUDE  NULL,INITIG, user-supplied error
                 routines
MAIN    GLOBAL   All labeled common blocks (user,
                 TIGS and system level)
        END
```

CAUTION

As in the case of overlay loading this simple structure realizes the greatest memory savings from segmenting the user's routines rather than TIGS.

When using the segment loader the following rules must be observed.

- The TIGS library must have been built with UPDATE directive *DEFINE SEGLDR, consult a systems analyst for this information.
- INITIG,NULL and any user-supplied error routines (refer to SMERR) must include the zero-level root segment.
- All labeled common blocks (particularly TIGS and system level) must be globalled in the zero-level root segment. Note that common block names vary from postprocessor to postprocessor and from system to system. Failure to global a common block in the zero-level root segment does not produce a loader error message and yields unpredictable results.

Any program that remains field length critical after the application routines have been segmented requires that the internal TIGS routines be segmented. Skeleton segment directive files unique to each postprocessor is available in this segmentation (consult a systems analyst for this information). This file should be copied as follows:

- The user's main program name should replace all references to MAIN in the directive file.
- Any user-supplied error routines must be included in segment USER.
- System level and appropriate user level common blocks must globalled in segment USER.
- Any additional user level segment directives should be inserted.

Depending upon the application and the TIGS functions it utilizes, this reasonably complex segment structure can save from 2K to 20K words of main memory. An experienced programmer with a good working knowledge of the segment loader can further tailor these directives to achieve further memory savings for his particular application, however, caution must be exercised when altering the TIGS portion of these directives because incorrect placement of certain TIGS routines can lead to excessive segment swapping and degraded performance.

TIGS UNDER NOS/BE

This section assumes a working knowledge of NOS/BE and INTERCOM procedures.

INTERCOM is the interactive facility of NOS/BE. TIGS applications in NOS/BE run on graphics terminals under INTERCOM control. A graphics terminal logged in to INTERCOM operates like any other time-sharing terminal when not executing a graphics application program. The full range of INTERCOM commands and directives can be entered from the graphics terminal, along with many NOS/BE control statements.

The TIGS user has several basic tasks to accomplish before running a graphics application program. He must:

- Establish communications with INTERCOM.
- Enter the source code of the application program.†
- Compile the program under the NOS/BE FORTRAN Extended compiler.†
- Select the correct postprocessor library.
- Execute the graphics application program.

INTERCOM has commands to accomplish all of the preceding including variations like interactive entry of code via EDITOR, XEQ options for compilation and execution, and so on. Because all these tasks except the selection of the postprocessor library are general time-sharing subjects and not peculiar to TIGS, this appendix does not describe how to accomplish them. The user should consult the INTERCOM Version 5 Reference Manual for this information.

Once execution of the compiled graphics program has been initiated, the program generates pictures on the terminal display screen and interacts with the terminal operator via TIGS FORTRAN calls. During the time the graphics program is executing, the terminal is not in the INTERCOM command mode; INTERCOM commands cannot be entered. The graphics program can be terminated by the application program through the QUITIG call, or by the terminal operator using the INTERCOM abort procedure of typing %A.††

When the program is terminated, the terminal is returned to INTERCOM command mode, which permits the operator to enter INTERCOM commands, execute another graphics program, or reexecute the same program.

FILES

Files used by the application program are controlled and maintained through standard NOS/BE procedures.

The user may declare file OUTPUT and WRITE to the terminal or to the printer. This file is not automatically connected by TIGS; therefore, if the user wishes to direct use of this file to the terminal, the user must CONNECT it before calling it. If this is not done, OUTPUT automatically is directed to the printer, and actual printing may be initiated by using the INTERCOM BATCH command.

†Can be accomplished as a batch job if preferred; refer to INTERCOM Version 5 Reference Manual.
††When a program is terminated in this manner, the neutral display file must be RETURNed to the system before another program using the same neutral display file name can be executed.

Output from program WRITE statements for a printer listing is made by writing a file which is not connected; such a file may be printed following execution of the job using the INTERCOM BATCH command.

Files other than INPUT and OUTPUT may be connected or disconnected from the terminal by use of the INTERCOM commands CONNECT or DISCONT, or with the FORTRAN Extended routines CONNEC or DISCON.

All files used from the console in either INTERCOM command mode or during a graphics application must be mass storage files or connected files. INTERCOM does not permit the use of magnetic tape from interactive programs. If a user wishes to refer to a magnetic tape, a batch job must be submitted to the system which manipulates a tape file and transfers required data to a permanent file. This permanent file may then be accessed under INTERCOM control.

TIGS uses units 61, 62, 63, 64, and 65 for internal input and output of graphics data to the terminal. These units must be defined on the PROGRAM statement of the application program together with any other units required by the application. These units are used for the following purposes:

TAPE61	TIGS input
TAPE62	TIGS output
TAPE63†	Error file
TAPE64	LOCATE queue
TAPE65	EVENT queue

The programmer should not attempt to read from or write to any of these units in his program.

The files listed on the PROGRAM card are each assigned a default buffer size of 515 words (decimal). The programmer can reduce field length requirements by reducing the buffer sizes of the files. File OUTPUT must not be equated to TAPE62, as this causes input/output conflicts; however, the user may reduce the OUTPUT file buffer size to a minimum of 64 words. File TAPE62 is used exclusively by TIGS, but due to the manner in which it is used, it can (with no degradation) be specified with a buffer size of 64 words. The following code is an example of a PROGRAM card that utilizes buffer space in the most efficient manner.

```
   PROGRAM TEST(TAPE61=64,TAPE62=64,
 c TAPE63=64,TAPE64=64,TAPE65=64)
```

TIGS LIBRARY SELECTION

All TIGS subroutines exist as library subroutines and are loaded as needed when referred to by application programs. There is a separate library for each postprocessor supported by TIGS. It is the operator's responsibility to attach the correct library file to his job before loading and execution. The names of the postprocessor libraries may vary from site to site; consult a systems analyst for the names in use. Also, the means of making the library local to the graphics job may vary. If the TIGS library exists as a permanent file, it must first be attached to the job and declared as a library:

```
ATTACH,libname,ID=ident††

LIBRARY,libname
```

If TIGS libraries have been included in the system deadstart tape, a different procedure will be used. Consult a systems analyst for more information.

NOS/BE INTERCOM CHARACTER SET

FORTRAN programs using TIGS use a 64-graphic character set. Appendix A contains character set tables.

DATA HANDLER RESTRICTION

If a TIGS application program uses the Data Handler utility, it is limited to using Data Handler files 1 through 7; TIGS software uses file 0. Refer to the Data Handler Reference Manual.

NEUTRAL DISPLAY FILE CONVERSION

Any neutral display file created under TIGS v.1.0 will need to undergo a one-time conversion in order to be used by TIGS v.1.1. The conversion program to be used should be installed as part of the TIGS installation procedure. Consult a systems analyst for the conversion program file name. The following job should be run in either interactive or batch mode.

```
ATTACH,NDF,ndfname,ID=xxx,PW=abc.†††
                     local file name must be NDF.

ATTACH,conprog,ID=xxx.
                     conversion program.

conprog.             execute conversion program.

EXTEND,NDF.          extend neutral display file.
```

Note that this procedure permanently alters the v.1.0 neutrual display file.

† If the user wishes to see error messages as they are logged onto the error file, the user should CALL CONNEC (63) after INITIG has been called.

†† Default library name and identifier may be changed by installation and will vary with the postprocessor used.

††† Be sure file is attached with extend permission.

TIGS UNDER NOS

This section assumes a working knowledge of NOS and NAM/IAF or the Time-Sharing Module.

NOS has two interactive facilities; NAM/IAF for sites employing network processing of jobs, and the Time-Sharing Module for non-network processing. TIGS applications in NOS run on graphics terminals under the control of one of these two facilities, depending upon what is available at the site. In either case, a graphics terminal logged in to the available time-sharing facility operates like any other time-sharing terminal when not currently executing a graphics application program. The full range of time-sharing commands can be entered from the graphics terminal, and when the terminal is in the batch subsystem, many NOS control statements can also be entered. There are minor differences between the interfaces of the two systems. A TIGS user would find the most noticeable differences in the log-in and program interrupt/termination procedures. Consult a systems analyst for information on the system in use at your site.

The TIGS user has several basic tasks to accomplish before running a graphics application program. He must:

- Establish communications with the time-sharing system.
- Enter the source code of the applications program.
- Compile the program under the NOS FORTRAN Extended Compiler.
- Select the correct postprocessor library.
- Execute the graphics applications program.

NOS time-sharing facilities have commands to accomplish all of the preceding, including options like batch processing for program compilation, on-line source code entry via Text Editor, the Execute Subsystem for program execution, and so on. Because all these tasks except the selection of the postprocessor library are general time-sharing tasks and not peculiar to TIGS, this appendix does not describe how to accomplish them. The user should consult the Time-Sharing User's Reference Manual for this information.

Once execution of the compiled graphics program has been initiated, the program generates pictures on the terminal display screen and interacts with the terminal operator via TIGS FORTRAN calls. The graphics program can be terminated by the application program through the QUITIG call, or by the terminal operator using the system abort procedure (different for NAM/IAF and the Time-Sharing Module).†

When the program is terminated, the operator can once again enter commands, execute another graphics program, or reexecute the same program.

FILES

Files used by graphics application programs are controlled and maintained through standard NOS procedures.

The files INPUT and OUTPUT for TIGS programs are automatically directed to and from the terminal. When a WRITE or PRINT to the OUTPUT file is made, the data is displayed in alphanumeric form on the screen. When a READ from the INPUT file is made, the system waits while the data is typed on the keyboard (followed by a press of the RETURN key), and then passes the typed data to the program.

To obtain a printed listing of output from program WRITE statements, the programmer writes to a file other than OUTPUT and then performs a SAVE on the file after the job. The programmer can then produce the printed listing on the host site printer or remote batch terminal by submitting (SUBMIT command) a batch job or remote batch job which copies the saved file to the OUTPUT file.

All files used from the console, whether in NOS command mode or during an application execution, must be mass storage files or system files. NOS normally does not allow the use of magnetic tape from interactive programs. If a user wishes to refer to a magnetic tape, it can be done by submitting a batch job to the system which manipulates a tape file and transfers required data to or from a permanent file. This permanent file may be accessed from the application program.

TIGS uses units 61, 62, 63, 64, and 65 for internal input and output of graphics data to the terminal. TAPE63 is the error log. TAPE64 is the LOCATE queue, and TAPE65 is the EVENT queue. TAPE61 is the TIGS input (this file must be equivalenced to INPUT on the PROGRAM card), and TAPE62 is the TIGS output file.

The files listed on the PROGRAM statement are each assigned a double buffer, which is included in the field length of the application program. The size of the double buffer defaults to 515 words decimal per file. The programmer may wish to reduce these buffer sizes on the PROGRAM statement to reduce the required field length for the application program. For example, buffer sizes of 64 words each are specified in the following statement.

```
      PROGRAM TEST(INPUT=64,OUTPUT=64,
    c TAPE61=INPUT,TAPE62=0,
    c TAPE63=64,TAPE64=64,TAPE65=64)
```

Minimum file size for all units is 64 words.

†When a program is terminated in this manner, the neutral display file must be returned to the system before another program can be executed.

TIGS LIBRARY SELECTION

All TIGS subroutines exist as library subroutines and are loaded as needed when referred to by application programs. There is a separate library for each postprocessor supported by TIGS. It is the operator's responsibility to get the correct library file for his job before loading and execution. The names of the postprocessor libraries may vary from site to site; consult a systems analyst for the names in use. Also, the means of making the library local to the graphics job may vary. If the TIGS library exists as a permanent file it must first be made local to the job and declared as a library:

```
ATTACH,libname

X,LIBRARY,libname
```

The example above assumes that the permanent file is a direct access file in the catalog of user name LIBRARY. The LIBRARY control statement is preceded by an X, to distinguish it from the time-sharing LIBRARY command, which is unrelated. The LDSET statement can be used instead of the LIBRARY statement if the library is to be used only for a single load sequence (local library instead of a global library; refer to the CYBER Loader Reference Manual).

All TIGS program libraries have version numbers to reflect library updates. This version number has special meaning for the INITIG routine (section 8).

PROGRAM EXECUTION

The execution of a TIGS application is initiated from either the batch or execute subsystems; the batch subsystem must be used to execute direct access files.

The TIGS user may find it convenient to use a NOS procedure file to accomplish the library selection, loading, and execution tasks; refer to the Time-Sharing User's Reference Manual for information.

NOS CHARACTER SET

FORTRAN programs use the NOS 64-character TTY set for normal READ and WRITE input/output for a terminal. Options are also available for input and output of additional characters for the full ASCII subset, and for other special modes of input/output including binary and transparent. Appendix A contains character set tables.

DATA HANDLER RESTRICTION

If a TIGS application program uses the Data Handler utility, it is limited to using Data Handler files 1 through 7; TIGS software uses file 0. Refer to the Data Handler Reference Manual.

Any neutral display file created under TIGS v.1.0 will need to undergo a one-time conversion in order to be used by TIGS v.1.1. The conversion program to be used should be installed as part of the TIGS installation procedures. Consult a systems analyst for the conversion program file name. The following job should be run in either interactive or batch mode:

```
ATTACH,NDF=ndfname/M=W.
                local file name must be NDF.

ATTACH,conprog.
                converion program.

conprog.        execute conversion program.
```

Note that this procedure writes over the v.1.0 neutral display file to make the v.1.1 file.

TEKTRONIX 401X POSTPROCESSOR

This appendix describes the postprocessor which supports the following Tektronix 401x series of graphics terminals.†

- Tektronix 4006 terminals.
- Tektronix 4010 series terminals (includes the 4012 and 4013 terminals).
- Tektronix 4014 terminals.
- Tektronix 4014 terminals with Enhanced Graphics Module (EGM).
- Any terminal capable of emulating the 4014.

The appendix is organized in the following manner.

- A subsection on hardware/operating system environments for the postprocessor.
- A subsection on TIGS software features whose implementation is dependent on the postprocessor used, including a list of values returned from TFxxx routines for the Tektronix postprocessor.
- A subsection containing information on each of the terminal classes mentioned (except for terminals in 4014 emulation). At the end of this subsection are short descriptions of optional Tektronix hardware, such as the tablet and hardcopier.

OPERATING ENVIRONMENT

NOS/BE AND INTERCOM SYSTEM USER'S HARDWARE ENVIRONMENT

Users with access to a CDC CYBER 70, CDC CYBER 170, or 6000 series computer running under NOS/BE and INTERCOM can run TIGS with the 401x postprocessor on any of the Tektronix terminals previously listed. The minimum hardware configuration needed to run TIGS is the same as that needed to run NOS/BE and INTERCOM. The terminal must be connected to the host computer by a telephone line through a data set.

Table F-1 shows the baud rates and modes of communication that are supported by TIGS for NOS/BE and INTERCOM.

TABLE F-1. BAUD RATES AND MODES OF COMMUNICATION FOR NOS/BE AND INTERCOM

Terminal	Baud Rate	Mode of Communication
4006	300 Baud	Asynchronous †
4010 series, 4014		
4006	1200 Baud to 9600 Baud	Asynchronous †
4010 series, 4014		
4010 series, 4014	2000 Baud	Synchronous
4010 series, 4014	2400 Baud	Synchronous
4010 series, 4014	4800 Baud	Synchronous

† When running asynchronously, using the SCREEN,132 command produces an improvement in throughput.

NOS NAM/IAF AND NOS TIMESHARING MODULE SYSTEM USER'S HARDWARE ENVIRONMENT

Users with access to a CDC CYBER 70, CDC CYBER 170, or 6000 series computer running under NOS NAM/IAF or NOS with the Time-Sharing Module (refer to appendix E) can run TIGS with the 401x postprocessor on any of the terminals previously listed. The minimum hardware configuration needed to run TIGS is the same as needed to run NOS. The terminal must be connected to the host computer by a telephone line through a data set. Table F-2 shows the baud rates and modes of communication that are supported by TIGS for NOS NAM/IAF and NOS Time-Sharing Module.

TABLE F-2. BAUD RATES AND MODES OF COMMUNICATION FOR NOS NAM/IAF AND NOS TIME-SHARING MODULE

Terminal	Baud Rate	Mode of Communication
4006	300 Baud	Asynchronous
4010 series, 4014		
4006	1200 Baud to 9600 Baud	Asynchronous
4010 series, 4014		
4010 series, 4014	2000 Baud	Synchronous†
4010 series, 4014	2400 Baud	Synchronous†
4010 series, 4014	4800 Baud	Synchronous†

† Synchronous communications supported on NOS NAM/IAF only.

† Tektronix 401x terminals whose model number is followed with a -1 (for example, 4006-1) are included in this discussion. The -1 signifies that the terminal can be fitted with a hardcopy device.

401x POSTPROCESSOR LIBRARY

The 401x postprocessor is selected by declaring the proper library in the graphics job before program loading takes place. This process is described in appendix E. After the 401x postprocessor library has been selected, a display of installation default options appears on the screen. If the default configuation is correct, no further questions will appear. If not, four questions will appear to define the current configuration. An example display follows:

```
INSTALLATION DEFAULT VALUES ARE
7       4014 W/EGM ASYNCHRONOUS
1200    BAUD
0       NO TABLET
0       NO HARD COPY
DO YOU ACCEPT THESE DEFAULTS (Y/N)

ENTER TERMINAL TYPE
1       4006 ASYNCHRONOUS
2       4010 SYNCHRONOUS
3       4010 ASYNCHRONOUS
4       4014 SYNCHRONOUS
5       4014 ASYNCHRONOUS
6       4014 W/EGM SYNCHRONOUS
7       4014 W/EGM ASYNCHRONOUS

ENTER BAUD RATE, 300,1200,2000,2400,4800,9600

IS TABLET GOING TO BE USED
ENTER 0 FOR NONE,1 FOR 4953,2 FOR 4954

IS HARD COPY UNIT AVAILABLE
ENTER 0 FOR NONE,1 FOR TEKTRONIX,2 FOR VERSATEC
```

Figure F-1. Initial Postprocessor Display

The questions appear one by one, each appearing only after the last has been properly answered. An improper answer causes the question to be repeated. It is important to realize that only invalid answers are rejected, not incorrect ones. That is, an operator response indicating that a 300-baud communications line is in use when the line is actually 1200 baud will be accepted by TIGS, even though it is not correct. A breakdown in communication is the result of such a valid, but incorrect, answer.

It is the operator's responsibility to know the characteristics of the communications line he is using, and to set the terminal to the proper baud rate and communications mode.

TIGS SOFTWARE FEATURES

The following TIGS software features are dependent on the postprocessor for specifics of implementation:

- Default initialization.
- Supported plotting symbol set.
- Supported continuous character set.
- Line style algorithm.
- Echoing segment picks.
- Implementing rotation of text.
- Determining the viewport for PROMPT routine use.
- Returning coordinate values on an EVENT call.
- Erasing the screen.
- Determining default screen layout.
- Keyboard input maximum.

TIGS also contains a set of subroutines that test postprocessor features. This subsection contains a table of values returned by these TFxxxx routines for the Tektronix postprocessor.

DEFAULT INITIALIZATION

If an applications program does not make a call to INITIG, the first TIGS routine called in the program results in a call to INITIG with the following default values.

CALL INITIG(.TRUE.,.TRUE.,6LNEWNDF)

SUPPORTED PLOTTING SYMBOL SET

The Tektronix postprocessor supports the plotting symbol set as defined in the xxSYM routine documentation, section 2.

SUPPORTED CONTINUOUS CHARACTER SET

The Tektronix postprocessor supports the continuous character set defined in the xxCSIZ routine documentation, section 2.

LINE STYLE ALGORITHM

The algorithm used in implementing the bit-pattern line style of the SMSTYL and RASTYL (istyle>5) is as follows: for each set bit of the binary representation of the **istyle** value, the drawing beam is turned on for the distance of .0034188 in terminal-independent coordinate system units. The physical distance represented by this quantity varies among the several Tektronix terminals; refer to the appropriate terminal user's guide for specific information. In general, a single set bit results in a dot; several set bits in a row result in a dash that is directly proportional in length to the number of bits set. For example, the value

ISTYLE=7652B

is equivalent to the binary bit pattern

111110101010

In this case, the drawing beam would be turned on for five screen units in a row, then alternating off and on for the next seven units before the pattern is repeated. This results in a line style consisting of a dash followed by three dots.

ECHOING SEGMENT PICKS

In the Tektronix 401x postprocessor, segment picks are echoed at the terminal by redrawing the picked segment. No screen erasure is involved in this redrawing.

ROTATION OF TEXT

Rotation of text is implemented in TIGS for the Tektronix 401x postprocesor although continuous character sizes (software-, not hardware-produced, character sizes) are not supported. The character itself is not rotated; rather the angle of the lower left corner of the character in relation to the next character is positioned at the set angle of rotation. Figure F-2 is an example of character rotation for the 40xx postprocessor. Positive angles of rotation are counterclockwise and negative angles of rotation are clockwise from the positive x-axis. Refer to TEXT in section 2.

DETERMINING VIEWPORT FOR PROMPT ROUTINE USE

The minimum size for a system viewport is 10 characters in width for the smallest character size. TIGS uses the smallest hardware character size for system viewport messages. If a programmer specifies a system viewport of insufficient width, TIGS adjusts the system viewport limits to adhere to the minimum size. Prompting messages are put in the system viewport in multiples of 10 characters per line.

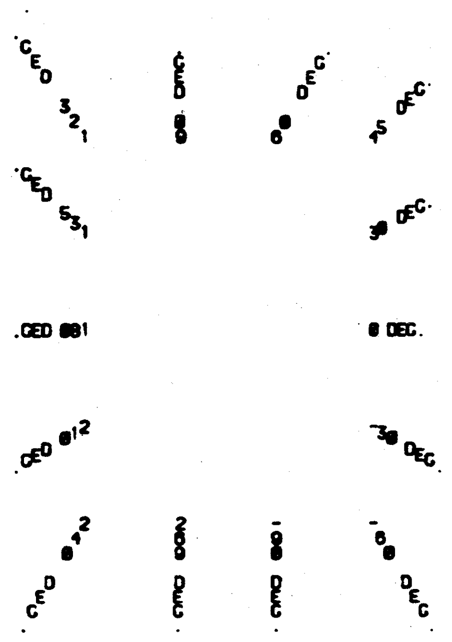


Figure F-2. Example of Character Rotation

For example, suppose the system viewport holds 29 characters and a program calls PROMPT with NCHAR=27. The first 20 characters appear on the next available line of the system viewport and the last 7 characters appear on the following line. Refer to PROMPT in section 7.

The default system viewport width is 20 characters for the 4006 and 4010 series terminals, and 40 characters for the 4014.

RETURNING COORDINATE VALUES ON AN EVENT CALL

When EVENT is called and the operator makes a segment pick, the COORDS array will contain the exact coordinates of the position the operator picked, within hardware limitations. The coordinate system is as defined in the most recent call to PREEVN (refer to section 7).

SCREEN ERASURE

The following are the four conditions that cause the screen to be erased.

- A segment is visible and any of RAPICT, RASTYL, RASYM, RAVIS, RAXFA, RAXFR, RAXFL, RAXFA3, RAXFR3, RAXFL3, EMPTY, or DELSEG involving that segment is called.
- A window, viewport, or picture is deleted.
- PROMPT or KEYBRD is called and there is insufficient room in the system viewport.
- SMSVP is called.

For the first two conditions, the erasure is done when the next display cycle is performed. For the third condition, erasure is performed after a three-second delay. For the last condition, the erasure is performed immediately. A display cycle is performed upon calling DSPLAY, EVENT, LOCATE, or KEYBRD (refer to the sections containing these call statements).

DEFAULT SCREEN LAYOUTS

The default screen layout depends on the value of the logical parameter **lsquar** in the call to INITIG (refer to section 8). Figure F-3 shows default screen layouts for **lsquar** values.

Default viewport covers entire screen; default viewport and default system viewport overlap.

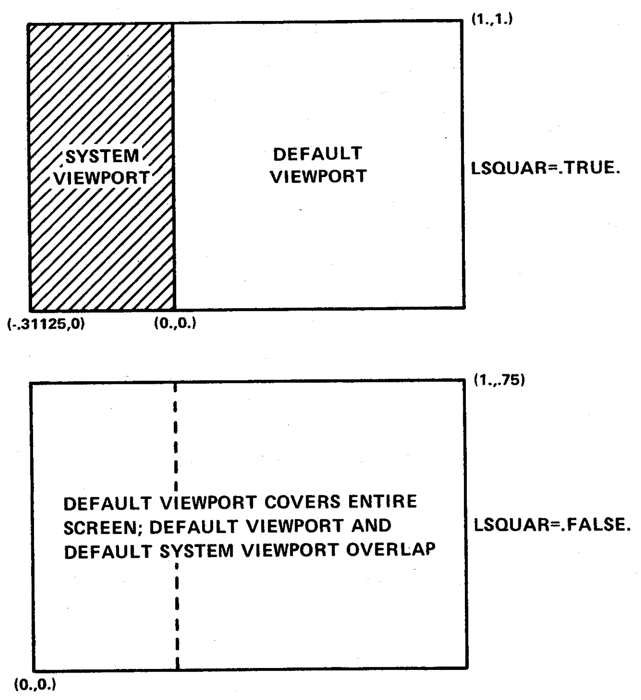


Figure F-3. Default Screen Layouts for **lsquar** Values

NOTE

The PROMPT routine requires that the minimum size for a system viewport is at least 10 characters of the smallest character size fit in the system viewport. When **lsquar** is false the system and default viewports overlap. When **lsquar** is true the system viewport occupies the left side of the default screen layout. The number of characters per line of prompting information for the default system viewport varies depending on the Tektronix terminal. For the Tektronix 4006 and 4010, 20 characters per line are allowed. For the Tektronix 4014, 40 characters per line are allowed.

KEYBOARD INPUT

The Tektronix postprocessor supports an input of 140 characters for each KEYBRD call.

TEST FEATURE ROUTINES

Table F-3 lists the TFxxx subroutines and the values returned for them for the Tektronix postprocessor.

TABLE F-3. TEST FEATURE ROUTINES

Test Feature Routine Name	Feature Tested	Returned Value from Routine
TFAC	Test for postprocessor support of EVENT routine.	LACTN = .TRUE.
TFCSIZ	Test for support of continuous character sizes.	LCCHAR=.TRUE.
TFDSIZ	Test discrete character size.	If LSQUAR = .FALSE.: <u>4006 and 4010 series</u> Characters/line: 74 Lines/frame: 35 WIDOUT = 1./74.† HIOUT = 1./35.† <u>4014</u> Characters/line: 133 Lines/frame: 64 WIDOUT = 1./133. HIOUT = 1./64. Characters/line: 121 Lines/frame: 58 WIDOUT = 1./121. HIOUT = 1./58. Characters/line: 81 Lines/frame: 38 WIDOUT = 1./81.† HIOUT = 1./38.† Characters/line: 74 Lines/frame: 35 WIDOUT = 1./74. HIOUT = 1./35. If LSQUAR = .TRUE.: <u>4006 and 4010</u> Characters/line: 56 Lines/frame: 35 WIDOUT = 1./56.† HIOUT = 1./35.† <u>4014</u> Characters/line: 101 Lines/frame: 64 WIDOUT = 1./101. HIOUT = 1./64. Characters/line: 92 lines/frames: 58 WIDOUT = 1./92.† HIOUT = 1./58.† Character/line: 62 Lines/frame: 38 WIDOUT = 1./62. HIOUT = 1./38. Characters/line: 56 Lines/frame: 35 WIDOUT = 1./56. HIOUT = 1./35.

TABLE F-3. TEST FEATURE ROUTINES (Contd)

Test Feature Routine Name	Feature Tested	Returned Value from Routine
TFERR	Test for support of user-supplied error routine.	LROUTN = .TRUE.
TFFONT	Test number of fonts supported.	NFONTS = 1
TFHARD	Test for existence of remote hardcopiers.	If fourth question (that is, availability of hardcopy unit) during initialization dialogue was answered 0, LREMOT=.FALSE. Otherwise, LREMOT=.TRUE.
TFHILT	Test for support of high-lighting.	LHILT = .FALSE.
TFID	Test for support of intra-segment identifiers.	NID = 32767
TFINT	Test for number of hard-ware levels of intensity supported.	NINTEN = 1
TFLOCR	Test for number of locators on the terminal and their characteristics.	<u>4006</u> NLOCRS = 0 <u>4010 series or 4014</u> If no tablet available: NLOCRS = 1 DESCRP (1) = 1001 (thumbwheels) LONE (1) = .TRUE. If small tablet (4953) available: NLOCRS = 2 DESCRP (1) = 1001 (thumbwheels) DESCRP (2) = 1002 (small) LONE (1) = .TRUE. LONE (2) = .TRUE. If large tablet (4954) available: NLOCRS = 2 DESCRP (1) = 1001 (thumbwheels) DESCRP (2) = 1003 (large tablet) LONE (1) = .TRUE. LONE (2) = .TRUE.

TFNSIZ	Test for number of discrete character sizes supported by hardware.	<u>4006 or 4010 series</u>
		
		NDSIZE = 1
		
		<u>4014</u>
		
		NDSIZE = 4

TFPICT	Test for maximum number of pictures allowed.	NPICT = 509
TFPORT	Test for maximum of view-ports allowed.	NPORT = 508

TFROT	Test for support of char-acter rotation by 90 de-grees and continuous rotation of characters.	LNINTY = .FALSE.
		LCONT = .TRUE.††

TABLE F-3. TEST FEATURE ROUTINES (Contd)

Test Feature Routine Name	Feature Tested	Returned Value from Routine

TFSCRN	Test size, shape, and resolution of the screen.	If LSQUAR = .FALSE.:
		
		LRTNGL = .TRUE.
		XLL = YLL = 0.
		XUR = 1.
		YUR = .75
		RESLTN = 1024. (without EGM)
		RESLTN = 4096. (with EGM)
		
		If LSQUAR = .TRUE.:
		
		LRTNGL = .TRUE.
		XLL = -.31125
		YLL = 0.
		XUR = YUR = 1.
		RESLTN = 780. (without EGM)
		RESLTN = 3120. (with EGM)

TFSTYL	Test for line styles supported by hardware.	<u>4006, 4010 series, and 4014 without EGM</u> LHARD (1) = .TRUE. LHARD (2) = .FALSE. LHARD (3) = .FALSE. LHARD (4) = .FALSE. LHARD (5) = .FALSE. LHARD (6) = .FALSE. <u>4014 with EGM</u> LHARD (1) = .TRUE. LHARD (2) = .TRUE. LHARD (3) = .TRUE. LHARD (4) = .TRUE. LHARD (5) = .TRUE. LHARD (6) = .FALSE.

TFSVP	Test for a default system viewport separate from the working area.	LSYSVP = .TRUE. if
		LSQUAR = .TRUE.
		LSYSVP = .FALSE. if
		LSQUAR = .FALSE.
		
		lsquar is a parameter of INITIG.

TFSYM	Test for maximum symbol number for which a symbol is defined.	NSYM = 13
TFVIS	Test to see if a complete retransmission is required to make a segment invisible.	LTRAN = .TRUE.

TFXFA (≡ TFXFR and TFXFL)	Test for existence of display processor hardware to perform transformations to a 2-D window.	LXLAT = .FALSE. (2-D translate capability)
		LSCAL = .FALSE. (2-D scale capability)
		LROT = .FALSE. (2-D rotation capability)

TFXFA3 (≡ TFXFR3 and TFXFL3)	Test for existence of display processor hardware to perform transformations to a 3-D window.	LXFM3 = .FALSE. (3-D translation, rotation, and scaling) LPERSP = .FALSE. (3-D perspective preservation during transformations) LPYRAM = .FALSE. (clipping to 3-D pyramid during transformations)

†Default character size for given value of **lsquar** parameter.
††Refer to Rotation of Text in this appendix for implementation of text rotation for Tektronix 401x terminals.

TEKTRONIX TERMINALS

This subsection is divided into five parts:

- 4006 terminals.
- 4010 series terminals.
- 4014 terminals.
- 4014 terminals with EGM.
- **Optional hardware for 401x terminals.**

It is assumed that the operator has access to the Tektronix publications dealing with his terminal. The publications are available from the manufacturer.

TEKTRONIX 4006 TERMINAL

The Tektronix 4006 display terminal contains a display screen, function switches, control indicators, and an alphanumeric keyboard. The power switch and the hardcopy intensity control for the 4006-1 are located on the back of the terminal. The hardcopy intensity control is a screw adjustment that, when turned clockwise, increases the intensity of hardcopies produced by an optional hardcopy device. For further explanation of the features and options available refer to the Tektronix 4006, 4006-1 Terminal User's Guide.

Strap Options

Strap options on the 4006 logic cards permit the operator to change the operating configuration of the terminal. Table F-4 lists these strap options, and the options that must be selected to enable the terminal to communicate with TIGS. The correct option choice is circled for each feature. The location of these straps is shown in the terminal user's guide. It should be noted that the options necessary for communication with TIGS are in some cases not the option strapped at the factory when the logic card is shipped.

Alphanumeric Mode

For alphanumeric input and output in alphanumeric mode the Tektronix 4006 terminal displays any of the printing characters available in the character set in appendix A. Lowercase characters are printed as uppercase. A cursor is displayed on the screen indicating the next character position. The display screen allows up to 35 lines of up to 74 characters each. Backspacing is accomplished by pressing the CTRL and H keys simultaneously.

The home position for the cursor is in the upper lefthand corner of the display screen. The display automatically sets the cursor to the home position upon powering up the terminal or whenever the PAGE key is pressed. The operator presses the PAGE key whenever the screen is to be cleared.

TABLE F-4. STRAP OPTIONS FOR 4006 TERMINAL

Feature	Description
CR-LF	[✓ 1.] Carriage Return does not cause line feed. 2. **Carriage Return does cause automatic line feed.**
ECHO/NORM†	1. ECHO–Display data is provided by the terminal logic. [✓ 2.] NORM–Display data is returned to the terminal by modem or computer.
Data Strap	1. Bit 8 is selected as a data bit. [✓ 2.] Bit 8 is selected as a parity bit. This strap is used with the Data Select strap described below.
Data Select	[✓ 1.] High (even). If the Data Strap is set to BIT 8, bit 8 is set high. If the Data Strap is set to PARITY, even parity is selected. 2. Low (odd). If the Data Strap is set to BIT 8, bit 8 is set low. If the Data Strap is set to PARITY, odd parity is selected.
Interface Connector	1. Direct Connect to computer. [✓ 2.] Telephone/Modem connection to computer.

†Replaced by a rear panel switch if the optional half-duplex interface is installed.

The 4006 screen is divided into two vertical pages. The left side of the screen is the left margin for 35 lines of data. The center of the screen becomes the left margin for the next 35 lines. After both vertical pages are filled, the screen must be cleared by the operator so data is not overwritten. Lines in the first page over 36 characters long extend into the second page. These long lines can be overwritten by lines centered on the second; therefore, operators frequently clear the screen after the first page of data is full.

The operator terminates each line of input by pressing the RETURN key (same function as the carriage return key on a teletypewriter) which signals the end of an input line, transmits the input line to the computer, advances the line position one line, and places the cursor at the lefthand margin ready for more input.

These margin and alphanumeric input and output considerations have meaning only in alphanumeric mode. When TIGS is initialized, the screen is cleared and data is written to and occasionally read from the terminal according to the formats specified by the TIGS subroutine calls. There are no preset margins when in graphics mode. The terminal is in alphanumeric mode:

- When logging in or entering NOS/BE INTERCOM or NOS NAM/IAF or Time-Sharing Module commands.
- When a graphics program calls KEYBRD requesting character input from the terminal.
- When a graphics program calls TEXT or PROMPT to write text.

Locator Device

The 4006 has no locator device and any application program requiring a locator is issued an error diagnostic by TIGS when a request is made. Since the default locator is 1, the programmer should call SMLOCR (0) before calling EVENT and should not call LOCATE. EVENT can be called for function key presses only, not segment picks.

Function Key Usage

Numeric keys 1 through 9 and 0 serve as function keys. When the EVENT subroutine is called and KYON or KYAC has been previously called, the operator may press any of the function keys followed by a carriage return. Since there are no locator crosshairs on a 4006 to prompt the operator to respond to the EVENT call, the message

WAITING FOR FUNCTION KEY PRESS AND CR

is written in the system viewport area.

Mode of Communication

The Tektronix 4006 terminal is supported at 300 baud to 9600 baud, in asynchronous mode only. It is the operator's responsibility to set the terminal baud rate correctly for the communications line he is using.

Options

Refer to Optional Hardware for Tektronix 401x Terminals in this appendix for specific usage information for the following option:

- Tektronix 4631 hardcopies.

TEKTRONIX 4010 SERIES TERMINAL

This discussion includes the 4012 and 4013 terminals.

The Tektronix 4010 display unit contains a pedestal unit, keyboard, function switches, and control indicators. The pedestal contains two controls: display on/off switch and hardcopy intensity control (4010-1). The display contains a display screen, function switches, control indicators, alphanumeric keyboard, and crosshair thumbwheels. More information about the terminal is available in the Tektronix 4010, 4010-1 User's Manual.

Strap Options

Strap options on the 4010 series terminals permit the operator to change the operating configuration of the terminal. Table F-5 lists these strap options, and the options that must be selected to enable the terminal to communicate with TIGS. The correct option choice is circled for each feature. The location of these straps is shown in the terminal user's guide. It should be noted that the options necessary for communication with TIGS are in some cases not the options strapped at the factory when the logic card is shipped.

TABLE F-5. STRAP OPTIONS FOR 4010 SERIES TERMINALS

Feature	Description
Character type (if alternate character memory is installed)	[✓ 1.] Normal characters.
	2. Alternate or normal characters with switch 2 (on front panel).
	3. Alternate or normal characters with SO or SI control characters.
Linefeed	1. Linefeed only.
	[✓ 2.] CR with Linefeed.
Graphic Input Terminators	1. CR, and EOT (CTRL D).
	2. CR only (Normal Position).
	[✓ 3.] No CR, No EOT.
Page Full Busy	[✓ 1.] Out.
	2. In (Makes 4010 Busy) (Normal Position).
Local Echo	[✓ 1.] In.
	2. Out.
Page Full Break	[✓ 1.] Out (Normal Position).
	2. In.
Baud Rate†	1. R (Receive) 150, 300, 600, 1200, 2400, 4800, or 9600 baud.
	2. T (Transmit) 150, 300, 600, 1200, 2400, 4800, or 9600 baud.
Modem/Computer Connections	1. To computer direct.
	[✓ 2.] To Modem (turn the cable plug over). (Normal Position).

†Depends on communication line baud rate.

Alphanumeric Mode

For alphanumeric input and output in alaphanumeric mode the Tektronix 4010 series terminals display any of the printing characters available in the character set in appendix A. Lowercase characters are printed as uppercase. A cursor is displayed on the screen indicating the next character position. The display screen allows up to 35 lines of 75 characters each. Backspacing is accomplished by pressing the BACKSPACE key when in the asynchronous communications mode or by pressing the RUBOUT key when in synchronous mode.

The home position for the cursor is in the upper lefthand corner of the display screen. The display automatically sets the cursor to the home position upon powering up the terminal or whenever the PAGE key is pressed. The operator presses the PAGE key whenever the screen is to be cleared.

The 4010 screen is divided into two vertical pages. The left side of the screen is the left margin for 35 lines of data. The center of the screen becomes the left margin for the next 35 lines. After both vertical pages are filled, the screen must be cleared by the operator so data is not overwritten. Lines in the first page over 36 characters long extend into the second page. These long lines can be overwritten by lines centered on the second; therefore, operators frequently clear the screen after the first page of data is full.

The operator terminates each line of input by pressing the RETURN key (same function as the carriage return key on a teletypewriter) which signals the end of an input line, transmits the input line to the computer, advances the line position one line, and places the cursor at the lefthand margin ready for more input.

These margin and alphanumeric input and output considerations have meaning only in alphanumeric mode. When TIGS is initialized, the screen is cleared, and data is written to and occasionally read from the terminal according to the formats specified by the TIGS subroutine calls. There are no preset margins when in graphics mode. The terminal is in alphanumeric mode:

- When logging in or entering NOS/BE INTERCOM or NOS NAM/IAF or Time-Sharing Module commands.
- When a graphics program calls KEYBRD requesting character input from the terminal.
- When a graphics program calls TEXT or PROMPT to write text.

Locator Device

Crosshair Cursor

The operator can use the Tektronix crosshair cursor for screen coordinates input to the application program and for making selections of, displayed objects on the screen (segment picks) if the crosshair cursor is the mode set locator. The crosshair cursor is the default locator.

The crosshair cursor is displayed on the screen whenever the LOCATE or EVENT subroutine is called. The **idvuwi** parameter on a PRELOC or PREEVN call has no effect on the positioning of the crosshairs. The crosshairs appear where last positioned manually. The crosshair cursor location (intersection of the crosshairs) can be positioned anywhere on the screen by use of the two thumbwheel controls to the right of the alphanumeric keys.

When an application program calls the LOCATE subroutine, the crosshairs appear on the screen and the program waits for the operator to position the cursor at a desired location and to press an alphanumeric key. The character T terminates a set of input coordinates and any other character indicates to TIGS this point is not the last point in a series of locations (refer to LOCATE, section 7).

When an application program calls the EVENT subroutine, the crosshairs appear on the screen. The operator may press a function key (numeric keys 0 through 9) or pick a displayed object. To pick an object, the operator places the crosshair cursor over any visible part of an object to be selected and presses any alphabetic key. If the operator picks a nonpickable object (a segment with an IGNORE action attribute), the crosshairs reappear and EVENT continues waiting until a valid selection is made. Each valid selection is echoed by redrawing. A segment or function key with TERMINATE action terminates the input sequence.

Tablet

The operator can use the optional Tektronix 4953 or 4954 Tablet for screen coordinates input to the application program and for making selections of displayed objects on the screen provided the tablet is the mode set locator (refer to Optional Hardware for Tektronix 401x Terminals in this appendix and to SMLOCR, section 7).

The tablet is enabled for input whenever the EVENT or LOCATE subroutine is called. The **idvuwi** parameter on a PRELOC or PREEVN call has no effect on the positioning of the pen; it is positioned by the operator.

When an application program calls the LOCATE routine, the program waits for the operator to press and release the pen tip on the tablet.

The corresponding pen position is indicated on the terminal display screen with a cursor (NOS/BE only). The coordinates of this point are placed on the location queue. To terminate a set of locations, the operator must press the T and RETURN keys on the keyboard. The application program waits until the T, RETURN sequence is pressed.

When an application program calls the EVENT subroutine, the operator can press a function key (numeric key 0 through 9) or pick a displayed object. To pick an object, the operator positions the pen to a position corresponding to a visible portion of the object and presses and releases the pen tip. If the operator picks a nonpickable object (a segment with an IGNORE action attribute), the operator is required to select another object and EVENT continues waiting until a valid selection is made.

Function Key Usage

Numeric keys 1 through 9 and 0 serve as function keys. When the EVENT subroutine is called and KYON or KYAC has been previously called, the operator can press any of the function keys. If the crosshair cursor is the mode set locator, the crosshairs serve as a prompt indicating a function key can be pressed. If the tablet is the mode set locator, the READY light on the tablet controller signals a function key can be pressed.

Mode of Communication

The Tektronix 4010 series terminal is supported at 300 to 9600 baud in asynchronous mode and at 2000, 2400, and 4800 baud in synchronous communications mode with option 20.

Options

Refer to Optional Hardware for Tektronix 401x Terminals in this appendix for specific usage information for the following options:

- Tektronix 4631 Hard Copy Unit.
- Tektronix 4953 or 4954 Tablet.
- Tektronix Option 20 Synchronous Interface.

TEKTRONIX 4014 TERMINAL

Information in this section is also applicable to terminals capable of emulating the 4014. However, keyboards and input devices on terminals in 4014 emulation mode do not necessarily correspond to information given here for the 4014; the operator is assumed to be familiar with the operation of a terminal in 4014 emulation mode.

The Tektronix 4014 terminal is a display and pedestal unit. The pedestal contains the display on/off switch and the display contains a display screen, function switches, control indicators, alphanumeric keyboard, and crosshair cursor position controls. Several switches and indicators are not used unless certain strappable options are connected to the terminal. Also, the AUTO PRINT/COPY switch and the screw adjustment for hardcopy intensity apply to the 4014-1 only. For further information, refer to the Tektronix 4014 and 4014-1 Computer Display Terminal User's Instruction Manual.

Strap Options

Strap options on the 4014 terminal permit the operator to change the operating configuration of the terminal. Table F-6 lists these strap options, and the options that must be selected to enable the terminal to communicate with TIGS. The correct option choice is circled for each feature. The location of these straps is shown in the terminal user's guide. It should be noted that the options necessary for communication with TIGS are in some cases not the options strapped at the factory when the logic card is shipped.

TABLE F-6. STRAP OPTIONS FOR 4014 TERMINAL

Feature	Effect
LF Effect	[✓ 1.] LF causes Line Feed only.
	2. LF-CR causes Line Feed and Carriage Return.
CR Effect	[✓ 1.] CR causes Carriage Return only.
	2. CR-LF causes Carriage Return and Line Feed.
DEL Implies LOY	[✓ 1.] DEL-LOY permits RUBOUT (DEL) to be used as LOY.
	2. DEL permits ESC ? to be substituted for DEL.
Graphic Input Terminators	1. CR and EOT transmits CR and EOT in GIN Mode.
	[✓ 2.] CR transmits CR in GIN Mode.
	3. NONE transmits neither CR nor EOT in GIN Mode.
Switch 3 and Indicator 3 (LED 3) connections	[✓ 1.] No effect; disconnected.
	2. Provides ground connections via Switch 3 from keyboard and provides +5 V to Indicator 3 on keyboard.

Alphanumeric Mode

Without the Enhanced Graphic Module (EGM), the Tektronix 4014 has nearly the same alphanumeric input and output characteristics as the Tektronix 4010. The 4014 can display 35 lines of alphanumeric input or output, each line containing 74 characters. Unlike the 4010, the 4014 displays characters in lowercase that are typed in lowercase. The RUB OUT key is used for backspacing when in synchronous communication mode and the BACKSPACE key when in asynchronous mode. The terminal operator can switch back and forth between character sizes by:

- Switching the LOCAL/LINE switch to LOCAL (temporarily cuts off communication with the host computer).
- Pressing the ESC key.
- Pressing the key to determine character size.

8	large size
9	medium size
:	normal size
;	small size

- Switching the LOCAL/LINE switch back to LINE.

Whenever the Tektronix 4014 terminal is on, the screen displays a cursor. The home position for the cursor is the upper lefthand corner of the display screen. The cursor is automatically set to the home position when power is applied to the terminal, and whenever the PAGE key is pressed. The operator presses the PAGE key whenever the screen is to be cleared.

The 4014 screen is divided into two vertical pages. The left side of the screen is the left margin for 35 lines of data. The center of the screen becomes the left margin for the next 35 lines. After both vertical pages are filled, the screen must be cleared by the operator so data is not overwritten. Lines in the first page over 36 characters long extend into the second page. These long lines can be overwritten by lines centered on the second; therefore, operators frequently clear the screen after the first page of data is filled. The operator terminates each line of input by pressing the RETURN key (same function as the carriage return key on a teletypewriter) which signals the end of an input line, transmits the input line to the computer, advances the line position one line, and places the cursor at the lefthand margin ready for more input.

These margin and alphanumeric input and output considerations have meaning only in alphanumeric mode. When TIGS is initialized, the screen is cleared, and data is written to and occasionally read from the terminal according to the formats specified by the TIGS subroutine calls. There are no preset margins when in graphics mode. The terminal is in alphanumeric mode:

- When logging in or entering NOS/BE INTERCOM or NOS NAM/IAF or Time-Sharing Module commands.
- When a graphics program calls KEYBRD requesting character input from the terminal.
- When a graphics program calls TEXT or PROMPT to write text.

Locator Device

Crosshair Cursor

The operator can use the Tektronix crosshair cursor for screen coordinates input to the application program and for making selections of displayed objects on the screen (segment picks) if the crosshair cursor is the mode set locator. The crosshair cursor is the default locator.

The crosshair cursor is displayed on the screen whenever the LOCATE or EVENT subroutine is called. The **idvuwi** parameter on a PRELOC or PREEVN call has no effect on the positioning of the crosshairs. The crosshairs appear where last positioned manually. The crosshair cursor location (intersection of the crosshairs) can be positioned anywhere on the screen by use of the two thumbwheel controls to the right of the alphanumeric keys.

When an application program calls the LOCATE subroutine, the crosshairs appear on the screen and the program waits for the operator to position the cursor at a desired location and to press an alphanumeric key. The character T terminates a set of input coordinates and any other character indicates to TIGS this point is not the last point in a series of locations (refer to LOCATE, section 7).

When an application program calls the EVENT subroutine, the crosshairs appear on the screen. The operator may press a function key (numeric keys 0 through 9) or pick a displayed object. To pick an object, the operator places the crosshair cursor over any visible part of an object to be selected and presses any alphabetic key. If the operator picks a nonpickable object (a segment with an IGNORE action attribute), the crosshairs reappear and EVENT continues waiting until a valid selection is made. Each valid selection is echoed by redrawing. A segment or function key with TERMINATE action terminates the input sequence.

Tablet

The operator can use the optional Tektronix 4953 or 4954 Tablet for screen coordinates input to the application program and for making selections of displayed objects on the screen provided the tablet is the mode set locator (refer to Optional Hardware for Tektronix 401x Terminals in this appendix).

The tablet is enabled for input whenever the EVENT or LOCATE subroutine is called. The **idvuwi** parameter of a PRELOC or PREEVN call has no effect on the positioning of the pen; it is brought into use by the operator.

When an application program calls the LOCATE routine, the program waits for the operator to press and release the pen tip on the tablet. The corresponding pen position is indicated on the terminal display screen with a cursor (NOS/BE only). The coordinates of this point are placed on the locate queue. To terminate a set of locations, the operator must press the T and the RETURN keys on the keyboard. The application program waits until the T, RETURN sequence is pressed.

When an application program calls the EVENT subroutine, the operator can press a function key (numeric key 0 through 9) or pick a displayed object. To pick an object, the operator positions the pen to a position corresponding to a visible portion of the object and presses and releases the pen tip. If the operator picks a nonpickable object (a segment with an IGNORE action attribute), the operator is required to select another object and EVENT continues waiting until a valid selection is made.

Function Key Usage

Numeric keys 0 through 9 serve as function keys. When the EVENT subroutine is called and KYON or KYAC has been previously called, the operator can press any of the function keys. If the crosshair cursor is the mode set locator, the crosshairs serve as a prompt indicating a function key can be pressed. If the tablet is the mode set locator, the READY light on the tablet controller signals a function key can be pressed.

Mode of Communication

The Tektronix 4014 terminal is supported at 300 to 9600 baud in asynchronous mode and at 2000, 2400, and 4800 baud in synchronous communication mode with option 20.

Options

Refer to Optional Hardware for Tektronix 401x Terminals in this appendix for specific usage information for the following options.

- Tektronix 4631 Hardcopy Unit.
- Tektronix 4953 or 4954 Tablet.
- Tektronix Option 20 Synchronous Interface.

TEKTRONIX 4014 TERMINAL WITH EGM

This section deals with the 4014 terminal equipped with the Enhanced Graphics Module (EGM). All of the material on the 4014 terminal without EGM is valid for the 4014 with EGM as well. The following material documents the additional capabilities that EGM hardware provides for the 4014 terminal.

Strap Options

The strap option selections described in table F-7 should be made in addition to those described for the 4014.

TABLE F-7. STRAP OPTIONS FOR 4014 TERMINALS WITH EGM

Feature	Location	Description
TIMING	EGM logic board	1 4010/4012
		[✓ 2] 4014
WRITE	EGM logic board	1 4010/4012
		[✓ 2] 4014

The EGM provides the following additional capabilities to the 4014 terminal.

- Expanded graphic resolution – a 12-bit graphic resolution allowing 4096 by 3120 addressable points as opposed to 1024 by 780 addressable points for a terminal without EGM.
- Five line styles – line styles that are hardware (as opposed to software) generated are:

Line Style	TIGS Style Number
Solid	1
Long-dash	2
Short-dash	3
Dash-dot	4
Dot	5

Two TFxxxx routines, TFSTYL and TFSCRN, return different values depending on whether a 4014 terminal is equipped with EGM or not, to reflect these added capabilities.

OPTIONAL HARDWARE FOR TEKTRONIX 401X TERMINALS

Tektronix Hardcopy Unit - 4631

The 4631 hardcopy unit is compatible with all Tektronix terminals supported by the 401x postprocessor. The unit can be shared by up to four terminals if the multiplexer option is available. Pressing the copy button on the front panel of the terminal causes the image on the screen to be copied on the hardcopy unit. Calling the TIGS routine REMSCR has the same results. For further information on the 4631 hardcopy unit refer to the Tektronix 4631 Hardcopy Unit User's Manual.

Tektronix Tablets - 4953 and 4954

The 4953 and 4954 Graphics Tablets send graphic data to a computer through a Tektronix 4010 series or 4014 Graphic Display Terminal. The devices are identical except for the size of the tablets; the 4953 (small) is 11 inches by 11 inches and the 4954 (large) is 40 inches by 30 inches.

Points picked by the tablet pen are digitized and sent to the computer. When the operator moves the pen on the tablet, a cursor indicates on the screen a corresponding position. When the pen is pressed and released, the coordinates of the lower left corner of this cursor are transmitted to TIGS. The cursor for tracking the pen position appears only when running on NOS/BE. Refer to the 4010 and 4014 terminal descriptions for TIGS application of the tablet.

Strap options must be selected as indicated in table F-8 for tablet input. These strap options are located on a separate tablet logic board. If not strapped as indicated, no assurance of proper operation is guaranteed. For further information, refer to Tektronix 4953/4954 Graphics Tablet Instruction Manual.

Synchronous Interface (Option 20)

This interface enables synchronous communication between a 4010 series or 4014 Graphics Display Terminal and the CDC CYBER 70, CDC CYBER 170, or 6000 series computer. The Tektronix terminal simulates a CDC 200 User Terminal in synchronous mode.

All strap options for the synchronous interface are as set by the factory, except for the C option which should be completely disconnected. The factory set positions are described in the Tektronix Data Communications Interface Instruction Manual.

To switch a 4010 series or 4014 series terminal back and forth between synchronous and asynchronous communication modes, do the following.

TABLE F-8. STRAP OPTIONS FOR 4953 OR 4954 GRAPHICS TABLET

Feature	Description	
DELAY	1	Out
	[✓ 2]	In
COMSUP	[✓ 1]	In
	2	Out
ESUP	[✓ 1]	In
	2	Out
CR	1	Out
	[✓ 2]	In
HEADER	[✓ 1]	Letter (NOS)
	[✓ 2]	Control (NOS/BE)
SMALL/ LARGE (set for appropriate tablet)	1	small
	2	large
STATUS	1	In
	[✓ 2]	Out

Synchronous Operation

1. Set LOCAL/LINE switch to LINE.
2. Set CODE EXPANDER switch to ON.
3. Set CLEAR WRITE switch to OFF.
4. Set ASCII/ALT switch to ASCII.
5. Push TTY LOCK key on (down position).
6. Set BAUD RATE rotary switch (back of terminal) to EXT.
7. Set BCD/ASCII switch (back of terminal) to ASCII.
8. Push RESET PAGE key after the power is turned on and before dialing up the computer.

Asynchronous Operation

1. Set BAUD RATE rotary switch (back of terminal) to desired speed, i.e., 300.
2. Set BCD/ASCII switch (back of terminal) to ASCII.
3. Set CODE EXPANDER switch to OFF.
4. Set ASCII/ALT switch to ASCII.
5. Push TTY LOCK key on (down position).
6. Set LOCAL/LINE switch to LOCAL.
7. Simultaneously press RESET PAGE and SHIFT keys.
8. Simultaneously press shift, CTRL and P keys.
9. Set LOCAL/LINE switch to LINE.

SANDERS GRAPHIC 7 POSTPROCESSOR

This appendix describes the postprocessor for the Sanders Graphic 7 terminal. It is organized in the following manner.

- A subsection on hardware/operating system environments for the postprocessor.
- A subsection on TIGS software features whose implementation is dependent on the postprocessor used, including a list of values returned from TFxxx routines for the Sanders postprocessor.
- A subsection containing information about the terminal and hardware options, and their use.
- A subsection describing the host to terminal loading procedure.
- A subsection on notes and cautions.

OPERATING ENVIRONMENT

NOS/BE AND INTERCOM SYSTEM USER'S HARDWARE ENVIRONMENT

Users with access to a CDC CYBER 70, CDC CYBER 170, or 6000 Series Computer running under NOS/BE and INTERCOM can run TIGS with the Sanders Graphic 7 postprocessor and terminal. The minimum hardware configuration needed to run TIGS is the same as that needed to run NOS/BE and INTERCOM. The terminal must be connected to the host computer by a telephone line through a data set.

Communication is supported at baud rates of 300 and 1200 in an asynchronous mode.

NOS NAM/IAF AND NOS TIME-SHARING MODULE SYSTEM USER'S HARDWARE ENVIRONMENT

Users with access to a CDC CYBER 70, CDC CYBER 170, or 6000 Series Computer running under NOS NAM/IAF or NOS with the Time-Sharing Module (refer to appendix E) can run TIGS with the Sanders Graphic 7 postprocessor and terminal. The minimum hardware configuration needed to run TIGS is the same as that needed to run NOS. The terminal must be connected to the host computer by a telephone line through a data set.

Communication is supported at baud rates of 300 and 1200 in an asynchronous mode.

GRAPHIC 7 POSTPROCESSOR LIBRARY

The Graphic 7 postprocessor is selected by declaring the proper library in the graphics job before program loading takes place. This process is described in appendix E. After the Graphic 7 postprocessor library has been selected and program execution has been initiated, a display of installation defined hardware options will appear. (Refer to the installation procedure for further details.)

An example display is as follows:

```
INSTALLATION DEFAULT VALUES ARE

LIGHTPEN IS AVAILABLE
TRACKBALL/FORCESTICK IS AVAILABLE
HARD COPY UNIT IS NOT AVAILABLE
ERROR DISPLAY IS ENABLED

DO YOU ACCEPT THESE DEFAULTS (Y/N)
```

If the user replies with a Y carriage return, the postprocessor will continue the initialization process. If the user entry is an N carriage return, the postprocessor will query the user with the following questions.

```
    IS LIGHTPEN AVAILABLE (Y/N)

    IS TRACKBALL/FORCESTICK AVAILABLE (Y/N)

    IS HARD COPY UNIT AVAILABLE (Y/N)

    SHOULD ERROR DISPLAY BE ENABLED (Y/N)
```

The four questions appear one by one, each appearing only after the last has been properly answered. An improper answer causes the question to be repeated. It is important to realize that only invalid answers are rejected, not incorrect ones.

TIGS SOFTWARE FEATURES

The following TIGS software features are dependent on the postprocessor for specifics of implementation:

- Default initialization.
- Supported plotting symbol set.
- Supported continuous character set.
- Line style algorithm.
- Echoing segment picks.
- Implementing rotation of text.
- Determining the viewport for PROMPT routine use.
- Returning coordinate values on an EVENT call.
- Erasing the screen.
- Determining default screen layout.
- Keyboard input maximum

TIGS also contains a set of subroutines that test postprocessor features. This subsection contains a table of values returned by these TFxxxx routines for the Sanders postprocessor.

DEFAULT INITIALIZATION

If an applications program does not make a call to INITIG, the first TIGS routine called in the program results in a call to INITIG with the following default values.

```
CALL INITIG(.TRUE.,.TRUE.,6LNEWNDF)
```

SUPPORTED PLOTTING SYMBOL SET

The Sanders postprocessor supports the plotting symbol set as defined in the xxSYM routine documentation, section 2.

SUPPORTED CONTINUOUS CHARACTER SET

The Sanders postprocessor supports the continuous character set defined in the xxCSIZ routine documentation, section 2.

LINE STYLE ALGORITHM

The algorithm used in implementing the bit-pattern line style of the SMSTYL and RASTYL (istyle>5) is as follows: for each set bit of the binary representation of the **istyle** value, the drawing beam is turned on for the distance of 0.0034188 in terminal-independent coordinate system units. In general, a single set bit results in a dot; several set bits in a row result in a dash that is directly proportional in length to the number of bits set. For example, the value

```
ISTYLE=7652B
```

is equivalent to the binary bit pattern

```
111110101010
```

In this case, the drawing beam would be turned on for five screen units in a row, then alternating off and on for the next seven units before the pattern is repeated. This results in a line style consisting of a dash followed by three dots.

The hardware supported line style of dash-dotted should not be used for arc generation. This style makes arcs appear solid. If a nonsolid arc is desired, use either a software bit pattern or the hardware style of short dashed.

ECHOING SEGMENT PICKS

Segment picks are echoed at the terminal by turning the visibility of the picked segment off and then on again. In the case where the picked segment is to be deleted, it will be seen to disappear, reappear, and finally disappear.

ROTATION OF TEXT

Hardware rotation of text for angles of exactly +90° is supported by the Sanders postprocessor. Other angles of character rotation are implemented by TIGS software. With software rotation, the character itself is not rotated; rather, the angle of the lower left corner of the character in relation to the next character is positioned at the set angle of rotation. If text rotation of +90° is desired but rotation of individual characters is not, the user may simply specify a rotation angle very close to, but not equal to, +90° (for example, +90.001°). Figure G-1 is an example of character rotation for the Graphic 7 post-processor. Positive angles of rotation are counterclockwise and negative angles of rotation are clockwise from the positive x axis. Refer to TEXT in section 2.

DETERMINING VIEWPORT FOR PROMPT ROUTINE USE

The Sanders postprocessor uses the smallest hardware character size for system viewport messages. The system viewport is in a fixed location on the screen above the working area. It consists of one line of text, a maximum of 100 characters in length. The user is not allowed to move the system viewport or change its size; calls to SMSVP are ignored by this postprocessor. Consecutive PROMPT calls should be used with care since the display of each system viewport message causes the preceding line to be removed first. Prompting for user interaction is handled automatically by the postprocessor and is displayed in a special area in the upper right corner of the screen.

RETURNING COORDINATE VALUES OF AN EVENT CALL

Processing of segment picks is performed in one of two ways, depending upon the locator used. Lightpen picks are processed by the terminal controller. This type of processing is fast but does not have the capability of returning to the user the exact coordinates of the picked point. The values returned in the COORDS array will be zero in this case.

Segment picks via the trackball/joystick are processed by finding the exact location picked and searching the Neutral Display File for a graphic primitive at this location. Although less time-efficient, this method will return the exact coordinates of the picked point in the COORDS array.

If both locators are available, the user should decide which one is appropriate to use in a specific situation with regard to the preceding considerations.

SCREEN ERASURE

Because the Sanders terminal is a refresh device, complete screen erasure is never necessary. Items on the screen may be selectively deleted. Calls to CLRSCR have no function under this postprocessor and are ignored.

DEFAULT SCREEN LAYOUT

The screen layout used by the Sanders postprocessor is independent of the value of the logical input parameter **lsquar** in the call to INITIG. The default viewport is always square with lower left and upper right coordinates of (0.,0.) and (1.,1.) respectively, as shown in figure G-2. Points outside this area cannot be addressed by the TIGS user. Other important fixed areas on the screen are:

- System viewport area.
- User interaction request area.
- Keyboard input scratchpad area.
- Alarm area.

The system viewport area has already been discussed. The user interaction request area is used to indicate that the application program is waiting for user input of a specific type: keyboard, function key, lightpen pick, lightpen

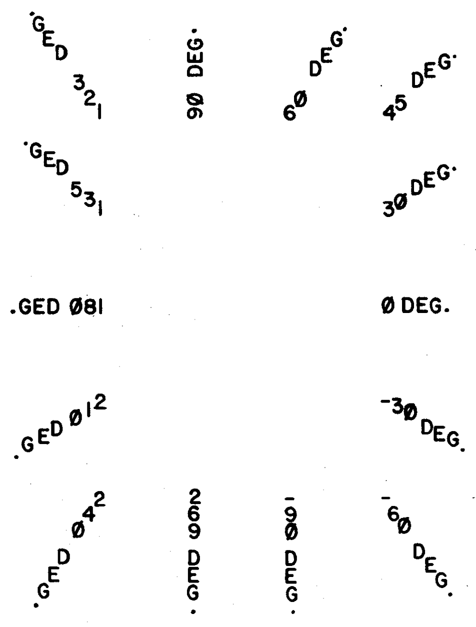


Figure G-1. Example of Character Rotation

locate, trackball pick, or trackball locate. If either a lightpen pick or a trackball pick is requested, the user has the option of pressing a function key instead.

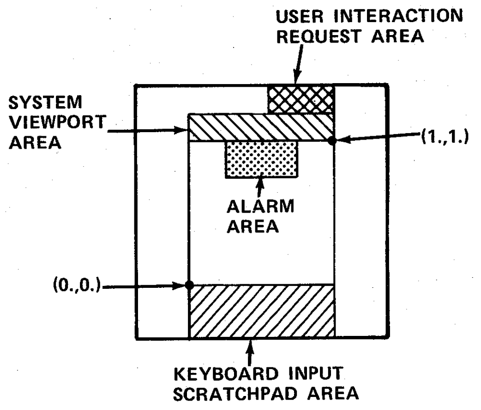


Figure G-2. Screen Layout

The keyboard input scratchpad area is used for echoing keyboard input on the screen whenever it is being requested. The alarm area is where the word ALARM will appear flashing on and off in the largest hardware character size when a call to the ALARM routine is made.

KEYBOARD INPUT

The Sanders postprocessor supports an input of 62 characters for each KEYBRD call.

TEST FEATURE ROUTINES

Table G-1 lists the TFxxx subroutines and the values returned for them from the Sanders postprocessor.

TABLE G-1. TEST FEATURE ROUTINES

Test Feature Routine Name	Feature Tested	Returned Value from Routine
TFAC	Support of EVENT routine.	LACTN=.TRUE.
TFCSIZ	Support of continuous character sizes	LCCHAR=.TRUE.
TFDSIZ	Discrete character size.	Characters/line: 93† Lines/frame: 62 WIDOUT = 10./931. HIOUT = 15./931. Characters/line: 62 Lines/frame: 41 WIDOUT = 15./931. HIOUT = 22.5/931. Characters/line: 46 Lines/frame: 31 WIDOUT = 20./931. HIOUT = 30./931. Characters/line: 31 Lines/frame: 20 WIDOUT = 30./931. HIOUT = 45./931.
TFERR	Support of user-supplied error routine.	LROUTN = .TRUE.
TFFONT	Number of fonts supported.	NFONTS = 1.
TFHARD	Existence of remote copiers.	If fourth question (that is, availability of hardcopy unit) during initialization dialogue was answered NO, LREMOT = .FALSE. If answered YES, LREMOT = .TRUE.
TFHILT	Support of highlighting.	LHILT = .TRUE.
TFID	Support of intrasegment identifiers.	NID = 32767. (not supported for lightpen picks.)
TFINT	Number of hardware intensity levels supported.	NINTEN = 8.
TFLOCR	Number of locators available and their characteristics.	If no locators available, NLOCRS = 0. If lightpen available, NLOCRS = 1, DESCRP(1) = 1001 (lightpen), LONE(1) = .FALSE. If trackball available, NLOCRS = 1, DESCRP(1) = 1002 (trackball), LONE(1) = .TRUE. If lightpen and trackball available, NLOCRS = 2, DESCRP(1) = 1001 (lightpen), DESCRP(2) = 1002 (trackball), LONE(1) = .FALSE., LONE(2) = .TRUE.
TFNSIZ	Number of discrete hardware character sizes.	NDSIZE = 4.
TFPICT	Maximum number of pictures allowed.	NPICT = 509.
TFPORT	Maximum number of viewports allowed.	NPORT = 508.
TFROT	Support of 90° character rotation and continuous character rotation.	LNINTY = .TRUE. LCONT = .TRUE.††

TABLE G-1. TEST FEATURE ROUTINES (Contd)

Test Feature Routine Name	Feature Tested	Returned Value from Routine

TFSCRN	Size, shape, and resolution of the screen.	LRTNGL = .TRUE.
		XLL = YLL = 0.
		XUR = YUR = 1.
		RESLTN = 931.

TFSTYL	Hardware-supported line styles.	LHARD(1) = .TRUE. LHARD(2) = .TRUE. LHARD(3) = .TRUE. LHARD(4) = .TRUE. LHARD(5) = .FALSE. LHARD(6) = .FALSE.
TFSVP	Default system viewport separate from working area.	LSYSVP = .TRUE.
TFSYM	Maximum symbol number for which a symbol is defined.	NSYM = 13.
TFVIS	Retransmission required to make segment invisible.	LTRAN = .FALSE.

TFXFA	Existence of 2-D hardware transformations.	LXLAT = .FALSE. (2-D translate capability)
		LSCAL = .FALSE. (2-D scale capability)
		LROT = .FALSE. (2-D rotation capability)

TFXFA3	Existence of 3-D hardware transformations.	LXFM3 = .FALSE. (3-D translation, rotation and scaling) LPERSP = .FALSE. (3-D perspective preservation during transformations) LPYRAM = .FALSE. (clipping to 3-D pyramid during transformations)

†Default character size.
††Refer to Rotation of Text in this appendix for implementation of text rotation for Sanders terminal.

SANDERS HARDWARE

TERMINAL DESCRIPTION

The Sanders Graphic 7 terminal is an intelligent interactive graphics terminal using a refreshed strokewriting CRT as a display. The basic hardware consists of a CRT display, a terminal controller unit, and an alphanumeric keyboard with function keys. The controller contains Sanders GSS4 software for handling communication between terminal and host, as well as refresh memory for storing graphics information displayed on the screen. Optional hardware available with the Sanders terminal consists of two different locators - a lightpen, and a trackball or a forcestick - as well as a hardcopy unit. At least one locator should be available if the full interactive capability of TIGS is to be used.

Both the display and controller have on/off switches in the front; on the controller, however, the on/off switch is behind a removable front panel. On the front of this panel are two other switches, the system mode and local mode switches, which will be explained in the next section. The display has a row of three knobs on the front for adjusting contrast, focus, and intensity.

OPERATING PROCEDURE

The Sanders terminal has several modes of operation of which the operator should be aware. Switching from one mode to another can be done either manually at the terminal by the operator, or under the control of an application program in the host. To enter either local mode or system mode manually, the terminal operator needs simply to push the correspondingly labeled button on the front panel of the controller unit. The operator should normally have no need to enter system mode manually. When a TIGS application program begins to execute, TIGS software will automatically cause the host to send the necessary commands to the terminal to put it into system mode and transfer control to GSS4 software. Once in this mode, any graphics commands sent to the terminal by the host will be properly interpreted and displayed.

Entering local mode manually causes a test pattern to be displayed. This pattern can be used to insure that the display and locators are working correctly. Once in local mode, it is possible to manually enter teletypewriter emulation mode, in which the terminal functions as an ordinary alphanumeric terminal, or teletypewriter. To do this the operator must enter a carriage return at the keyboard. This causes the test pattern to go down and an M to appear in the center of the screen. The operator should next enter a Y followed by carriage return. The characters G7 followed by an F will then appear at the top of the screen. This indicates that the terminal is in teletypewriter emulation mode, full duplex. Full duplex means that characters entered at the keyboard will not be echoed on the screen. Normally, the operator will want to run in half duplex. Pressing the function key labeled F1, at the upper left of the keyboard, will change the F to an H, indicating half duplex. Pressing the function key labeled F0 will reverse the effect of F1 and also cause a screen clear.

Once in teletypewriter emulation mode, the operator can log in to the operating system, attach or manipulate his files, and execute his graphics application (refer to appendix E).

TIGS will next prompt him to enter information about hardware availability. Once the information has been entered, the screen will go blank, indicating that the terminal is in system mode. Next, a border around the usable area of the screen will appear and then disappear. Two zeros will also appear in the upper left corner of the screen. These two digits are an error display used by the terminal-resident GSS4 software to indicate whether it has detected any illegal conditions. The operator has the option of whether or not to display these digits (refer to the Sanders Graphic Support Software manual for more details). Normally, the digits remain zeros throughout program execution. However, if an error does occur, there is usually no way to recover the error, and program execution must be aborted.

When program execution has been successfully completed, a G7 followed by an F appears at the top of the screen, indicating that the terminal has been returned to teletypewriter emulation mode. The operator can then proceed to execute more programs, or log off the system.

LOCATOR DEVICES

Determining the Mode-Set Locator

The current mode-set locator is determined by two conditions:

- The locator number specified in the latest call to SMLOCR.
- The locators which were said to be available at initialization time.

If, for instance, locator number 1 has been specified and only the trackball is available, the trackball becomes the mode-set locator. If, however, the lightpen is available and locator number 1 is specified, the lightpen becomes the mode-set locator, regardless of the availability of the trackball (refer to the section on Test Feature Routines).

Lightpen

When it is available, the lightpen is the default locator. It operates in two different modes, one for locating and one for picking.

When an application program calls the LOCATE routine, the postprocessor enables the lightpen for locating and requests lightpen interaction. It then waits for the operator to position the tip of the pen at the desired location on the screen and depress the tip by pushing the pen into the screen. The request for lightpen interaction is then terminated and the coordinates of the selected point are added to the location queue. Continued requests for lightpen locations are made until the function key labeled F15 is pressed to terminate locating.

When an application program calls the EVENT routine, the lightpen is enabled for picking and lightpen interaction is requested. The operator can then press a function key, or pick a displayed object. To pick an object the operator positions the pen tip at a location on the screen corresponding to a visible portion of the object, and depresses the pen tip. A slight increase in the intensity of a line segment or hardware character will be evident when the lightpen is pointing directly at it. This is a hardware feature which helps the operator determine where the lightpen is pointing. Graphic primitives which are drawn at low intensities might not be pickable with the lightpen. When a hit has been detected, the request for lightpen interaction is terminated. If the object picked was non-pickable (was a segment with an IGNORE action attribute), EVENT continues waiting until a valid selection is made.

NOTE

When the lightpen has last been used for locating, it remains in this mode. If the operator is then requested to make a lightpen pick, he must first terminate the locate mode condition. Pressing the lightpen tip against the screen will accomplish this.

Trackball/Forcestick

The trackball and forcestick are equivalent input devices from the viewpoint of the terminal and the application program. The Sanders postprocessor will support one or the other, but not both, of these devices at one time. Any comments made here concerning the trackball also apply to the forcestick. Whenever the LOCATE or EVENT routine is called and the trackball is the mode-set locator, a cursor will appear and a request will be made for either a trackball locate or a trackball pick. The cursor can then be positioned anywhere on the screen by rotating the trackball (or pushing on the forcestick).

After a call to the LOCATE routine, the program waits for the operator to position the center of the cursor at a desired screen location, and to press the function key labeled F0 to register the location. The cursor will then disappear and reappear. The operator can then enter another location, or terminate the set of input coordinates by pressing the function key labeled F15. Pressing any function key other than F0 or F15 will have no effect.

(F0) 16	(F1) 17	(F2) 18	(F3) 19	(F4) 20	(F5) 21	(F6) 22	(F7) 23	(F8) 24	(F9) 25	(F10) 26	(F11) 27	(F12) 28	(F13) 29	(F14) 30	(F15) 31

(M7) 7	(M8) 8	(M9) 9	(M15) 15
(M4) 4	(M5) 5	(M6) 6	(M14) 14
(M1) 1	(M2) 2	(M3) 3	(M13) 13
(M10) 10	(M0) 0	(M11) 11	(M12) 12

Figure G-3. Function Keys

After a call to the EVENT routine the program waits for the operator to press a function key, or to pick a displayed object. To pick an object, the operator places the center of the cursor over any visible part of the object and presses the function key labeled F0. If the object picked is non-pickable (is a segment with an IGNORE action attribute), the cursor reappears and EVENT continues waiting until a valid selection is made. Each valid selection is echoed on the screen by making the object momentarily disappear. A segment or function key with the TERMINATE action type terminates the input sequence. The function key labeled F0 is reserved for registering segment picks and cannot, therefore, be used as an ordinary function key.

FUNCTION KEYS

There are 32 function keys on the keyboard, numbered 0 through 31. Function keys 0 through 15 are in a cluster on the right side of the keyboard and are labeled M0 through M15. Function keys 16 through 31 are in a row along the top of the keyboard and are labeled F0 through F15 (refer to figure G-3). When the EVENT subroutine is called and KYON or KYAC has been previously called, the operator can press any of the keys, even though the interaction requested may be lightpen pick or trackball pick. Only if the mode-set locator is locator number 0, or if no locators are available, will function key input be specifically requested. Function keys 16 and 31 (F0 and F15) have special uses with regard to trackball interaction and lightpen locating.

NOTE

It is important that the operator always wait for an interaction request to appear in the user interaction request area (refer to Default Screen Layout) before attempting any interaction. The previous interaction request must disappear before a new one can appear.

HARDCOPY UNIT

A hardcopy of the image on the screen will be generated whenever the following three conditions are met.

- The hardcopy unit is turned on and connected to the terminal.
- It was indicated at initialization that the hardcopy unit was available.
- A call to the TIGS routine REMSCR has been made by the application program.

SANDERS GRAPHIC 7 HARDWARE AND FIRMWARE REQUIREMENTS

Minimum Configuration

The following list contains the Sanders products and options which are necessary for the utilization of TIGS with the Sanders postprocessor.

Description	Model Number
Terminal controller	5709
Refresh memory (8K, 16K, or 24K)	5702,5703,5704
Multiport serial interface	5713
Alphanumeric keyboard	5782
CRT display	0530

Optional Hardware

The following list contains optional Sanders hardware supported by the Sanders postprocessor for TIGS.

Description	Model Number
Photopen (lightpen)	5781
Trackball	5786
Forcestick	5787
Hardcopy unit	0570

Host to Terminal Loading Procedure

The Sanders postprocessor presently utilizes features not found in the standard terminal firmware. This necessitates loading from the host into the terminal the upgraded code. Each time the terminal is powered up the below described procedure must be followed.

I. After powering up the terminal, place it in the TTY or teletypewriter emulator mode (see the section entitled "Operating Procedure" for further details).

II. Login to the operating system.

III. Attach the loading program and its data file.

   A. On NOSBE

      1. ATTACH(WTSAND,ID=TIGS)
         ATTACH(TAPE5,ID=TIGS)

   B. On NOS

      1. ATTACH(WTSAND,TAPE5)

IV. Execute the program.

   A. On NOSBE

      1. enter: WTSAND (CR)

   B. On NOS

      1. enter: BAT (CR) (Enter the BATCH subsystem)

      2. enter: WTSAND (CR)

V. As the program begins execution, the user will observe a screen erasure. The code is now being loaded. The loading time is dependent on the baud rate used. At 1200 baud the user should expect a 2 1/2 minute load.

VI. When loading is complete, the terminal will return to the TTY emulator mode. This will occur when the "G7 F" appears at the top center of the screen. The terminal is now ready to run a TIGS program or allow the user to perform operating system commands.

Notes and Cautions

A TIGS program is run with the terminal in system mode. While in this mode the terminal will not display error messages issued by the operating system. The operator's only indication of program failure is that nothing new is displayed on the terminal screen. The operator must return the terminal to the teletypewriter (TTY) emulation mode (see the section entitled "operating procedure" for details) and then abort the program.

Prior to entering system mode, certain system messages such as loader error messages can be forced to appear at the terminal. This is accomplished by avoiding the use of "type ahead" in answering the TIGS initialization questions.

While running in system mode, the terminal will not display a FORTRAN PAUSE. If the program requires a pausing mechanism it is recommended to use a combination of calls to the PROMPT and KEYBRD routines.

The Sanders postprocessor maintains a host resident symbol table. This table is in addition to the preprocessor symbol table. The postprocessor table is used to maintain a representation of the terminal's memory layout. Each segment found in the preprocessor table is also found in the postprocessor table. However, a segment which is displayed in multiple windows will appear as one entry in the preprocessor table but as multiple entries (one for each window-segment combination) in the postprocessor table. If a program generates a large number of segments all shown through multiple windows, an overflow of the postprocessor table may occur. This condition is reported via TIGS error number 11001.

The user routine USRDAT (called by TIGS due to a segment containing user information) is only called if that segment is modified. Refer to the UDATA description in section 8.

INDEX

COMMENT SHEET

MANUAL TITLE: CDC Terminal-Independent Graphics System (TIGS) Version 1.1 Reference Manual

PUBLICATION NO.: 60455940 REVISION: C

NAME:________________________________

COMPANY:________________________________

STREET ADDRESS:________________________________

CITY:________________ STATE:________ ZIP CODE:________

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND STAPLE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
4201 North Lexington Ave.
St. Paul, Minnesota 55112

FOLD FOLD

TIGS SUBROUTINE SUMMARY

Subroutine	Page
TAINT(idseg,finten)	2-8
TAPICT(idseg,idpict)	3-5
TAPLIM(idseg,xll,yll,xur,yur)	3-6
TAPLI3(idseg,xllh,yllh,zllh,xury,yury,zury)	3-6
TAROT(idseg,deg)	2-8
TAROT3(idseg,xbase,ybase,zbase,xplane,yplane, zplane)	2-9
TASTYL(idseg,istyle)	2-9
TASYM(idseg,isym)	2-10
TAVIS(idseg,lvis)	2-10
TAXFA(idseg,bmat23)	5-8
TAXFA3(idseg,bmat34)	5-8
TAXFL(idseg,bmat23)	5-8
TAXFL3(idseg,bmat34)	5-8
TAXFR(idseg,bmat23)	5-8
TAXFR3(idseg,bmat34)	5-8
TA3D(13D)	3-7
TEXT(nchar,itext)	2-5
TEXT3(nchar,itext)	2-5
TFAC(lactn)	7-6
TFCSIZ(lcchar)	2-5
TFDSIZ(wide,high,widout,hiout)	2-6
TFERR(lroutn)	9-1
TFFONT(nfont)	2-7
TFHARD(lremot)	8-5
TFHILT(lhilt)	2-7
TFID(nid)	7-7
TFINT(ninten)	2-8
TFLOCR(maxloc,nlocrs,descrp,lone)	7-7
TFNSIZ(nsize)	2-11
TFPICT(npict)	3-5
TFPORT(nport)	4-7
TFROT(lninty,lcont)	2-8
TFSCRN(lrtang,xll,yll,xur,yur,resltn)	8-5
TFSTYL(lhard)	2-9
TFSVP(lsysvp)	4-8
TFSYM(nsym)	2-10
TFVIS(ltran)	2-10
TFXFA(lxlat,lscal,lrot)	5-7
TFXFA3(lxfm3,lpersp,lpyram)	5-7
TFXFL(lxlat,lscal,lrot)	5-7
TFXFL3(lxfm3,lpersp,lpyram)	5-7
TFXFR(lxlat,lscal,lrot)	5-7
TFXFR3(lxfm3,lpersp,lpyram)	5-7
TMAC(iactn)	7-6
TMCSIZ(widout,hiout)	2-5
TMDSIZ(widout,hiout)	2-6
TMERR(routin)	9-1
TMFONT(ifont)	2-7
TMHILT(lhilit)	2-7
TMID(idintr)	7-7
TMINFO(ninfo,info)	7-7
TMINT(finten)	2-8
TMLOCR(ilocr)	7-7
TMPICT(idpict)	3-5

Subroutine	Page
TMPLIM(xll,yll,xur,yur)	3-6
TMPLI3(xllh,yllh,zllh,xury,yury,zury)	3-6
TMPORT(idport)	4-7
TMROT(deg)	2-8
TMROT3(xbase,ybase,zbase,xplane,yplane,zplane)	2-9
TMSTYL(istyle)	2-9
TMSVP(lscren,xll,yll,xur,yur)	4-8
TMSYM(isym)	2-10
TMVIS(lvis)	2-10
TMXFA(bmat23)	5-8
TMXFA3(bmat34)	5-8
TMXFL(bmat23)	5-8
TMXFL3(bmat34)	5-8
TMXFR(bmat23)	5-8
TMXFR3(bmat34)	5-8
TM3D(l3d)	3-7
UDATA(nwords,idat)	8-4
UNISCR	8-4
VUPORT(idport,xll,yll,xur,yur)	4-3
VUPOR3(idport,xllh,yllh,zllh,xury,yury,zury)	4-3
WHERE(x,y)	8-5
WHERE3(x,y,z)	8-5
WINCLP(idwind,lclpnr,lclpfr,disner,disfar)	4-3
WINDIR(idwind,xeye,yeye,zeye,xat,yat,zat)	4-3
WINDOW(idwind,xll,yll,xur,yur)	4-3
WINPER(idwind,lpersp)	4-3
WINPLN(idwind,distat)	4-3
WINSIZ(idwind,width,height)	4-3
WINUP(idwind,dxup,dyup,dzup)	4-3
XIDNT(bmat23)	5-4
XIDNT3(bmat34)	5-5
XINVR(bmat23,binv23)	5-5
XINVR3(bmat34,binv34)	5-5
XROTA(deg,bmat23)	5-5
XROTA3(idaxis,deg,bmat34)	5-5
XROTL(ddeg,bmat23)	5-5
XROTL3(idaxis,ddeg,bmat34)	5-5
XROTR(ddeg,bmat23)	5-5
XROTR3(idaxis,ddeg,bmat34)	5-5
XSCLA(sx,sy,bmat23)	5-6
XSCLA3(sx,sy,sz,bmat34)	5-6
XSCLL(sdx,sdy,bmat23)	5-6
XSCLL3(sdx,sdy,sdz,bmat34)	5-6
XSCLR(sdx,sdy,bmat23)	5-6
XSCLR3(sdx,sdy,sdz,bmat34)	5-6
XTRNA(x,y,bmat23)	5-7
XTRNA3(x,y,z,bmat34)	5-7
XTRNL(dx,dy,bmat23)	5-7
XTRNL3(dx,dy,dz,bmat34)	5-7
XTRNR(dx,dy,bmat23)	5-7
XTRNR3(dx,dy,dz,bmat34)	5-7

102680404