USGS-OFR-90-615 USGS-OFR-90-615

UNITED STATES DEPARTMENT OF THE INTERIOR

GEOLOGICAL SURVEY

GEOTRANS:
An Interface Program from GEOPROGRAM to a Geographic Information System

Steve P. Schilling

Open-File Report
90-615

Prepared in cooperation with the
Nevada Operations Office
U.S. Department of Energy
(Interagency Agreement DE-AI08-78ET44802)

Denver, Colorado
1991

Copies of this Open-File Report
may be purchased from

Books and Open-File Reports Section
Branch of Distribution
U.S. Geological Survey
Box 25425 Federal Center
Denver, Colorado 80225

PREPAYMENT IS REQUIRED

Price information will be published
in the monthly listing
"New Publications of the Geological Survey"

FOR ADDITIONAL ORDERING INFORMATION

CALL: Commercial: (303) 236-7476
FTS: 776-7476

TABLE OF CONTENTS

ILLUSTRATIONS

ABSTRACT

The U.S. Geological Survey Plotter Lab, Denver, Colorado, has created a computer program to translate data from GEOPROGRAM recording files to a geographic information system (GIS) and a relational database. The program, GEOTRANS, takes files recorded on a Kern DSR 11 Analytical Plotter and translates the coordinate information into the KORK Geographic Information System (KGIS) and places the non-coordinate information into ORACLE, a relational database program. The advantage of linking the data collection capabilities of GEOPROGRAM with KGIS and ORACLE is to offer geologists a means of merging, editing, and querying coordinate and relational databases on-line.

GEOTRANS is written in Pascal v. 3.8 running under the DEC VMS operating system on a Microvax II computer. The program is structured in such a manner as to facilitate converting and restructuring of the program to translate 3D coordinate and attribute data collected with an analytical plotter to either 3D or other 2D GIS. This report describes how files from GEOPROGRAM are read, how 2D topology is created, and how GEOTRANS procedures and various calls to libraries transform and insert data to the proper place. A user's manual is included to run GEOTRANS.

INTRODUCTION

A photogrammetric mapping system that incorporates GEOPROGRAM software (Dueholm, 1989, Dueholm and Coe, 1989) with a KERN DSR 11 analytical plotter has been developed at the U.S. Geological Survey, Denver. The system is needed for the planned mapping of geologic features from stereo photographs that will be taken in exploratory shafts, in associated drifts, and in trenches excavated to study structural features as part of the Yucca Mountain Project at Yucca Mountain, Nevada (Interagency Agreement DE-AI08-78ET44802). In addition, this system will be available for photo interpretation, mapping, and data recording needs of other geologic projects.

Geologists use the GEOPROGRAM software and analytical plotter to collect three-dimensional (3D) coordinates of geologic features from stereo photographs such as, fault traces and strike and dip measurement locations. As geologists make these measurements, GEOPROGRAM automatically stores various parameters of geometric planes e.g., attitude measurements in a planes_record file, and real world X, Y, and Z coordinates, e.g., state plane coordinates, in a recording file (geofile). In addition, GEOPROGRAM also records how the data was collected by storing command information in the geofile. For example, commands indicate whether coordinates were collected point by point or incrementally, whether lines should be splined or not when plotted, or defines what symbol type should be plotted.

GEOPROGRAM is a data collection tool with limited editing capabilities. The KORK Geographic Information System (KGIS) (KORK, 1988) combined with the ORACLE relational database (RDB), is a tool with powerful database editing capabilities, with the capacity to build large cohesive databases from small data sets, with interactive on-screen database query, and with the ability to maintain two-dimensional (2D) topology when projecting data to the screen. The unique advantages offered by each of these tools led to the development of a reliable means to move data collected in GEOPROGRAM to KGIS. The translation program (GEOTRANS), written in VAX Pascal v. 3.8, running under VAX-VMS v. 5.0-2, translates and sends GEOPROGRAM coordinate data to KGIS for merging, editing, and plotting. GEOTRANS also translates a variety of non-coordinate attribute data (recorded in the planes_record file) to ORACLE RDB that may be queried in KGIS.

ACKNOWLEDGMENTS

The author would like to acknowledge the assistance of Dr. Keld S. Dueholm, Institute of Surveying and Photogrammetry, Technical University of Denmark, Lyngby, Denmark, Mr. Kevin J. Ingram, formerly of Kork Systems, Inc., Bangor, Maine, and Mr. Jeffrey A. Coe, U.S. Geological Survey, Denver, Colorado, in the development of GEOTRANS. Their comments and suggestions significantly improved the computer program and this report. The author would also like to acknowledge the comments and suggestions of Virginia M. Glanzman, U.S. Geological Survey, Denver, Colorado, that significantly improved the cohesiveness and clarity of this report.

GEOFILE COMMANDS

GEOTRANS reads, interprets, and acts on each command that has been recorded in a GEOPROGRAM geofile. These files begin with a START command and end with a QUIT command (see figure 1). The rest of the geofile consists of both 3D coordinate data and various commands that describe what features the 3D coordinates represent and the information needed to display them. In geofile, every command is preceded by the ASCII character sequence "CHANGE TO" that characterizes it as a command. The following paragraphs describe these commands and their purpose in GEOPROGRAM. These same commands take on a different meaning during the GEOTRANS translation to KGIS and ORACLE, and are described later in the section "GEOTRANS PROGRAM".

Change to Plot Area

Before recording data with GEOPROGRAM, the operator defines one or more plot areas (windows) and assigns a number (code) to each plot area. The 3D coordinates are projected onto the plotting media in two dimensions within the defined plot area. Each projection type available in GEOPROGRAM (orthographic, perspective, diametric, isometric, full periphery) requires a transformation to convert the 3D coordinates to 2D plot coordinates of the selected projection plane. A projection file created in GEOPROGRAM contains the parameters that define each 3D to 2D transformation.

```
START
CHANGE TO PLOT AREA    1
CHANGE TO CONT LINE OFF
CHANGE TO SMOOTH LINE OFF
CHANGE TO LINE TYPE    2
CHANGE TO SYMBOL TYPE     1 VALVES
CHANGE TO PLOT AREA     1
CHANGE TO LABEL rock bolts
CHANGE TO SYMBOL TYPE     3 rock_BOLTS
CHANGE TO SMOOTH LINE OFF
CHANGE TO OBJECT  0.00000 ROCKBOLT1
   948024.12   797464.99   207632.842
CHANGE TO OBJECT  0.00000 ROCKBOLT2
   950010.29   799102.62   207647.26
CHANGE TO OBJECT  0.00000 ROCKBOLT3
   950077.24   798886.50   212023.18
CHANGE TO SYMBOL TYPE   46
CHANGE TO SYMBOL TYPE   26 str./dip
CHANGE TO LABEL Slickensides
CHANGE TO ATTITUDE   5.41052  10.0/50
   631995.03   882828.86      6215.88
CHANGE TO OBJECT  DL    5.41052 Slickensides
CHANGE TO SYMBOL TYPE   31   plunge/arrow
CHANGE TO ATTITUDE   1.91986 60.0/62
   632001.85   882821.88         6218.83
CHANGE TO OBJECT  DV    6.76410  Fold Axis
CHANGE TO PLOT AREA    7
CHANGE TO CONT LINE ON
CHANGE TO SMOOTH LINE ON
CHANGE TO LINE TYPE     3
CHANGE TO SYMBOL TYPE      2
CHANGE TO PLOT AREA     7
CHANGE TO LABEL  frac1
CHANGE TO LINE TYPE     2 Fracture
   950230.51   799212.64   214143.91
CHANGE TO PEN DOWN
   950233.23   799186.60   214154.81
   950229.85   799136.76   214174.93
   950219.41   799124.29   214190.68
   949400.17   793758.78   216695.33
   949395.28   793716.54   216703.06
   949387.73   793670.92   216722.16
   949377.47   793606.57   216749.38
   949366.41   793559.55   216764.32
   949360.03   793540.64   216770.45
CHANGE TO PEN UP
CHANGE TO SYMBOL TYPE   1 target
CHANGE TO PLOT AREA     7
CHANGE TO LINE TYPE     2 Fracture
CHANGE TO LABEL frac2
   952646.34   799152.54   215437.10
```

Figure 1.--Example of a geofile.

```
CHANGE TO PEN DOWN
   952628.54   799156.51   215429.99
   952610.20   799154.59   215416.76
   952586.91   799173.86   215410.53
   952559.13   799188.02   215399.69
CHANGE TO PEN UP
CHANGE TO SYMBOL TYPE   1 target
CHANGE TO PLOT AREA     2
CHANGE TO LINE TYPE     2 Fracture
CHANGE TO LABEL frac3
   950355.44   800107.36   213069.13
CHANGE TO PEN DOWN
   950362.46   800112.58   213049.96
   950372.67   800131.07   212972.04
   950383.54   800146.45   212904.51
   950396.27   800159.56   212871.14
   950414.74   800179.92   212823.46
   950445.07   800211.28   212733.98
   950451.63   800223.25   212682.37
   950453.30   800231.82   212664.40
   950450.56   800245.95   212645.18
   950464.10   800262.40   212575.17
   950475.86   800273.17   212527.10
   950505.48   800303.22   212419.90
   950511.09   800319.08   212390.87
   950524.70   800344.64   212302.69
   950524.00   800353.88   212283.96
CHANGE TO PEN UP
QUIT
```

Figure 1.--Continued.

Change to Label

The operator may assign an alphanumeric label (geolabel) of up to 40 characters to each collected point or line. Each recorded object may be given a unique geolabel. For example, five fracture traces may be given consecutive geolabels, frac1 through frac5. Alternatively, a single geolabel may describe many subsequent data entries and in fact, will continue to do so until the operator changes the geolabel.

Change to Line Type

The line type is used to classify lines. The line type describes what feature a line represents and(or) how the lines are to be plotted. Similar linear features, e.g., fault traces, are appointed the same line type when collected in GEOPROGRAM. The GEOPROGRAM user assigns each line type an integer value (code) and, if desired, an alphanumeric label. For example, all fracture traces might be given the line type "1 FRACTURE" (here the code 1 is followed by the alphanumeric label FRACTURE), while faults may be given line type "3" (here only a code is used). The code is recorded and is associated with the 3D coordinates of the succeeding feature in the geofile.

Change to Cont <On> <Off>

CONT (a shortened form of continuous) ON describes if the operator used continuous point collection mode (points collected automatically at defined increments) while recording a line. CONT OFF describes point by point (points manually selected) collection mode. CONT ON does not specify which point rate criteria (distance/angle or time) was used when continuously collecting coordinates.

Change to Smooth <On> <Off>

SMOOTH ON indicates that a line collected either by continuous or point by point mode should be smoothed using a spline or similar function as it is sent to an output device. SMOOTH OFF indicates a collected line should not be smoothed or splined went sent to an output device.

Change to Pen Number

A pen number is defined in the line type file for automatic pen selection during on-line plotting when working in GEOPROGRAM. If the user enters 0 in the line type file, he can interactively select the pen number while plotting.

Change to Symbol Type

The symbol type defines what symbol is associated with the succeeding 3D coordinates and(or) how those symbols should be plotted. The GEOPROGRAM operator assigns each symbol, whether a physical object location or measurement location, an integer value (code) and, if desired, an alphanumeric label. For example, a surveying target may be recorded as symbol type "34 TARGET" (here 34 is the code with a label of TARGET) while the location of a plunge and trend measurement may be designated as "27 SLICKENSIDE".

Change to Attitude

In the geofile, the attitude command contains a real number value that defines the angle of rotation in radians for a particular symbol. This value is succeeded by an ASCII label of the measured values of strike and dip or plunge and trend.

Change to Object

Object denotes plotting of the current symbol. As part of defining the symbol in GEOPROGRAM, the operator may select annotation to plot beside the symbol. For attitudinal symbols the operator selects a value for the dip and strike (D), plunge and trend (P), or one or more coordinate values (X), (Y), or (Z) for control points. In addition, the operator may also select annotation that will plot next to the attitudinal symbol. Annotation may be fixed (F), variable (V), or the current geologic attitude (geolabel L). The object command records this one or two ASCII character code. In the code DL for example, the D indicates a dip and strike symbol should be plotted using the dip and strike information recorded in the most recent ATTITUDE command, while the L indicates that the current geolabel be plotted beside the symbol. For the code DV, the D indicates that a dip and strike symbol should be plotted and V indicates that the varying label (a separate label is entered for each symbol by the GEOPROGRAM operator) should be plotted next to it. Following the one or two letter code is a real number value in radians dictating rotation of the symbol, succeeded by the alphanumeric annotation. In the case of attitude symbols, the symbol rotation is redundant to that given in the attitude command.

Change to Pen Down

Pen down begins a line at the most recent coordinate read. The command tells an on-line screen or plotter device to put the "pen" down at the first coordinate location and to wait for the next point to draw to or for the next PEN UP command.

Change to Pen Up

Pen up indicates that the user is finished collecting the coordinates of a line. This command tells the device to raise the "pen".

KORK GEOGRAPHIC INFORMATION SYSTEM

KGIS is a hybrid data model that stores attribute data in a relational database and coordinate data in an object-oriented database management system (Ingram and Phillips, 1987). Complex data structures are hidden in the lower levels of the system that provides a powerful high-level view to the user where both coordinate and attribute may be queried at the same time. The topologic data structure allows relations of features between layers as easy as within a layer. Tessellated data structure in the coordinate database provides easy access to any subset of the data base (Ingram and Phillips, 1987). During the GEOTRANS translation, KGIS library procedures, available from KORK (KORK Systems, 1988), are called to create nodes, edges, faces, topology, themes, and geographic features that are stored for later display, editing, or plotting.

EXPLANATION OF TOPOLOGY

Managing and maintaining large spatial databases have plagued the design of geographic information systems. This problem has been addressed through the development of topologically structured databases. Herring (1987) describes topology as "coordinate free geometry", i.e., those relationships that are maintained between objects regardless of the coordinate system. He lists curve, connected, adjacent, bounded, inside, outside, boundary, and orientation as examples of these topologic relationships. Imagine putting a Boeing 747 jet with the nose facing north on a magic platform. The platform can enlarge or reduce whatever is on it and, magically, the jet begins to shrink. Although the size of the jet changes, the nose of the jet continues to face north, the seats are still inside, and the curves of the jet engines are still curved. Removing the coordinate framework from spatial data has given rise to faster and more efficient algorithms to process and manipulate topologically related objects.

Herring (1987) draws analogy between the topologic structure of a map to a jigsaw puzzle. The puzzle pieces are analogous to faces while the lines between pieces are analogous to nodes and edges. A topologically structured "puzzle" is aware of relationships to adjacent pieces and would be able to assemble itself.

In KGIS, 2D topologic information is created as edges, nodes, and faces are built from digitized points. These features are in turn grouped into themes and geographic features that are described later.

KGIS TOPOLOGIC DEFINITIONS

Terminology or definitions used for topology, data structures, and objects may vary in specifics from one GIS to another. Therefore, the following sections provide an overview of specific terms defined and used by KGIS (fig. 2).

Topologic Features

Topologic features are nodes, edges, and faces that are built from raw digitized point, line, and polygon coordinates. Both a single, isolated digitized point and the coordinate location where two or more lines intersect define a node location. Line coordinates (a stream of X,Y coordinates) form an edge until that line crosses another line (in 2D). At the intersection of any two lines, the coordinates leading up to the intersection become an edge, a node is created at the intersection point, and a second edge is created from the remaining coordinates of the digitized line. Faces are unbroken areas, defined by a series of edges and nodes that surround an area. KGIS keeps track of each object and its relationship with all surrounding objects in 2D space monitoring which faces, edges, and nodes give rise to cartographic features.

As an example of the process of building topology, figure 3 shows a hypothetical map. Here, Highway 80 was digitized from the southwest to the northeast (a single line shown as the segments labeled with boxes 1, 3, 9, 32, and 111). As topology was created, the single digitized line was split up into edges 1, 3, 9, 32, and 111; a new edge being created wherever a line intersects or crosses Highway 80. At each such intersection, a node is created (indicated by black dots labeled with circles 15, 26, 137, and 14). The nodes and edges listed above now make up a cartographic feature (explained in next section) called Highway 80.

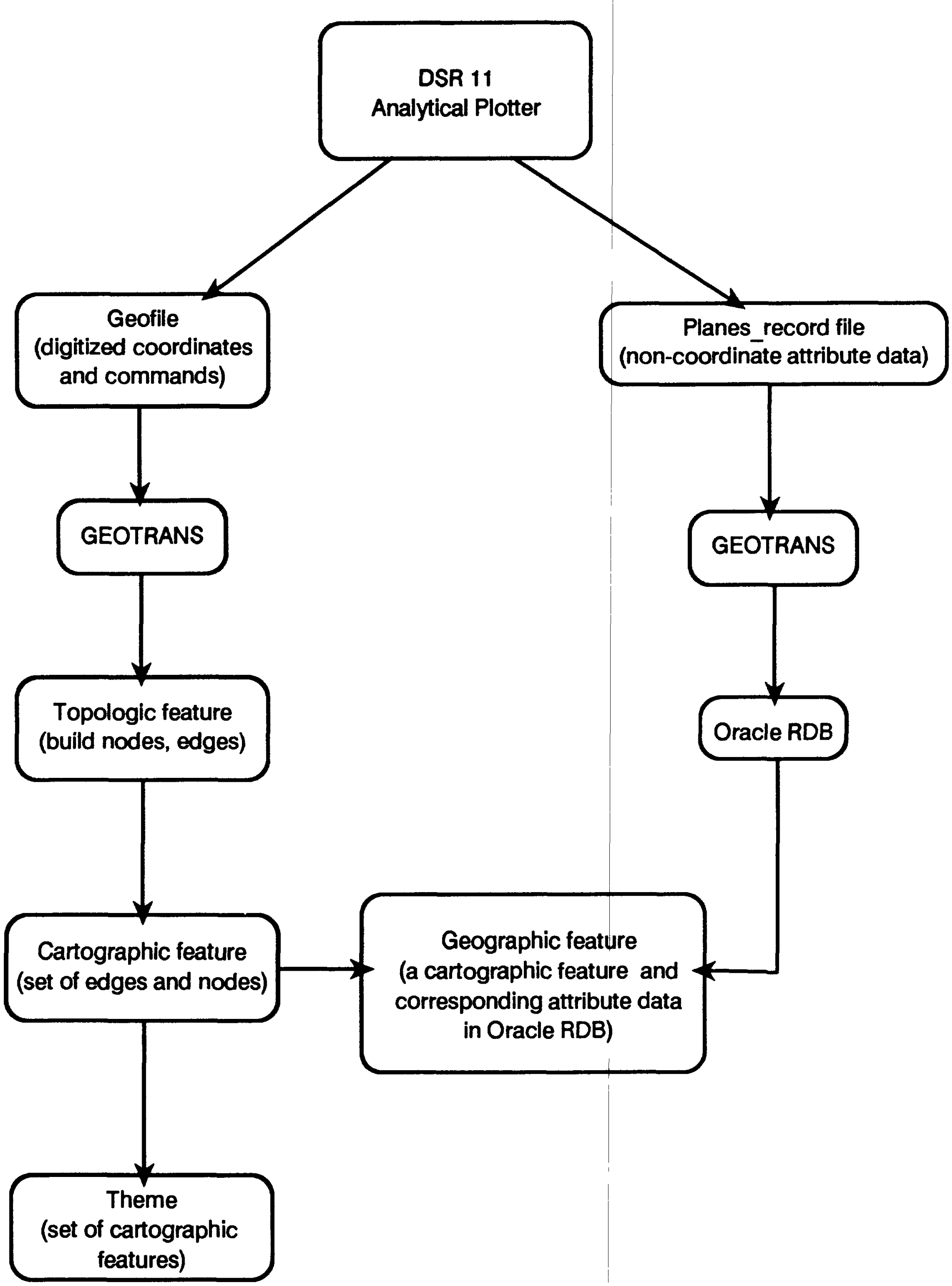

Figure 2. -- KGIS Topologic Terminology.

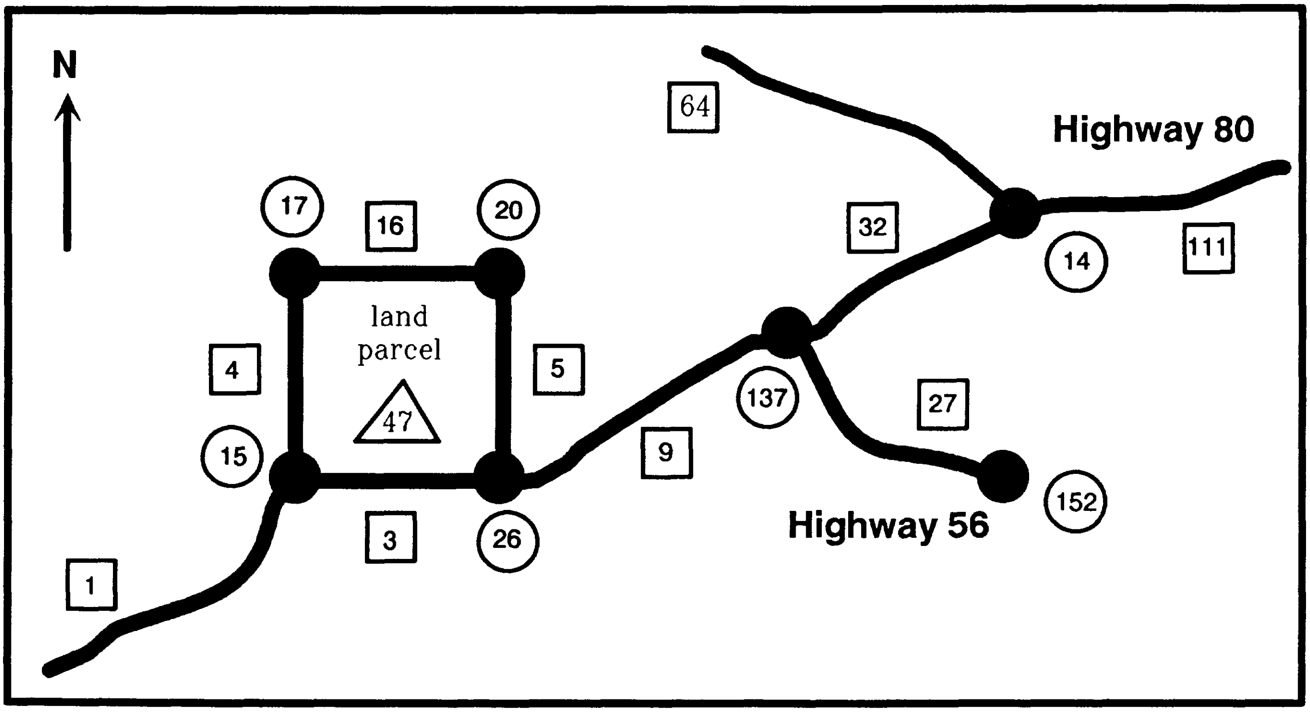

Figure 3. -- Hypothetical map with topologic building blocks.

A single edge, node, or face may play many roles by being a part of several different cartographic features. For example, the edge labeled with box 3 is a segment of Highway 80 and also a part of a face: land parcel 47 (identified by a triangle in fig. 3). By building topology, the GIS can keep track of the coordinate "pieces" an which cartographic feature they help to make up. This bookkeeping of nodes, edges, and faces allows the GIS to keep track of relationships (topology) in 2D space between one cartographic feature and any other cartographic features.

Cartographic Features

Cartographic features are lines, points, or polygons (unfortunately the raw digitized coordinates or streams of coordinates are also often termed lines, points, and polygons; the cartographic feature, in addition, contains topologic information) built from nodes, edges, and faces. A linear cartographic feature is created from a series of edges and nodes. Two lines may share some edges, and at the same time contain some edges they do not share. This relationship of one cartographic feature being made up by many edges and nodes is often termed a one to many relationship.

Themes

A theme is a set of cartographic features that are similar in some regard. For example, all of the trout streams for an area that are open all year to fishermen might make up one theme, improved roads that contain greater than 30 percent gravel might make up a second, and unimproved roads might make up a third theme. The relationship here is many to one: many individual streams make up one theme.

Geographic Features

A geographic feature is the combination of the cartographic feature described above and the supplementary non-coordinate data residing in ORACLE. The coordinate and non-coordinate data are joined by means of a spatial "key" (KORK Systems, 1988). The relationship here is one to one: one geographic feature to one cartographic feature and its associated non-coordinate data.

3D Topology

In order for geologists to be able to study 3D geologic relationships, 3D topology must carry through from a 3D data collection tool such as GEOPROGRAM to a GIS database to create a solid earth model. The use of such a model, made possible with 3D topology, would allow geologists to study and model true geologic relationships. They could investigate the intersections of fracture planes at various depths, model fault curvature, and even look at a cross section several feet "into" a drift wall to study and model the impact of fractures on fluid flow. A true 3D GIS must be able to manage and use the third dimension (z value) dynamically, as it does the x and y values. Unfortunately, most GIS databases are based on 2D rather than 3D topology. Attempts at adding the third dimension to the topology have resulted in storing the third dimension as an attribute along with other supplementary data. Although some GIS software are capable of using elevation data to create a surface for a perspective view, these systems fall short of a true 3D GIS and 3D topology.

Rather than just an attribute, the z coordinate must be placed at the same level, with all the functions of the x and y coordinates in the creation of a solid earth model. Such a model would require modifications in the creation of topology and topologic operations that are not currently addressed in 2D GIS programs. The geologic 3D GIS must be able to store and access the 3D structure of geologic elements both quickly and efficiently.

PROCEDURAL CALLS TO THE KGIS LIBRARY

The KGIS procedures used by GEOTRANS are called from a PASCAL procedure library. Below, the procedures are grouped together in roughly the order used in GEOTRANS, beginning with calls to a group of initialization procedures. The second set of calls translates the coordinate data and builds topology; the third set builds themes, and the last set of calls close and cleanup the ties needed for translation between GEOPROGRAM and KGIS.

Initialization Procedures

The first call to the KGIS library is **db_opened**. This procedure opens the coordinate database created by the user outside of GEOTRANS with the KGISINIT program (KORK Systems, 1988). After the database is opened successfully, two procedures are used to set up and scale the coordinate database: 1) **Map_extents** reads the map coordinate range that was entered during the initialization of the database, and 2) **world_to_range** converts the real world coordinates defining the extents of the map to internal "KORK space" coordinates (a scaling and translation function). Two procedures, **display_init** and **top_window_display**, initialize and set up the Tektronix graphics terminal, respectively. The graphics screen is scaled to the current database range and communication is established for drawing to the screen. As a final step in the initialization, the KGIS **coll_parm** (data collection parameters) record is assigned values, and defined for the desired precision to automatically trim and extend lines. The collection parameters consist of values that determine when points or lines will automatically snap to other points or lines, extend or peel back a line, or trim a line.

Translation Procedures

The translation procedures read coordinate values and act on commands recorded in a GEOPROGRAM geofile. A **world3D_to_loc** procedure transforms real world coordinates into internal "KORK space" coordinates. After transformation, the topology is constructed, and lines and points are placed into the correct theme.

GEOTRANS calls two KORK library procedures as it begins to build topology for any point or line. **Init_new_cart** gets a new cartographic element (a data type), defines its record components to be a line or point, and assigns an individual identification code to that element. **Init_id_list** initializes the identification (id) list for tracking cartographic elements. As mentioned above, the coordinates are collected, projected onto the desired 2D plane, and transformed into "KORK space" coordinates.

While translating single point coordinates (i.e., symbols), library procedures are called on to build the required topology, in this case isolated nodes. **Create_isol_node** creates a topologic element (a node) and **add_id_to_end** adds the identification code to the end of the list that keeps track of the nodes.

When translating streams of coordinates (i.e., lines) with the GEOTRANS program, the topologic element used is an edge rather than a node. The cartographic element and identification list are initialized as described above for points, however, because lines contain many sets of coordinates rather than just a single set, a unique procedure, **collect_string** is called.

The **collect_string** procedure uses calls to other procedures as part of its passed parameters. In this way, **collect_string** calls a GEOTRANS procedure **get_point**, that gets the next point, and **process_edge** that calls **add_id_to_end** to keep track of edges and nodes that make up the line. In addition, **collect_string** needs **coll_parm**, the collection parameter record for snapping and clipping lines, and **start_loc**, the first set of coordinates of the line. **Collect_string** returns the **end_loc**, the coordinates for the ending point of the line.

The **get_point** procedure processes both points and lines. To differentiate between the two, a flag is set that governs whether the procedure processes a line or a point. When the program encounters a command PEN DOWN or CHANGE TO OBJECT, the flag will be turned on. While collecting the points for a line, the flag is turned off to re-route processing for string collection.

Themes

Each line or symbol is assigned an integer code in the GEOPROGRAM and this code is used to define the theme name. The library procedure **them_found** checks if the theme already exists, or if the theme should be added to the list of valid themes. The first time a theme name is used in a geofile requires the help of the **init_new_theme** and **store_theme** procedures from the library. These procedures store a new theme by comparing the theme name with a name in a look-up table (an array containing a list of GEOPROGRAM line or symbol codes and corresponding alphanumeric theme names fig. 4).

A library function, **legend_line** returns the line in a legend file (created outside KGIS by the user) corresponding to the theme name. The legend file contains parameters that govern how each theme will be displayed on the Tektronix graphics terminal (e.g., color, brightness, order of importance of themes to govern which themes overlap other themes on the screen display, etc.). If no entry occurs in the legend file, a set of default values are used (KORK Systems, 1988).

Currently, if the object being translated is a point (symbol), procedures are called to position the point at the correct location on the Tectronix screen and draw a fixed diameter circle at that location (circles are used to represent any symbol during the translation). If the object is a line, a line is drawn at appropriate locations on the screen.

Example of a line type table file

```
1	fracture
2	fault
3	contact
4	joint
5	anomaly
	.
	.
	.
```

Example of a symbol type table file

```
1	survey-target
2	cavity
3	strike/dip
4	plunge/trend
5	fold-axis
6	lithophysae
	.
	.
	.
```

Figure 4.--Examples of line and symbol type tables.

Completion

Finally, a set of four procedures will either build the topology and submit the translation of the coordinates into the KGIS coordinate database or, if there is an error, will abort the creation of the topology. **Trans_abort** is a procedure that aborts a transaction if errors occur, otherwise, **build_line** or **build_point** is called. The **build_line** procedure constructs a line from coordinate data, a cartographic element defined as a line, and a list of directed edges and nodes. The completed line with topologic relationships is placed into the database. **Build_point** actually produces point topology from a node by combining coordinate data with a cartographic element and the list of nodes, and placing it in the theme. **Trans_end** completes the transaction and commits the new topology to the database, while **free_id_list** frees the node id list. After the translation process is complete, the procedure **db_closed** closes the database.

SENDING INFORMATION TO ORACLE RDB

Non-coordinate data is sent to the ORACLE relational database. Procedures from the KGIS library that address ORACLE are called to initialize, to log on, and to submit this information. GEOTRANS initializes communications with ORACLE using the **rdbInit_Comm** procedure followed by the **rdbLogon** to log on to the relational database (rdb).

Outside of GEOTRANS, the user names and sets up a table in ORACLE that will receive the non-coordinate information from GEOPROGRAM files (see fig. 5 and "User's Manual" in this report). The user communicates with ORACLE, via structured command language (SQL). By entering SQL commands, the user creates a table made up of columns, each with a specific heading. GEOTRANS will place each piece of information needed from GEOPROGRAM files into the correct column under the correct heading in ORACLE. A library procedure, **sql_command**, holds a table name and all of the column headings that are to receive information from the translation (the headings must be in the same order they exist in the table). **Rdb_Submit_SQL** submits this command to the rdb, thereby setting up communication paths (links) to the table and preparing the table to receive the data.

Currently, GEOTRANS sends information contained in a **planes_record** file (a file that contains parameters for each measured plane orientation recorded in GEOPROGRAM) to ORACLE. The file is opened and values are bound into ORACLE as parameters. **RdbBind_strval**, **rdbBind_intval**, and **rdbBind_dbleval** bind strings, integers, and double precision real values, respectively, to the ORACLE database. **RdbExec_SQL** begins execution of the SQL command, and **rdbCommit** commits the transaction to the database. After the data is sent, **rdbRelease_SQL** closes the SQL statement cursor and **rdbLogoff** logs off ORACLE. Finally, **rdbAlldone** ends all communications with the ORACLE relational database.

Column name	Width	Data Type
sname	20	CHAR
skind	20	CHAR
inumpnt	10	INTEGER
dne1	15	REAL
dne2	15	REAL
dne3	15	REAL
dne4	15	REAL
dne5	15	REAL
dne6	15	REAL
dne7	15	REAL
dne8	15	REAL
dne9	15	REAL
dnv1	15	REAL
dnv2	15	REAL
dnv3	15	REAL
dqv1	15	REAL
dqv2	15	REAL
dqv3	15	REAL
dcv1	15	REAL
dcv2	15	REAL
dcv3	15	REAL
dstrike	15	REAL
ddip	15	REAL
ddir	15	REAL
dms	15	REAL
dmd	15	REAL
dmp	15	REAL

Figure 5.--Format of the columns to set up an ORACLE table.

GEOTRANS PROGRAM

To complete a translation, the GEOTRANS program requires access to these files: 1) the KGIS database, 2) the legend file, 3) the geofile, 4) the symbol types file, and 5) the line types file. A description of each file follows.

KGIS Database

Prior to translation, the user must run the KGISINIT program to initialize a KGIS coordinate database. KGISINIT will create two files, a <database filename>.DAC file and a <database filename>.DAB file. These files will receive and store coordinate information from a geofile.

Legend File

The construction of the legend file is done prior to running the translation. The legend file assigns graphics attributes for drawing various themes to the screen.

Geofile

As described above, the geofile is a file of GEOPROGRAM coordinates and commands.

Line Type File

The line type file is an ASCII file that contains line codes assigned in GEOPROGRAM and corresponding alphanumeric theme names that will be used in KGIS. The file is read by GEOTRANS and read into an array for looking up and assigning theme names while completing a translation.

Symbol Type File

This file is identical in function to the line type file described above, however, rather than looking up line themes, the file searches a different table for symbol themes.

Proceeding with the Translation

After checking for the existence of these files and completing the initialization procedures described above, GEOTRANS proceeds to read the geofile. If the program encounters a command, it is carried out and the program reads the next line. If it encounters a coordinate, the coordinate is read into a buffer. If the next line read is a CHANGE TO OBJECT command, then the coordinate in the buffer represents a symbol, and GEOTRANS will create the node and its topology. If the next line contains a command that indicates that the coordinate in the buffer is the first point of a line (CHANGE TO PEN DOWN), the procedure **get_point** is called, processing the rest of the coordinates making up the line. If two or more coordinate sets are encountered before the CHANGE TO PEN DOWN command, those points successively enter the buffer eliminating the previous point(s). A flag is used to govern this process of delegating which procedure is given control of processing.

Each coordinate, as well as alphanumeric labels and real number values that accompany commands are simply read into buffers. Each command must be interpreted to carry out the appropriate action on the information contained in these buffers. The GEOTRANS program takes the following actions on geofile commands.

Change to Plot Area

The first plot area code is read from the geofile as part of the initialization process. GEOTRANS uses the code to open a projection parameter file that has been created with the GEOPROGRAM and reads the record corresponding to the plot area code. This record contains the plot area parameters that are used in subsequent coordinate transformations. Succeeding commands to change the plot area initiate a comparison between the current plot area code and the new code. If they are the same, no action is taken. If the codes are different, the translation is stopped, and the user must enter whether to change plot areas or to keep the original plot area. If the user changes the plot area, a new window is created, the projection parameter file is read for new record information, and subsequent coordinates are processed for the new plot area. If the user wants to keep the original plot area, no action is taken and the translation proceeds.

Change to Label

The geolabel is read into the label buffer and will remain in the buffer until changed by another geolabel command.

Change to Line Type

The line type code is read and compared to the line type look-up table for a match. The geofile text is also compared with the themes in the table, reporting to the user whether there is a match or mismatch. Regardless, the alphanumeric text from the look-up table is used to place the line into the theme indicated by the line type code in the geofile and the line type code is placed into a buffer.

Cont <On> <Off>

In the current version, information as to whether the CONT command is ON or OFF is held in a buffer but no action is taken in the translation.

Smooth <On> <Off>

In the current version, information as to whether the SLICK command is ON or OFF is held in a buffer but no action is taken in the translation.

Change to Pen Number

In the current version, this command is ignored by GEOTRANS.

Change to Symbol Type

The symbol type code is read and the symbol type look-up table is searched for a match. It compares the geofile text with the symbol table alphanumeric entries, reporting whether there is a match or mismatch. In either case, the geofile symbol type code will be used to attach the symbol to the correct theme and the symbol type code is placed into a buffer.

Change to Attitude

The rotation buffer is filled with the rotation value of an attitude symbol that, in the case of the strike and dip symbol reflects the strike. The strike and dip buffer receives the ASCII strike and dip information. The information in these buffers is used later with the CHANGE TO OBJECT command.

Change to Object

This command sets a flag to indicate that the coordinates held in the buffer make up a point (symbol), retrieve the theme name from the theme name buffer, and call the procedure to initialize and store the theme if necessary. This command will then plot the symbol type from the buffer and read the symbol rotation value and the object label into their respective buffers. When used in concert with the Change to Attitude command, the strike and dip or plunge and trend with the buffered attitudes and/or geolabels are plotted. Finally, a call is made to the **GN_Get_Node** procedure that will build the required topology.

Change to Pen Down

This command sets a flag to indicate that the coordinates that will follow in the geofile make up a line, retrieve the theme name from the theme name buffer, and call the procedure to initialize and store the theme if necessary. The first coordinate will have been read into a coordinate buffer. The **GS_Get_String** procedure is called. This procedure summons the needed KGIS procedures for building topology and sets a flag to true that will get the first point from the coordinate buffer. In the **get_point** procedure, if the first point flag is set to true, the coordinate buffer is read, the translation of 3D coordinates into a 2D projection takes place using a GEOPROGRAM transformation procedure, and, as mentioned above, a KGIS procedure transforms the coordinates to "KORK space". Because the object being translated is a line, there must be a movement of the cursor to the first point location on the Tektronix screen to begin the drawing of the line. After this move, the first point flag is turned off.

The remainder of the **get_string** procedure consists of calling a KGIS procedure, **collect_string**. As mentioned above, the key to the successful collection of the remaining coordinates that make up the line is the first point flag; a flag that allows collection of successive points when it is turned off. Successive line coordinates are read, a transformation performed to the plane of projection, topology created, and the line drawn on the screen.

Warning: If the operator inadvertently deletes this command while editing the geofile, the coordinates will successively enter and exit the coordinate buffer, but no line will be translated to the database.

Change to Pen Up

This command is used to tell GEOTRANS that it has received all the coordinates for the current line. GEOTRANS will also stop processing a line if it encounters any other command while in **get_point**.

Communicating Progress to the User

The GEOTRANS program opens all of the required files needed to complete the translation. GEOTRANS reads each command or coordinate consecutively from the geofile and executes the specified request. It is robust enough to continue even though an operator may have edited a geofile and deleted some commands. Initially, GEOTRANS accesses the geofile to acquire maximum and minimum x and y coordinates and the first addressed plot area. GEOTRANS writes messages of what it has found in the geofile to the screen (to indicate its progress) and to a <geofile name>.SWF file on disk (for a hard copy record of translation) as the translation proceeds. The GEOTRANS program is structured so there is a modular interface to the KORK library and should be relatively easy to interface with other 2D or 3D GIS systems.

GRAPHICS SCREEN PROCEDURES

While translating geofiles, calls are made to the KGIS library to address the Tektronix graphics terminal. These procedures will plot lines and symbols on the screen as they are translated, and set the coordinates for an information location (infoloc), where information blocks will be displayed on the screen with data from ORACLE. Each GEOTRANS procedure that addresses the graphics screen includes calls to **AGIsymBlk_start** and **AGIsymBlk_end**, KGIS library procedures that use records containing parameters for addressing the graphics screen. The following is a description of GEOTRANS procedures that in turn call KGIS library routines:

Move_node_to_location: Calls the KGIS library procedure **set_cart_infoloc** that defines the screen coordinates for the location for information block for each object. **AGImove** moves the screen cursor to the current coordinates to draw a circle.

Move_to_start_of_line: Calls **set_cart_infoLoc**, as described above. **AGImove** moves the screen cursor to the current coordinates to draw a line.

Draw_point_to_screen: Uses the **AGIdraw_circle** to draw a circle at the current location and with the given diameter.

Draw_line_to_screen: Uses the **AGIdraw** procedure to draw line segments to the graphics screen.

CALLS TO THE GEOPROGRAM LIBRARY

GPCGTT_Geo_Program_Compute_Ground_To_Trans_Transformation is a procedure that calls supplementary GEOPROGRAM procedures to compute matrices and parameters for 3D to 2D coordinate transformations (fig. 6).

CGTTM_Compute_Ground_To_Trans_Matrix computes a homogeneous transformation matrix from ground coordinates to a transformation plane.

Subsequent calls are made to one of three procedures to compute transformation parameters if the transformation is a full periphery projection.

Compute_Drift1_Transformation_Parameters computes **Drift_Parameters** for a normal drift periphery projection. **Compute_Drift2_Transformation_Parameters** computes **Drift_Parameters** for radial drift periphery projections.

Compute_Shaft_Transformation_Parameters computes **Shaft_Parameters** for the shaft periphery projection. These specialized projections are used for underground mapping projects (Dueholm and Coe, 1989).

Prior to transforming 3D coordinates to KORK space coordinates, a GEOTRANS procedure **OTT_Object_To_Table** uses the homogeneous transformation matrix to transform 3D coordinates to the selected 2D transformation plane in a plane similarity transformation. For full periphery transformations the matrices are modified from the tunnel parameters and one of the following: (1) a **Ground_To_Drift1_Periphery** procedure that transforms ground coordinates to periphery coordinates using a normal drift periphery projection, (2) a **Ground_To_Drift2_Periphery** procedure that transforms ground coordinates to periphery coordinates using a radial drift periphery projection, or (3) a **Ground_To_Shaft_Periphery** procedure that transforms ground coordinates to periphery coordinates using a shaft periphery projection.

Geofiles are made up of 512 byte records that do not necessarily coincide with single command lines or coordinate sets entered by the user. Two procedures are used to move backward and forward through these records one line at a time. **FIOF_Forward_In_Output_File** reads forward through the file, while **BIOF_Backward_In_Output_File** moves backwards through the geofile records.

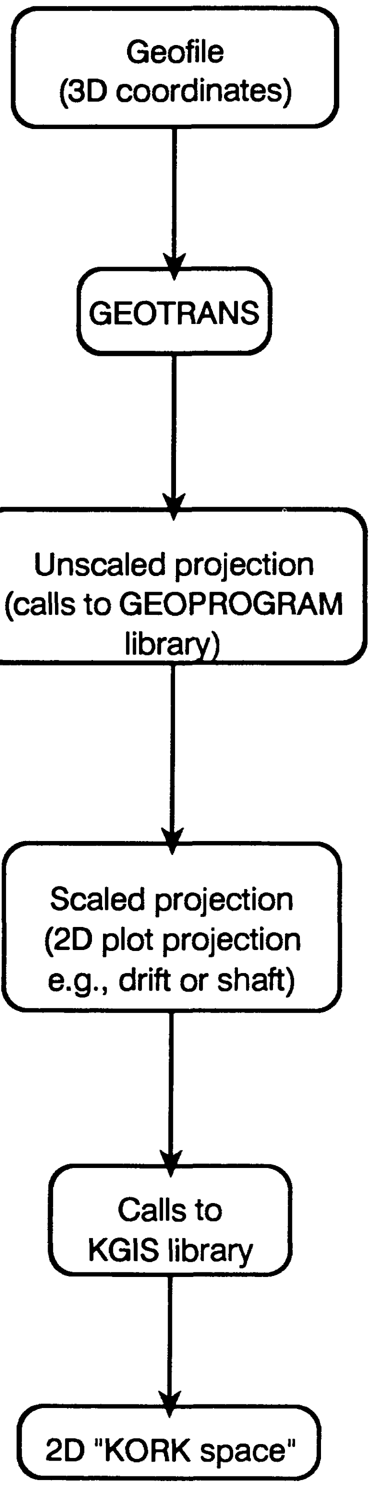

Figure 6.--Flow diagram of coordinate transformations during translation.

CALLS TO THE VT100 LIBRARY

GEOTRANS can be run from DEC VT100, VT200, or VT300 series terminals. The VT100 Library (KERN and CO., Ltd.) provides procedures to address these terminals. GEOTRANS uses the following procedures from this library:

CURPOS_Cursor_Position - Positions the cursor on the screen.
DBWIDT_Double_Width - Prints double width characters to the screen.
SELGRA_Select_Graphics - Select a variety of screen graphics operations such as blinking characters.
SELSGR_Select_Special_Graphics - Prints special characters to the screen.
SELASC_Select_ASCII - Prints ASCII characters to the screen.
ERASCR_Erase_Screen - Clears the screen.
BELLRI_Bell_Ring - Rings the Bell.
IC_Invisible_Cursor - Makes cursor invisible.
VC_Visible_Cursor - Restores cursor to the screen.

USER'S MANUAL

Before running the GEOTRANS program, the user will need to make sure the necessary files are accessible to GEOTRANS. For translation of coordinate data, as mentioned above, GEOTRANS must have access to: 1) the KGIS database, 2) the KGIS legend file, 3) the GEOPROGRAM geofile, 4) the GEOTRANS symbol type file, 5) the GEOTRANS line type file, and 6) the GEOPROGRAM plot area file (a file created in GEOPROGRAM that contains plot area parameters such as index points, angles for perspective views, and so on). For translation of non-coordinate data into ORACLE, GEOTRANS must have access to: a user created ORACLE table with the correct columns (specified below), ORACLE, and the planes_record file. In addition, the user must have a valid username and password to enter ORACLE.

FILES NEEDED FOR DATA TRANSLATION TO KGIS

KGIS provides three programs the user must use for translating and using data from GEOPROGRAM. KGISINIT is a program that initializes an empty coordinate database, KGISEDIT will allow the user to edit data in a database, and KGISSQL is a program to query and view coordinate and non-coordinate data. The user should consult the KGIS User's Manual for detailed information concerning these programs (KORK Systems, 1988).

Before running the GEOTRANS program, an empty KGIS database is created with the KGISINIT program. This program creates a <filename>.DAC file and a <filename>.DAB file. The legend file is created outside of GEOTRANS according to instructions in the KGIS manual. The geofile is created while running the GEOPROGRAM with the filename and extension being selected by the user. The symbol type file and the line type file can be created with a text editor (see fig. 5). These files will contain simply line or symbol codes used in GEOPROGRAM and corresponding text that designate theme names for KGIS. The plot projection file is created while running GEOPROGRAM with the filename and extension selected by the user.

FILES NEEDED FOR DATA TRANSLATION TO ORACLE

An ORACLE table must be created by running an ORACLE option, i.e., EasySQL or SQLPlus. At present, 27 columns must be entered in the order specified and with the parameters given in figure 5. ORACLE must be running and available to the station where the user is working. The planes_record file is created while running the GEOPROGRAM with the filename and extension selected by the user.

RUNNING GEOTRANS

In the following section, bold type indicates what the user will see on or type to the alphanumeric screen. **<RETURN>** indicates the user should press the return key. Small case letters between angle brackets indicates a user supplied name.

The user should turn on the Tektronix graphics terminal and access the KGIS account via the VT220 or VT320 screen. GEOTRANS is started by typing: **GEOTRANS** at the VAX $ prompt. An introductory screen will come up and the user should press **<RETURN>**. A menu screen will appear that will give the operator the option to: 1) translate data to ORACLE, 2) translate a geofile to KGIS, or 3) Quit.

If the user selects **1** and presses **<RETURN>**, they will be asked: **Have you prepared an ORACLE Table? (y/n)**. If **n** and a **<RETURN>** is entered, the program returns to the main menu. If **y** and a **<RETURN>** is selected, the program informs the user that it is opening communications with ORACLE: **initializing communications with ORACLE database...** GEOTRANS asks the user to enter a user name and password: **Enter ORACLE account Username:** and **Enter ORACLE account Password:**. The user enters these items, pressing **<RETURN>** after each. GEOTRANS tells the user: **Logging onto ORACLE Account: <username>...** If the entries are not valid, the program returns to the main menu. If they are valid, GEOTRANS will ask for the name of the ORACLE table to send data to: **Enter ORACLE TABLE name translating to:**. The user should enter the name of the table they have created and press **<RETURN>**. If the table is not found, the error message: **I am unable to submit your command** comes up and returns to the main menu. If the table is found, the user is asked: **Enter the PLANES record file name translating from:**. The user must enter: **<plane record filename> .<extension>**. If the plane record cannot be found or is empty, the program will return to the main menu. If the file is found, GEOTRANS flashes **WORKING...** on the screen as it translates the data to the ORACLE table. After completing the translation, GEOTRANS returns to the main menu.

If the user enters a **2**, the screen clears and GEOTRANS asks: **Enter the KGIS database name:**. The user enters: **<KGIS database name> WITHOUT THE EXTENSION**. A message of **Please Wait...** displays on the screen. If the file is found, the message: **KGIS Database, OK** is displayed. The user should press **<RETURN>**.

The query: **Enter name of legend file: [<KGIS database name>]** is written to the screen. In square brackets will be the name of the KGIS database entered in the previous step as a default value. The KGIS user's manual describes in detail the structure of the legend file and how commands are used to set the attributes it contains. If the user does not know the name of the legend file or has failed to create one, press **<RETURN>** to have the program accept the default. The default file will have the same name as the KGIS database and will contain a set of default file parameters determined by KGIS. Again the message **Please wait...** appears followed by the message: **Legend file, OK.** The user should press **<RETURN>**.

GEOTRANS will ask: **Enter geofile name for translation to KGIS:.** The user should enter: **<geofile name>.<extension>** they want translated to KGIS and press **<RETURN>**. Once more, the message **Please wait...** appears followed by the message: **GEOPROGRAM file, OK.** The user should press **<RETURN>**.

During each of the three steps above, if the file is not found, the error message: **FILE NOT FOUND :::::> <filename>.<ext.>** will be displayed, followed by a question asking if the user would like to try again or quit: **Would you like to enter <file> name again? (y/n).** If **n** and a **<RETURN>** is entered, the program returns to the main menu. If **y** and a **<RETURN>** is entered then the program gives the user another chance to enter the name of the file.

After the user has entered the file names successfully, the symbol and line type files are accessed and read into arrays. The GEOTRANS program compares codes in the geofile with those in the look-up tables to assign theme names. If there is a difference, GEOTRANS will always use the alphanumeric term in the look-up table. A file status box will appear on the screen to inform the user if all of the files were opened without problem. The user should press **<RETURN>**.

The program accesses the geofile to get the first plot area and retrieves some parameters from the plot file for display on the screen, so the user can verify that the plot area record is correct. If the information is correct, the user should press **<RETURN>**.

The message **SEARCHING...** will flash at the top of the screen. Four labels: **X MAX, Y MAX, X MIN,** and **Y MIN** are placed on the screen while maximum and minimum coordinate values are searched for in the geofile. After the coordinate search is complete, a message: **FINAL MAX AND MIN VALUES** and the coordinate values appear on the screen. The user should press **<RETURN>**.

A message: **Starting geofile coordinate data translation** comes up to inform the user the translation has begun. All commands and their respective values are displayed to the screen. Codes are compared to symbol and line type files and a message of either a match or mismatch is displayed, along with what value was actually used. Coordinates are not displayed but are represented as objects drawn to the Tektronix graphics terminal.

The translation will continue without stopping unless it encounters a CHANGE TO PLOT AREA command, and then will inquire if the user would like to change plot areas. If the user enters an **n** and a **<RETURN>** the translation proceeds with the same plot area, disregarding the new plot area it found. If the user enters a **y** and a **<RETURN>** the program will proceed with information from the new plot file record. After the translation is complete, a message appears on the screen: **TRANSLATION COMPLETE**. The user should press **<RETURN>**. The program returns to the main menu and the user may select 3 to quit.

All information that is written to the screen during a coordinate data translation to KGIS is also written to a file, **<KGIS database name>.SWF**. This file may be printed for inspection and (or) deleted.

SUMMARY

Presently (December 1989), GEOTRANS is able to translate non-coordinate data from a planes record file created in GEOPROGRAM to a table in ORACLE. The data may then be queried in KGIS or queried and edited in ORACLE. The program is easy to run in that the user need only enter names of required files and have access to an ORACLE user name and password. Information in the ORACLE table can be brought up in information blocks on the Tektronix terminal or displayed on an alphanumeric terminal. These information blocks may contain some or all of the information entered in the relational database for a single or group of objects. These blocks may be brought up by entering KGISSQL commands at the keyboard or by selecting an object on a Tektronix terminal with a mouse.

For coordinate data, GEOTRANS will translate lines and (or) points into KGIS assigning them into themes that have been determined by the user and building the 2D topology. The user need only know the name of the geofile and the KGIS database name to run the translation. A file is created on disk during the coordinate translation that may be examined to help solve possible problems. The program will search for and display the maximum and minimum coordinates for X and Y in the geofile. Several KGIS databases may be combined into one database if needed and data may be edited while using KGISEDIT.

Currently, GEOTRANS sends information to the ORACLE database in a rigorously structured manner; the ORACLE table must be set up prior to translation. Development is continuing to make the translation more flexible to the user.

2D topology is sufficient for many applications, however, for the mapping of underground fractures in a shaft or drifts it is vital to have 3D topology. It is necessary to know how objects relate to one another in 3D. We are currently waiting delivery of the 3D version of the KGIS database to further refine the system.

The GEOTRANS program modular design provides flexibility and should make the task of converting to a 3D GIS or to other 2D GIS systems relatively easy.

We are also waiting for delivery of polygon fill procedures and procedures that allow plotting of a greater variety of symbols. KGIS currently supports only three types of symbols (a cross, an open circle, and a small circle with a line through it) that can be sent to an output device.

REFERENCES CITED

Dueholm, K. S., 1989, GEOPROGRAM a program for geologic photogrammetry on the Kern DSR analytical plotter User's Manual, Open-File Report 89-481, 47 p. (NNA.910415.0002)

Dueholm, K. S. and Coe, J. A., 1989, GEOPROGRAM: Program for Geologic Photogrammetry: The Compass. (NNA.910415.0003)

Herring, J. R., 1987, TIGRIS: Topologically integrated geographic information system: International Symposium on Computer-Assisted Cartography, 8th, Baltimore, Maryland, 1987, Proceedings, p. 282-291. (NNA.910415.0004)

Ingram, K. J. and Phillips, W. W., 1987, Geographic information processing using a SQL-based query language: International Symposium on Computer-Assisted Cartography, 8th, Baltimore, Maryland, 1987, Proceedings, p. 326-335. (NNA.910415.0005)

KERN and CO., Ltd., CH-5001, Aarau, Switzerland (Leica, Inc.).

KORK Systems, Inc., 1988, KORK Geographic Information System User's Manual: KORK Systems, Inc., Bangor, Maine. (NNA.910415.0006)

ORACLE Corporation, 1986, Oracle for DEC VAX/VMS installation and user's guide: Oracle Corporation, Belmont, California.

APPENDIX
Program Listing

```
PROGRAM GEOTRANS (INPUT, OUTPUT);


%include '$disk1:[kgis]kgislib.env'
%include 'geoglo.pas'


CONST
  Change = 'CHANGE TO ';
  Clean_Scr = 24;
  Max_Array = 255;


TYPE
  Element_Type = (ET_Point, ET_Line);
  Line_Type_Record = RECORD
                        LT_Number : INTEGER;
                        LT_Text   : VARYING [256] OF CHAR;
                     END;

  LTR_Array = ARRAY [1..Max_Array] OF Line_Type_Record;

  Symbol_Type_Record = RECORD
                        ST_Number : INTEGER;
                        ST_Text   : VARYING [256] OF CHAR;
                     END;

  STR_Array = ARRAY [1..Max_Array] OF Symbol_Type_Record;


VAR
  S_Label             : String_10 ;
  Check_String        : String_10 ;
  i, j, Fracount      : INTEGER ;
  First_Pa            : INTEGER ;
  Line_Count          : INTEGER ;
  Symbol              : INTEGER ;
  Husk_Area           : INTEGER ;
  Nchar               : INTEGER ;
  Line_Type_Number    : INTEGER;
  Symbol_Type_Number  : INTEGER;
  Pen_Num_Buf         : INTEGER;
  Cont_Line_Buf       : BOOLEAN;
  Slick_Line_Buf      : BOOLEAN;
  Minimax             : BOOLEAN ;
  OK                  : BOOLEAN ;
  Retry               : BOOLEAN ;
  Dun                 : BOOLEAN ;
  Finished            : BOOLEAN ;
  Pause               : BOOLEAN ;
  Error               : BOOLEAN ;
  Look_Forward        : BOOLEAN;
  File_Open           : BOOLEAN ;
  Strike_Buf          : DOUBLE;
```

```
Rotation_Buf        : DOUBLE;
{Dip_Buf            : DOUBLE;}
R                   : DOUBLE ;
Count               : DOUBLE ;
X_Table_Out         : DOUBLE;
Y_Table_Out         : DOUBLE;
Z_Table_Out         : DOUBLE;
S_X_Max             : DOUBLE;
S_X_Min             : DOUBLE;
S_Y_Max             : DOUBLE;
S_Y_Min             : DOUBLE;
Annotation          : String_40;
Read_File_String    : String_40;
Line_Type_Text      : String_40;
Symbol_Type_Text    : String_40;
Screen_Write_File   : String_20;
Input_File_Name     : String_20;
GEOP_File_Name      : String_20;
TC_Name_File        : string_14;
YN                  : String_3;
SW_File             : TEXT;
Trans_Coord         : TEXT;
Drift_File          : TEXT;
Line_Types          : TEXT;
Symbol_Types        : TEXT;
Point_Buffer        : Real_Array_3;
LT_Store            : LTR_Array;
ST_Store            : STR_Array;
Line_Type_Buf       : VARYING [256] OF CHAR;
Sym_Type_BUF        : VARYING [256] OF CHAR;
ET_Element          : Element_Type;
SD_Buf              : string256 ;


{ KORK-KGIS Variables }
First_Point:    BOOLEAN ;
Flag:           BOOLEAN ;
new_leg:        BOOLEAN ;
cartE:          Element ;
nodeE:          Element ;
themE:          Element ;
edgeE:          Element ;
Coll_Parm:      coll_parmType ;
Start_Loc:      locationType ;
Loc_Buf:        locationType ;
End_Loc:        locationType ;
Id_List:        idListType ;
Node_List:      idListType ;
Err_Code:       ErrorType ;
Sql_Command:    string256 ;
Varying_Label:  string256 ;
Varying_Label2:string256;
Sql_Pil:        sql_cursor ;
User_Name:      string20 ;
```

```
Pass_Word:       string20 ;
Table_Name:      string20 ;
K_Name:          string20;
Theme_Name:      nameType ;
Leg_Name:        nameType ;
Kdb_Name:        nameType ;
Legend:          legLineType ;
Xlow, Xhigh:     baseType ;
Ylow, Yhigh:     baseType ;
Db_Range:        rangeType ;
Response :       CHAR ;
I_response:      INTEGER ;


{projection parameters}
W, H, B            : DOUBLE ;
Drift1_Parameters  : Tunnel_Parameter_Type ;
xyzs               : Real_Array_3 ;
xyz1               : Real_Array_3 ;
xyz2               : Real_Array_3 ;
```

```
{**********************************************************}
FUNCTION  Alfa(x,y: double): double;
VAR
  a, offset     : DOUBLE;
  direct, octet : INTEGER;

BEGIN
  IF x >= 0 THEN octet := 4 ELSE octet := 0;
    IF y >= 0 THEN octet := octet + 2;
    IF abs(x) <= abs(y) THEN
       BEGIN
         a := x;
         x := y;
         y := a;
         direct := -1
       END
    ELSE
       direct := 1;
    IF direct > 0 THEN octet := octet + 1;
    CASE octet of
       7  :   offset := 0;
       6,2:   offset := Phi/2;
       3,1:   offset := Phi;
       0,4:   offset := 3 * Phi/2;
       5  :   offset := 2 * Phi;
    END; { case }
    IF x = 0 THEN
      alfa := 0
    ELSE alfa := offset + direct * arctan(y/x);
END; {* alfa *}


{**********************************************************}
PROCEDURE Open_Input_File;
VAR
  i : INTEGER;

BEGIN
  Input_File_Name := GEOP_File_Name;
  OPEN ( FILE_VARIABLE    := Output_File_Data,
         FILE_NAME        := Input_File_Name,
         HISTORY          := OLD,
         ACCESS_METHOD    := DIRECT,
         ERROR            := CONTINUE );

  RESET (Output_File_Data, ERROR := CONTINUE);

  Output_File_Pointer := 1;
  FIND (Output_File_Data, Output_File_Pointer);
  READ (Output_File_Data, Logical_Record);
  Logical_Record_Pointer := 1 ;
END; {* Open_Input_File *}
```

```
{**********************************************************************}
PROCEDURE BIOF_Backup_In_Output_File
          (VAR Edit_String: String_80; VAR Nchar: INTEGER);
VAR
   i     : INTEGER;
   Stop  : BOOLEAN;
   First : BOOLEAN;

BEGIN

  { CHR(%O'15')CHR(%O'12') }

  FOR I := 1 to 80 DO Edit_String[I] := ' ';
  Nchar := 0;
  Stop  := FALSE;
  First := TRUE;

  REPEAT
    IF Logical_Record_Pointer <= 1 THEN
      BEGIN
             IF Output_File_Pointer = 1 THEN
               BEGIN
                 Stop := TRUE;
               END
             ELSE BEGIN
            Output_File_Pointer := Output_File_Pointer - 1;
            FIND (Output_File_Data,Output_File_Pointer);
            READ (Output_File_Data, Logical_Record);
            Logical_Record_Pointer := 513;
                  END;
      END;

    IF NOT Stop THEN
      BEGIN
       Logical_Record_Pointer := Logical_Record_Pointer - 1;
       IF Logical_Record[Logical_Record_Pointer]
                    IN [CHR(%O'15'), CHR(%O'12')] THEN
          BEGIN
           IF First THEN
            BEGIN
             IF Logical_Record [Logical_Record_Pointer] =
                 CHR(%O'15') THEN
               First := FALSE;
            END
           ELSE BEGIN
             Stop:= TRUE;
             Logical_Record_Pointer :=
             Logical_Record_Pointer + 1;
             IF Logical_Record_Pointer > 512 THEN
               BEGIN
                Output_File_Pointer :=
                Output_File_Pointer + 1;
                FIND (Output_File_Data,
                      Output_File_Pointer);
```

```
                READ (Output_File_Data, Logical_Record);
                Logical_Record_Pointer := 1;
               END;
           END ;
          END
      ELSE BEGIN
        Nchar := Nchar + 1;
        Edit_String[81 - Nchar] :=
        Logical_Record [Logical_Record_Pointer];
       END;
      END;
UNTIL Stop;

IF Nchar <> 0
    THEN FOR i := 1 TO Nchar DO
      BEGIN
       Edit_String[i] := Edit_String[80 - Nchar + i];
       Edit_String[80 - Nchar + i] := ' ';
      END;

END; (* BIOF_Backup_In_Output_File *)
```

```
{*************************************************************}
PROCEDURE FIOF_Forward_In_Output_File
          (VAR Edit_String: String_80; VAR Nchar: INTEGER);
VAR
   i     : INTEGER;
   Stop  : BOOLEAN;
   Quit  : String_4;

PROCEDURE Step;
  BEGIN
    IF Logical_Record_Pointer >= 512 THEN
      BEGIN
        Output_File_Pointer := Output_File_Pointer + 1;
        FIND (Output_File_Data, Output_File_Pointer);
        READ (Output_File_Data, Logical_Record);
        Logical_Record_Pointer := 1;
      END
  ELSE BEGIN
        Logical_Record_Pointer:= Logical_Record_Pointer + 1;
       END;
  END; {* Step *}

BEGIN
  FOR i := 1 to 80 DO Edit_String[i] := ' ';
  Nchar := 0;
  Stop  := FALSE;
  REPEAT
   IF Logical_Record [Logical_Record_Pointer] <>
      CHR(%O'15') THEN
        BEGIN
          Nchar:= Nchar + 1;
          Edit_String[Nchar] :=
           Logical_Record [Logical_Record_Pointer] ;
          Step;
        END
   ELSE BEGIN
          Stop := TRUE;
          Step; Step;
        END;
  UNTIL Stop;

  FOR i := 1 to 4 DO Quit[I] := Edit_String[I];
  IF Quit = 'QUIT' THEN
    BEGIN
      { WRITELN ( chr (bell) ) ; }
      BIOF_Backup_In_Output_File ( Edit_String, Nchar );
    END;
END; (* FIOF_Forward_In_Output_File *)
```

```
{*********************************************************}
FUNCTION Change_To ( Mode: PACKED ARRAY
                     [f..l: INTEGER] OF CHAR ): BOOLEAN ;
VAR
  i    : INTEGER;
  buuh : BOOLEAN;

BEGIN
  Buuh      := FALSE;
  Change_To := TRUE;
  i         := f - 1;
  REPEAT
    i := i + 1;
    IF Plot_String[i + 10] <> Mode[i] THEN Buuh := TRUE;
  UNTIL (i = l) OR Buuh;
  IF Buuh THEN Change_To := FALSE;
END; {* Change_To *}
```

```
{*************************************************************}
PROCEDURE Numeric ( String                   : String_40;
                    Number_Of_Characters     : INTEGER;
                    VAR Value                : DOUBLE;
                    VAR Error                : BOOLEAN );
VAR
  Deno           : DOUBLE;
  i, n, Zero     : INTEGER;
  Negative_Value : BOOLEAN;

BEGIN
     Value := 0.0;
     Zero  := ORD('0');
     N     := Number_Of_Characters;
     Error := FALSE;
     Negative_Value := FALSE;
     IF N <= 0 THEN Error := TRUE;
     IF NOT Error THEN FOR i := 1 TO n DO
       IF NOT (String[I] IN ['0'..'9',' ','.','+','-']) THEN
         Error := TRUE;

     IF NOT Error THEN
       BEGIN
         i := 0;
         REPEAT
           i := i + 1;
         UNTIL (i = n) OR (String[I] <> ' ');
       END;

     IF NOT Error THEN
       BEGIN
         IF String[I] = '-' THEN
           BEGIN
             Negative_Value := TRUE;
             i := i + 1;
           END;
         IF String[I] = '+' THEN i := i + 1;
         IF i > n THEN Error := TRUE;
       END;

     IF (NOT Error) AND (i <= n) THEN
       IF (String[I] <> '.') THEN
         REPEAT
           IF NOT ( String[I] IN ['0'..'9','.'] ) THEN
             Error := TRUE;
           IF NOT Error THEN
             Value := 10 * Value + (ORD(String[I]) - Zero);
             i := i + 1;
         UNTIL Error OR (i > n) OR (String[I] = '.');

     IF NOT Error THEN
       IF (String[I] = '.') THEN i := i + 1;

     IF NOT Error AND (i <= n) THEN
```

```
        BEGIN
          Deno := 1.0;
          REPEAT
            Deno := Deno * 10;
            IF NOT (String[I] IN ['0'..'9']) THEN
              Error := TRUE;
            IF NOT Error THEN
              Value := Value + (ORD(String[i]) -Zero)/Deno;
            i := i + 1;
          UNTIL Error OR (i > n);
        END;
      IF NOT Error AND Negative_Value THEN Value := -Value;
      IF Error THEN Value := 0.0;
END; (* Numeric *)
```

```
{********************************************************}
PROCEDURE GA_Get_Annotation ( VAR From_To : INTEGER;
         VAR In_String : PACKED ARRAY[F..L:INTEGER] OF CHAR;
         VAR String    : String_40; VAR Error : BOOLEAN );
VAR
  j, i  : INTEGER;

BEGIN
  Error := TRUE;
  i := From_To - 1;
  REPEAT
    i := i + 1;
    IF In_String [I] <> ' ' THEN Error := FALSE;
    UNTIL NOT Error OR (i = L);

    IF NOT Error THEN
      BEGIN
        String := Blank_40;
        i := i - 1;
        j := 0;
        REPEAT
          i := i + 1;
          j := j + 1;
          String[j] := In_String[i];
        UNTIL (i = L) OR (j = 40);
        From_To := i;
      END;
END; (* GA_Get_Annotation *)


{*********************************************************}
PROCEDURE GV_Get_Value ( VAR From_To : INTEGER;
        VAR In_String : PACKED ARRAY [F..L:INTEGER] OF CHAR;
        VAR Integer_Value: INTEGER; VAR Real_Value : DOUBLE;
        VAR Error : BOOLEAN );
VAR
   String       : String_40;
   Value        : DOUBLE;
   j, i         : INTEGER;
   First_Time   : BOOLEAN;
   Finished     : BOOLEAN;

BEGIN
  i          := From_To;
  String     := Blank_40;
  j          := 0;
  First_Time := TRUE;
  Finished   := FALSE ;
  REPEAT
    i := i + 1;
    IF First_Time AND ( In_String [I] <> ' ' ) THEN
      First_Time := FALSE;
      IF NOT First_Time THEN
        BEGIN
```

```
          j := j + 1;
          String[j] := In_String[i];
          IF String[j] = ' ' THEN
            BEGIN
              Finished := TRUE;
              j := j - 1;
            END;
        END;
      IF (j = 40) OR (i = L) THEN Finished := TRUE;
  UNTIL Finished;

  IF j > 0 THEN
    Numeric (String, j, Value, Error)
  ELSE
    Error := TRUE;
  IF NOT Error THEN
    BEGIN
      From_To := i;
      Real_Value := Value;
      Integer_Value := TRUNC(Value);
    END;
END; {* GV_Get_Value *}
```

```
{***********************************************************}
PROCEDURE Slet_Blanks ( In_String: PACKED ARRAY
                        [F..L: INTEGER] OF CHAR;
                        VAR Out_String: string256 );
VAR
  i,j  : INTEGER;
  Stop : BOOLEAN;

BEGIN
  j    := L + 1;
  Stop := FALSE;

  REPEAT
    j := j - 1;
    IF j = 0 THEN
      stop := TRUE
    ELSE
      IF (In_String[j] <> ' ') THEN Stop := TRUE;
  UNTIL Stop ;

  Out_String := Substr (In_String, 1, j);
{  FOR i := 1 TO j DO Out_String[i] := In_String[i];}
END; {* Slet_Blanks *}
```

```
{*********************************************************}
PROCEDURE Write_Plane_Record_To_Oracle_Table;
VAR
   i, Index, Numpnt  : INTEGER;
   End_Record        : BOOLEAN;
   Planerecfound     : BOOLEAN;
   Plane_Record_File : String_20;
   p                 : Planes;
   Pkind_String      : Varying[11] of CHAR;
   Varying_Label     : String256;

BEGIN
  CURPOS_Cursor_Position (7,10);
  WRITELN
   ('Enter the PLANES record file name translating from: ');
  READLN (Plane_Record_File);

  SELGRA_Select_Graphics ('B');
  DBWIDT_Double_Width (24);
  CURPOS_Cursor_Position (24,14);
  WRITELN ('WORKING...');

  OPEN ( FILE_VARIABLE   :=  Planefile,
         FILE_NAME       :=  Plane_Record_File,
         HISTORY         :=  OLD,
         ACCESS_METHOD   :=  DIRECT,
         ERROR           :=  CONTINUE );

  RESET ( Planefile, ERROR := CONTINUE );

  CASE STATUS (Planefile) OF
     -1    :  BEGIN
                 WRITELN
                  ('PLANE RECORD FILE IS EMPTY', crlf);
                 Planerecfound := FALSE;
              END;
      0    :  BEGIN
                 Planerecfound := TRUE;
              END;
      OTHERWISE
              BEGIN
                WRITELN ('PLANE RECORD FILE NOT FOUND::> ' +
                           plane_record_file, crlf);
                Planerecfound := FALSE;
              END;
  END; { of case }

  IF Planerecfound THEN
    RESET ( Planefile, ERROR := CONTINUE );

  i := 0;
  REPEAT
   i := i + 1;
   FIND (Planefile, i);
```

```
READ (Planefile, p);
IF p.kind = ptom THEN
  End_Record := TRUE ;
IF End_Record = FALSE THEN
  BEGIN
     Slet_Blanks (p.name, Varying_Label);
     rdbBind_strval
     (SQL_pil, 1, Varying_Label, 0, Err_Code);

     CASE p.kind OF
       strike   :     Pkind_String := 'strike' ;
       apparent :     Pkind_String := 'apparent' ;
       ppoint   :     Pkind_String := 'ppoint' ;
       field    :     Pkind_String := 'field' ;
       comphor  :     Pkind_String := 'comphor' ;
       compfol  :     Pkind_String := 'compfol' ;
       fold     :     Pkind_String := 'fold' ;
       ptom     :     Pkind_String := 'ptom' ;
     END; { of case }

     rdbBind_strval
     (SQL_pil, 2, Pkind_String, 0, Err_Code);

     Numpnt := p.n;
     rdbBind_intval
     (SQL_pil, 3, Numpnt, 0, Err_Code);

     Index := 3;
     FOR j := 1 TO 9 DO
       BEGIN
         Index := Index + 1;
         rdbBind_dbleval
        (SQL_pil, Index, p.ne[j], 0, Err_Code);
       END;

     FOR j := 1 TO 3 DO
       BEGIN
         Index := Index + 1;
         rdbBind_dbleval
        (SQL_pil, Index, p.nv[j], 0, Err_Code);
       END;

     FOR j := 1 TO 3 DO
       BEGIN
         Index := Index + 1;
         rdbBind_dbleval
        (SQL_pil, Index, p.qv[j], 0, Err_Code);
       END;

     FOR j := 1 TO 3 DO
       BEGIN
         Index := Index + 1 ;
         rdbBind_dbleval
        (SQL_pil, Index, p.cv[j], 0, Err_Code);
```

```
          END;

         rdbBind_dbleval
        (SQL_pil, 22, p.str, 0, Err_Code);

         rdbBind_dbleval
        (SQL_pil, 23, p.dip, 0, Err_Code);

         rdbBind_dbleval
        (SQL_pil, 24, p.dir, 0, Err_Code);

         rdbBind_dbleval
        (SQL_pil, 25, p.ms, 0, Err_Code);

         rdbBind_dbleval
        (SQL_pil, 26, p.md, 0, Err_Code);

         rdbBind_dbleval
        (SQL_pil, 27, p.mp, 0, Err_Code);

         rdbExec_SQL (SQL_pil, Err_Code);
         rdbCommit (Err_Code);
      END;
  UNTIL (i = 99) OR (End_Record = TRUE);
  CLOSE (Planefile);
END; {* Write_Plane_Record_To_Oracle_Table *}
```

```
{********************************************************}
PROCEDURE PFOT_Planes_File_To_Oracle_Translator;

BEGIN
  IC_Invisible_Cursor;
  CURPOS_Cursor_Position (24,10);
  WRITELN
  ('Initializing communications with the ORACLE
database...');
  IF rdbInit_comm ( Err_Code ) THEN
    BEGIN
      ERASCR_Erase_Screen;
      SELSGR_Select_Special_Graphics;
      IC_Invisible_cursor;
      CURPOS_Cursor_Position (7,16);
      WRITELN ('l');
      FOR i := 17 TO 71 DO
        BEGIN
          CURPOS_Cursor_Position (7,i);
          WRITELN ('q');
        END;
      CURPOS_Cursor_Position (7,71);
      WRITELN ('k');
      FOR i := 8 TO 11 DO
        BEGIN
          CURPOS_Cursor_Position (i,71);
          WRITELN ('x');
        END;
      CURPOS_Cursor_Position (12,71);
      WRITELN ('j');
      FOR i := 70 DOWNTO 17 DO
        BEGIN
          CURPOS_Cursor_Position (12,i);
          WRITELN ('q');
        END;
      CURPOS_Cursor_Position (12,16);
      WRITELN ('m');
      FOR i := 11 DOWNTO 8 DO
        BEGIN
          CURPOS_Cursor_Position (i,16);
          WRITELN ('x');
        END;
      SELASC_Select_ASCII;
      CURPOS_Cursor_Position (9,20);
      WRITELN ('Enter Oracle Account User Name: ');
      READLN (User_Name);
      CURPOS_Cursor_Position (10,20);
      WRITELN ('Enter Oracle Account Password: ');
      READLN (Pass_Word);
      ERASCR_Erase_Screen;
      CURPOS_Cursor_Position (2,10);
      WRITELN
      ('Logging onto Oracle Acount: ', User_Name, '...');
      IF rdbLogon ( User_Name, Pass_Word, Err_Code ) THEN
```

```
        BEGIN
          CURPOS_Cursor_Position (5,10);
          WRITELN
             ('Enter Oracle TABLE Name Translating TO: ');
          READLN (Table_Name);
          CS_Capitalise_String (Table_Name);
          Slet_Blanks (Table_Name, Varying_Label2);
          sql_command:='insert into '+ Varying_Label2 +
                        'values(:sname,:skind,:i
                        numpnt,:dne1,:dne2,:dne3,:dne4,
                        :dne5,:dne6,:dne7,:dne8,:dne9,
                        :dnv1,:dnv2,:dnv3,:dqv1,:dqv2,
                        :dqv3,:dcv1,:dcv2,:dcv3,:dstrike,
                        :ddip,:ddir,:dms,:dmd,:dmp)';
          IF rdbSubmit_SQL
            (sql_command, sql_pil, err_code) THEN
            BEGIN
               Write_Plane_Record_To_Oracle_Table;
               rdbRelease_SQL (SQL_pil, err_code);
            END;
          IF Err_Code <> 1 THEN
            WRITELN
          ('I am unable TO submit your SQL command', crlf);
            rdbLogoff (Err_Code);
        END;
      IF Err_Code <> 1 THEN
        WRITELN
        ('I am unable TO logon your ORACLE account', crlf);
      rdbAlldone (err_code) ;
      SELASC_Select_ASCII;
      SELGRA_Select_Graphics ('O');
      VC_Visible_Cursor;
    END;
END; {* PFOT_Planes_File_To_Oracle_Translator *}
```

```
{************************************************************}
PROCEDURE  Draw_Line_To_Screen (VAR loc: locationType);

BEGIN
   AGIsymBlk_start (legend^.symBlk, AGI_LINE );
   WITH loc DO
     AGIdraw ( x, y);
   AGIsymBlk_end (legend^.symBlk, AGI_LINE );
END; {* Draw_Line_To_Screen *}


{************************************************************}
PROCEDURE  Draw_Point_To_Screen (VAR loc: locationType);

BEGIN
   AGIsymBlk_start (legend^.symBlk, AGI_POINT);
   WITH loc DO
     AGIdraw_circle ( x, y, 100.0);
   AGIsymBlk_end (legend^.symBlk, AGI_POINT );
END; {* Draw_Point_To_Screen *}


{************************************************************}
PROCEDURE  Move_To_Start_Of_Line (VAR loc: locationType);

BEGIN
   set_cart_infoLoc (loc, MIDDLE_CENTER, cartE);
   AGIsymBlk_start (legend^.symBlk, AGI_LINE );
   WITH loc DO
     AGImove (x, y);
   AGIsymBlk_end (legend^.symBlk, AGI_LINE );
END; {* Move_To_Start_Of_Line *}


{************************************************************}
PROCEDURE  Move_To_Node_Location (VAR loc: locationType);

BEGIN
   set_cart_infoLoc (loc, MIDDLE_CENTER, cartE);
   AGIsymBlk_start (legend^.symBlk, AGI_POINT );
   WITH loc DO
     AGImove ( x, y);
   AGIsymBlk_end (legend^.symBlk, AGI_POINT);
END; {* Move_To_Node_Location *}
```

```
{**********************************************************}
PROCEDURE OTT_Object_To_Table (X_Object, Y_Object,
                               Z_Object: DOUBLE ;
                               VAR   X_Table, Y_Table,
                               Z_Table: DOUBLE ) ;
VAR
  hg, ht : Real_Array_4 ;
  i, j   : INTEGER;
  Error  : BOOLEAN;

BEGIN
  hg[1] := X_Object;
  hg[2] := Y_Object;
  hg[3] := Z_Object;
  hg[4] := 1.00;

  IF (Transformation = normal_drift) THEN
    BEGIN
      Ground_To_Drift1_Periphery
        (Tunnel_Parameters, hg, hg, Error);
      hg[2] := hg[2] + Tunnel_Parameters.per/2.00;
      hg[4] := 1;
    END;
  IF (Transformation = radial_drift) THEN
    BEGIN
      Ground_To_Drift2_Periphery
        (Tunnel_Parameters, hg, hg, Error);
      hg[2] := hg[2] + Tunnel_Parameters.per/2.00;
      hg[4] := 1;
    END ;
  IF (Transformation = shaft) THEN
    BEGIN
      Ground_To_Shaft_Periphery
        (Tunnel_Parameters, hg, hg, Error);
      hg[1] := hg[1] + Tunnel_Parameters.per/2.00;
      hg[4] := 1;
    END;

  FOR i := 1 TO 4 DO
    BEGIN
      ht[i] := 0;
      FOR j := 1 TO 4 DO
         ht[i] := ht[i] + gtt_mat[i,j] * hg[j];
    END;
  FOR i := 1 TO 3 DO
    ht[i] := ht[i] / ht[4];

  X_Table := ht[1]; Y_Table := ht[2]; Z_Table := ht[3];
END; {* OTT_Object_To_Table *}
```

```
{***********************************************************}
PROCEDURE Translate_Coordinates (VAR XYZ_Measured:
                                 Real_Array_3;
                                 VAR Error: BOOLEAN;
                                 VAR loc: locationType );
BEGIN
  OTT_Object_To_Table
      (XYZ_Measured[1], XYZ_Measured[2], XYZ_Measured[3],
       X_Table_out, Y_Table_out, Z_Table_out);
  world3D_to_loc (METERS, X_Table_out,
                          Y_Table_out,
                          Z_Table_out, loc);
   IF Minimax THEN
     BEGIN
       IF X_Table_out > S_X_Max THEN
         BEGIN
           S_X_Max := X_Table_out;
           CURPOS_Cursor_Position (5,8);
           WRITELN ('X MAX:  ', S_X_Max:7:3);
         END;
       IF X_Table_out < S_X_Min THEN
         BEGIN
            S_X_Min := X_Table_out;
            CURPOS_Cursor_Position (5,45);
            WRITELN ('X MIN:  ', S_X_Min:7:3);
         END;
       IF Y_Table_out > S_Y_Max THEN
         BEGIN
            S_Y_Max := Y_Table_out;
            CURPOS_Cursor_Position (8,8);
            WRITELN ('Y MAX:  ', S_Y_Max:7:3);
         END;
       IF Y_Table_out < S_Y_Min THEN
         BEGIN
            S_Y_Min := Y_Table_out;
            CURPOS_Cursor_Position (8,45);
            WRITELN ('Y MIN:  ', S_Y_Min:7:3);
         END;
     END;
END; {* Translate_Coordinates *}
```

```
{**********************************************************}
PROCEDURE Get_Point (VAR loc: locationType;
                     VAR flag: BOOLEAN );
VAR
  i : INTEGER;

BEGIN
  IF First_Point THEN
     BEGIN
       FOR i:= 1 TO 3 DO
          XYZ_Measured[i] := Point_Buffer[i];
       IF NOT Error THEN
          BEGIN
            Translate_Coordinates
                     (XYZ_Measured, Error, loc);
          END
       ELSE
          BEGIN
            WRITELN (Error, crlf);
            WRITELN (SW_File, Error, crlf);
            WRITELN (Check_String, crlf);
            WRITELN (SW_File, Check_String, crlf);
          END;
       CASE ET_Element OF
         Et_Point : BEGIN
                      Move_To_Node_Location (loc);
                      Draw_Point_To_Screen (loc);
                    END;
         Et_Line  : BEGIN
                      Move_To_Start_Of_Line (loc);
                      First_Point := FALSE;
                    END;
       END; {of case}
     END
  ELSE
     BEGIN
       Flag := FALSE;
       FIOF_Forward_In_Output_File (Plot_String, Nchar);
       FOR i := 1 TO 10 DO
          Check_String[i] := Plot_String[i];
       CASE Check_String[1] OF
          ' ' : BEGIN
                  J := 1;
                  GV_Get_Value (j, Plot_String,
                                i, XYZ_Measured[1], Error);
                  GV_Get_Value (j, Plot_String,
                                i, XYZ_Measured[2], Error) ;
                  GV_Get_Value (j, Plot_String,
                                i, XYZ_Measured[3], Error) ;
                  IF NOT Error THEN
                     BEGIN
                       Translate_Coordinates
                                  (XYZ_Measured, Error, loc);
                     END
```

```
                  ELSE
                    BEGIN
                      WRITELN (Error, crlf);
                      WRITELN (SW_File, Error, crlf);
                      WRITELN (Check_String, crlf);
                      WRITELN
                             (SW_File, Check_String, crlf);
                    END;
                  Draw_Line_To_Screen (loc);
                END;
          OTHERWISE
            BEGIN
              Look_Forward := FALSE;
              Flag := TRUE;
            END;
        END; {of case}
      END;
END; {* Get_Point *}


{*****************************************************************}
PROCEDURE Process_New_Edge (VAR edgeE: Element);

BEGIN
  add_id_to_end (topo_recnr (edgeE), id_list);
END; {* Process_New_Edge *}


{*****************************************************************}
PROCEDURE Process_New_Node (VAR nodeE: Element);

BEGIN
  add_id_to_end (topo_recnr (nodeE), node_list);
END; {* Process_New_Node *}


{*****************************************************************}
PROCEDURE ICL_Initialize_Cart_And_Id;

BEGIN
  init_new_cart (0, varying_label, cartE);
  CASE ET_Element OF
    ET_Point: init_id_list (node_list);
    ET_Line : init_id_list (id_list);
  END; { of case }
END; {* ICL_Initialize_Cart_And_Id *}
```

```
{************************************************************}
PROCEDURE Build_Or_Abort;
BEGIN
  IF Error_Raised THEN
     BEGIN
       Display_Error (output);
       trans_abort;
     END
  ELSE
     BEGIN
       CASE ET_Element OF
         ET_Point :  BEGIN
                       build_point
                             (node_list, themE, cartE);
                       trans_end;
                       free_id_list (node_list);
                     END;
         ET_Line  :  BEGIN
                       build_line (id_list, themE, cartE);
                       trans_end;
                       free_id_list (id_list);
                     END;
       END; {of case}
     END;
END; {* Build_Or_Abort *}


{************************************************************}
PROCEDURE GS_Get_String;

BEGIN
  WRITELN ('Pen down and drawing... ', Geo_Label, crlf);
  WRITELN
  (SW_File, 'Pen down and drawing... ', Geo_Label, crlf);
  Slet_Blanks (Geo_Label, Varying_Label);
  ICL_Initialize_Cart_And_Id;
  First_Point := TRUE;
  Get_Point (Start_Loc, Flag);
  Collect_String
            ( Get_Point, Process_New_Edge,
              Coll_Parm, Start_Loc, End_loc );
  Build_Or_Abort;
END; {* GS_Get_String *}
```

```
{************************************************************}
PROCEDURE GN_Get_Node;

BEGIN
  WRITELN
     ('Pen down and drawing NODE... ', Varying_Label, crlf);
  WRITELN (SW_File,'Pen down and drawing NODE... ', +
                                     Varying_Label, crlf);
  ICL_Initialize_Cart_And_Id;
  First_Point := TRUE;
  Get_Point (start_loc, flag);
  Create_Isol_Node (start_loc, nodeE);
  Process_New_Node (nodeE);
  Build_Or_Abort;
END; {* GN_Get_Node *}


{************************************************************}
PROCEDURE IST_Initialize_And_Store_Theme;
BEGIN
  IF NOT them_found (Theme_Name, themE) THEN
    BEGIN
      WRITELN
             ('Initializing New Theme: ', Theme_Name, crlf);
      WRITELN
     (SW_File,'Initializing New Theme: ', Theme_Name, crlf);
      WRITELN (crlf);
      WRITELN (SW_File,crlf);
      CASE ET_Element OF
         ET_Point : Init_New_Theme
                                (Theme_Name, POIN, themE);
         ET_Line  : Init_New_Theme
                                (Theme_Name, LINE, themE);
       END; {of case}
       store_theme (themE);
     END;

  Legend := legend_line (Theme_Name, Leg_Name, new_leg);
END; {* IST_Initialize_And_Store_Theme *}
```

```
{**********************************************************}
PROCEDURE
GPCGTT_Geo_Program_Compute_Ground_To_Trans_Transformation
                          ( VAR Par: Plot_Definition_Record;
                            VAR GtT: Real_Array_4_4 );
VAR
  i, j                 : INTEGER;
  xyzd1, xyzg1, xyzg2  : Real_Array_3;
  W, H, B              : DOUBLE;
  Error                : BOOLEAN;

BEGIN
  WITH Par DO
    BEGIN
      FOR i := 1 TO 4 DO
        BEGIN
          FOR j := 1 TO 4 DO GtT[i,j] := 0;
          GtT[i,i] := 1.0
        END;

      IF (Trans_Type[1] = 'O')
      OR (Trans_Type[1] = 'P')
      OR (Trans_Type[1] = 'D')
      OR (Trans_Type[1] = 'I') THEN
        BEGIN
          CGTTM_Compute_Ground_To_Trans_Matrix (Par, GtT);
        END;

      IF (Trans_Type[1] = 'N')
      OR (Trans_Type[1] = 'R')
      OR (Trans_Type[1] = 'S') THEN
        BEGIN
          xyzd1[1] := 0.00;
          xyzd1[2] := 0.00;
          xyzd1[3] := Camera_Height;
          xyzg1[1] := X_Index;
          xyzg1[2] := Y_Index;
          xyzg1[3] := Z_Index;
          xyzg2[1] := E_Factor;
          xyzg2[2] := N_Factor;
          xyzg2[3] := H_Factor;
          W        := View_Direction;
          H        := Zenith_Angle;
          B        := Focal_Length;

          IF (Trans_Type[1] = 'N') THEN
            BEGIN
              Compute_Drift1_Transformation_Parameters
                 (xyzd1, xyzg1, xyzg2, W, H, B,
                  Tunnel_Parameters, Error);
              Y_Trans_Index := - Tunnel_Parameters.per/2.00;
              Ver_Size :=
                   Tunnel_Parameters.per*1000/Y_Plot_Scale;
            END;
```

```
        IF (Trans_Type[1] = 'R') THEN
          BEGIN
            Compute_Drift2_Transformation_Parameters
               (xyzd1, xyzg1, xyzg2, W, H, B,
                Tunnel_Parameters, Error);
            Y_Trans_Index := - Tunnel_Parameters.per/2.00;
            Ver_Size :=
                 Tunnel_Parameters.per*1000/Y_Plot_Scale;
          END;

        IF (Trans_Type[1] = 'S') THEN
          BEGIN
            Compute_Shaft_Transformation_Parameters
               (xyzd1, xyzg1, xyzg2, B,
                Tunnel_Parameters, Error);
            X_Trans_Index := - Tunnel_Parameters.per/2.00;
            Hor_Size :=
                 Tunnel_Parameters.per*1000/X_Plot_Scale;
          END;
  END;

  IF Trans_Type[1] = 'O' THEN
     Transformation := orthographic;
  IF Trans_Type[1] = 'P' THEN
     Transformation := perspective;
  IF Trans_Type[1] = 'I' THFN
     Transformation := isometric;
  IF Trans_Type[1] = 'D' THEN
     Transformation := dimetric;
  IF Trans_Type[1] = 'N' THEN
     Transformation := normal_drift;
  IF Trans_Type[1] = 'R' THEN
     Transformation := radial_drift;
  IF Trans_Type[1] = 'S' THEN
     Transformation := shaft;

 END; {of with}
END;
{GPCGTT_Geo_Program_Compute_Ground_To_Trans_Transformation}
```

```
{*********************************************************}
PROCEDURE CPAM_Change_Plot_Area_Modify
                     (VAR Area_Number: INTEGER);
VAR
  Plotfile    : FILE OF Plot_Definition_Record;
  i, Field    : INTEGER;
  String      : String_40;
  Save_String: PACKED ARRAY[1..36] OF CHAR;

BEGIN
   OPEN ( FILE_VARIABLE   := Plotfile,
          FILE_NAME       := Plot_Par_File_Name,
          HISTORY         := OLD,
          ACCESS_METHOD   := DIRECT,
          ERROR           := CONTINUE );

   RESET (Plotfile, ERROR := CONTINUE);
   FIND  (Plotfile, Area_Number);
   READ  (Plotfile, Current_Plot_Area);
   CLOSE (Plotfile);

   Area_Name := Current_Plot_Area.Area_Name;
   CURPOS_Cursor_Position (2,15);
   WRITELN
    (' Plot_Definition_Record = ', Area_Number);
   WRITELN (SW_File,' Plot_Definition_Record = ' ,
       Area_Number);
   WITH Current_Plot_Area DO
     BEGIN
       CURPOS_Cursor_Position (5,3);
       WRITELN ('area name: ', area_name);
       WRITELN (SW_File, 'area name: ', area_name);
       CURPOS_Cursor_Position (7,3);
       WRITELN ('transformation type: ', trans_type);
       WRITELN (SW_File,'transformation type: ',
                trans_type);
       CURPOS_Cursor_Position (10,3);
       WRITELN ('Ground Index Point (X): ', X_Index:5:2);
       WRITELN (SW_File,'Ground Index Point (X): ',
                X_Index:5:2);
       CURPOS_Cursor_Position (11,3);
       WRITELN ('Ground Index Point (Y): ', Y_Index:5:2);
       WRITELN (SW_File,'Ground Index Point (Y): ',
                Y_Index:5:2);
       CURPOS_Cursor_Position (12,3);
       WRITELN ('Ground Index Point (Z): ', Z_Index:5:2);
       WRITELN (SW_File, 'Ground Index Point (Z): ',
                Z_Index:5:2);
       CURPOS_Cursor_Position (14,3);
       WRITELN ('View Direction from south (degrees):  ',
                 View_Direction:5:2);
       WRITELN
         (SW_File,'View Direction from south (degrees):  ',
          View_Direction:5:2);
```

```
      CURPOS_Cursor_Position (24,40);
      WRITELN (' RETURN TO Continue...');
      READLN;
      ERASCR_Erase_Screen;
   END; {of with}

   GPCGTT_Geo_Program_Compute_Ground_To_Trans_Transformation
                                 (Current_Plot_Area, gtt_mat);
END; {* CPAM_Change_Plot_Area_Modify *}
```

```
{**********************************************************}
PROCEDURE Change_Mode ;
VAR
  k : INTEGER;

  BEGIN
    IF Check_String = 'CHANGE TO ' THEN
      BEGIN
        Look_Forward := TRUE;
        CASE Plot_String[11] OF
          'P'  : BEGIN
                   IF Change_To ( 'PLOT AREA' ) THEN
                     BEGIN
                       j := 20;
                       GV_Get_Value (j, Plot_String, i, r,
                                     Error);
                       WRITELN
                       ('found line that contains: CHANGE' +
                        'TO PLOT AREA ',i, crlf);
                       WRITELN
                      (SW_File,'found line that contains:' +
                      'CHANGE TO PLOT AREA ',i, crlf);
                       IF i <> First_PA THEN
                         BEGIN
                           WRITELN
                          ('Found new PLOT AREA number: ',i,
                            crlf);
                           WRITELN (SW_File,'Found new' +
                            'PLOT AREA number: ', crlf);
                           WRITELN
                           ('Do you want TO change PLOT' +
                             'AREA? (Y/N)', crlf);
                           WRITELN (SW_File,'Do you want' +
                               'to change PLOT AREA? (Y/N)',
                                crlf);
                           READLN (response);
                           WRITELN (SW_File, response);
                           IF response IN ['Y','y'] THEN
                             BEGIN
                               First_PA := I;
                           CPAM_Change_Plot_Area_Modify (I);
                             END;
                     END;
                     END;
                   IF Change_To ( 'PEN NUMBER' ) THEN
                     BEGIN
                       J := 21;
                       GV_Get_Value (j, Plot_String, i, r,
                                     Error);
                       WRITELN
                        ('found line that contains:' +
                         'CHANGE TO PEN NUMBER ', i,crlf);
                       WRITELN
```

```
          (SW_File,'found line that contains:' +
          ' CHANGE TO PEN NUMBER ', i,crlf);
          Pen_num_BUF := i;
         END;
       IF Change_To ( 'PEN UP' ) THEN
         BEGIN
           WRITELN ('found line that contains: +
                    'CHANGE TO PENUP', crlf);
           WRITELN (SW_File,'found line that' +
            'contains: CHANGE TO PENUP', crlf);
           Location:= terminate;
         END;
       IF Change_To ( 'PEN DOWN' ) THEN
         BEGIN
           ET_Element := ET_Line;
           Theme_Name := Line_type_BUF;
           IST_Initialize_And_Store_Theme;
           GS_Get_String;
         END;
     END;
'L'   : BEGIN
      IF Change_To ( 'LABEL' ) THEN
        BEGIN
         j := 16;
         GA_Get_Annotation (j, Plot_String,
                           Annotation, Error);
         FOR k:= 1 TO 20 DO
            Geo_Label [k] := Annotation[k];
         WRITELN ('found line that contains:' +
                  'LABEL: ',Geo_Label, crlf);
         WRITELN (SW_File,'found line that' +
                  'contains: LABEL: ',Geo_Label,
                  crlf);
        END;
      IF Change_To ( 'LINE TYPE' ) THEN
        BEGIN
          j := 20;
          GV_Get_Value (j, Plot_String, i, r,
                        Error);
          WRITELN
           ('found line that contains: CHANGE' +
            ' TO LINE TYPE ', i,crlf);
          WRITELN
          (SW_File,'found line that contains:' +
           'CHANGE TO LINE TYPE ', i,crlf);
          k := 0;
          REPEAT
            k := k + 1;
          UNTIL (I = LT_Store[k].LT_Number) OR
                (k = Max_Array);
          IF (k = Max_Array) AND
             (i <> LT_Store[k].LT_Number) THEN
            BEGIN
```

```
                    WRITELN
                    ('Did not find your Line Type...',
                      crlf);
                    WRITELN
                    (SW_File,'Did not find your Line'+
                    'Type...', crlf);
                  END;
                GA_Get_Annotation (j, Plot_String,
                                     Annotation, Error);
                CS_Capitalise_String (Annotation);
                Slet_Blanks (Annotation,
                             Varying_Label);
                IF Varying_Label = LT_Store[k].LT_Text
                   THEN BEGIN
                    WRITELN ( crlf);
                    WRITELN ('MATCH....', crlf);
                    WRITELN ('Line Annotation: ',
                          LT_Store[k].LT_Text, crlf);
                    WRITELN(crlf);
                    WRITELN (SW_File, crlf);
                    WRITELN
                      (SW_File, 'MATCH....', crlf);
                    WRITELN
                      (SW_File, 'Line Annotation: ',
                       LT_Store[k].LT_Text, crlf);
                    WRITELN (SW_File, crlf);
                   END
                ELSE
                   BEGIN
                     WRITELN(crlf);
                     WRITELN ('MISMATCH....', crlf);
                     WRITELN ('FOUND: ',
                               Varying_Label, crlf);
                     WRITELN ('              ', crlf);
                     WRITELN (' USED: ',
                         LT_Store[k].LT_Text, crlf);
                     WRITELN (crlf);
                     WRITELN (SW_File, 'MISMATCH....',
                              crlf);
                     WRITELN (SW_File, 'FOUND: ',
                              Varying_Label, crlf);
                     WRITELN (SW_File, '             ',
                              crlf);
                     WRITELN (SW_File, ' USED: ',
                           LT_Store[k].LT_Text, crlf);
                   END;
                Line_type_BUF := LT_Store[k].LT_Text;
              END;
          END;
'C'   : BEGIN
            IF Change_To ( 'CONT LINE ON' ) BEGIN
              BEGIN
                WRITELN
```

```
            ('found line that contains: CHANGE' +
             ' TO CONT LINE ON', crlf);
            WRITELN
            (SW_File,'found line that contains:' +
            ' CHANGE TO CONT LINE ON', crlf);
            Cont_line_BUF := TRUE;
          END;
        IF Change_To ( 'CONT LINE OFF' ) THEN
          BEGIN
            WRITELN
            ('found line that contains: CHANGE' +
            ' TO CONT LINE OFF', crlf);
            WRITELN
            (SW_File,'found line that contains:' +
            ' CHANGE TO CONT LINE OFF', crlf);
            Cont_line_BUF := FALSE;
          END;
      END
'S'   : BEGIN
        IF Change_To ( 'SLICK LINE ON' ) THEN
          BEGIN
            WRITELN
             ('found line that contains: CHANGE' +
              ' TO SLICK LINE ON', crlf);
            WRITELN
            (SW_File,'found line that contains:' +
            ' TO CHANGE TO SLICK LINE ON', crlf);
            Slick_line_BUF := TRUE;
          END;
        IF Change_To ( 'SLICK LINE OFF' ) THEN
          BEGIN
            WRITELN
            ('found line that contains: CHANGE' +
            ' TO SLICK LINE OFF', crlf);
            WRITELN
            (SW_File,'found line that contains:' +
            ' TO CHANGE TO SLICK LINE OFF', crlf);
            Slick_line_BUF := FALSE;
          END ;
        IF Change_To ( 'SYMBOL TYPE' ) THEN
          BEGIN
            j := 22;
            GV_Get_Value (j, Plot_String, i, r,
                          Error);
            WRITELN
             ('found line that contains: CHANGE' +
             ' TO SYMBOL TYPE ',I, crlf);
            WRITELN
             (SW_File,'found line that contains:'+
             ' CHANGE TO SYMBOL TYPE ',I, crlf);
            k := 0;
            REPEAT
              k := k + 1;
```

```
                UNTIL (i = ST_Store[k].ST_Number) OR
                      (k = Max_Array);
                IF (k = Max_Array) AND
                   (i <> ST_Store[k].ST_number) THEN
                  BEGIN
                    WRITELN ('Did not find your' +
                             'Symbol Type...', crlf);
                    WRITELN (SW_File,'Did not find' +
                         'your Symbol Type...', crlf);
                  END;
                GA_Get_Annotation (j, Plot_String,
                                   Annotation, Error);
                CS_Capitalise_String (Annotation);
                Slet_Blanks (Annotation,
                             Varying_Label);
                IF Varying_Label = ST_Store[k].ST_Text
                  THEN BEGIN
                   WRITELN (crlf);
                   WRITELN ('MATCH....', crlf);
                   WRITELN ('Symbol Annotation: ',
                           ST_Store[k].ST_Text, crlf);
                   WRITELN (crlf);
                   WRITELN (SW_File,'MATCH....',
                            crlf);
                   WRITELN (SW_File,'Symbol' +
                            'Annotation: ',
                           ST_Store[k].ST_Text, crlf);
                  END
                ELSE
                  BEGIN
                    WRITELN (crlf);
                    WRITELN ('MISMATCH....', crlf);
                    WRITELN ('FOUND: ', Varying_Label,
                              crlf);
                    WRITELN ('             ', crlf);
                    WRITELN (' USED: ',
                    ST_Store[k].ST_Text, crlf);
                    WRITELN (crlf);
                    WRITELN (SW_File,'MISMATCH....',
                             crlf);
                    WRITELN (SW_File,'FOUND: ',
                             Varying_Label, crlf);
                    WRITELN (SW_File,'             ',
                             crlf);
                    WRITELN (SW_File,' USED: ',
                           ST_Store[k].ST_Text, crlf);
                  END;
                Sym_type_BUF := ST_Store[k].ST_Text;
              END ;
          END;
'O'   : BEGIN
            IF Change_To ( 'OBJECT' ) THEN
              BEGIN
                ET_Element := ET_Point;
```

```
                Theme_Name := Sym_type_BUF;
                IST_Initialize_And_Store_Theme;
                j := 17;
                GA_Get_Annotation (j, Plot_String,
                                   Annotation, Error);
                Slet_Blanks (Annotation,
                             Varying_Label);
                WRITELN
                 ('found line that contains: CHANGE' +
                 ' TO OBJECT ', Varying_Label, crlf);
                WRITELN
                 (SW_File,'found line that contains:'+
                 ' CHANGE TO OBJECT ', Varying_Label,
                  crlf);
                WRITELN ('Object notation: ',
                  Varying_Label, crlf);
                WRITELN (SW_File,'Object notation: ',
                         Varying_Label, crlf);
                 IF (Varying_Label = 'DL') OR
                    (Varying_Label = 'DV') THEN
                    BEGIN
                      WRITELN ('Strike and Dip: ',
                               SD_Buf, crlf);
                      WRITELN (SW_File,'Strike and
                              Dip: ', SD_Buf, crlf);
                    END;
                 IF (Varying_Label = 'PL') OR
                    (Varying_Label = 'PV') THEN
                    BEGIN
                      WRITELN ('Plunge and Trend: ',
                               SD_Buf, crlf);
                      WRITELN (SW_File,'Plunge and' +
                            ' Trend: ', SD_Buf, crlf);
                    END;
                 GV_Get_Value (j, Plot_String, i, r,
                               Error);
                 WRITELN ('Symbol Rotation: ', r:6:4,
                          crlf);
                 WRITELN (SW_File,'Symbol Rotation: ',
                         r:6:4, crlf);
                 GA_Get_Annotation (j, Plot_String,
                                    Annotation, Error);
                 Slet_Blanks (Annotation,
                              Varying_Label);
                 WRITELN ('object label: ',
                          Varying_Label, crlf);
                 WRITELN (SW_File,'object label: ',
                         Varying_Label, crlf);
                 GN_Get_Node;
               END;
          END;
  'A'   : BEGIN
            IF Change_To ( 'ATTITUDE' ) THEN
              BEGIN
```

```
                      WRITELN
                     ('found line that contains: CHANGE' +
                     ' TO ATTITUDE', crlf);
                      WRITELN
                     (SW_File,'found line that contains:' +
                     ' CHANGE TO ATTITUDE', crlf);
                      j := 18;
                      GV_Get_Value (j, Plot_String, i, r,
                                    Error);
                      Rotation_Buf := r;
                      WRITELN ('Symbol Rotation: ', r:6:4,
                                crlf);
                      WRITELN (SW_File,'Symbol Rotation: ',
                               r:6:4, crlf);
                      GA_Get_Annotation (j, Plot_String,
                                         Annotation, Error);
                      Slet_Blanks (Annotation,
                                   Varying_Label);
                      SD_Buf := Varying_Label;
                    END;
                 END;
           OTHERWISE;
           END;
         END;
END; {* Change_Mode *}
```

```
{************************************************************}
PROCEDURE RLTA_Read_Line_Type_Array;
VAR
  k : INTEGER;

BEGIN
   OPEN (FILE_VARIABLE   := Line_Types,
         FILE_NAME       := 'Line_Types.table',
         HISTORY         := OLD,
         ACCESS_METHOD   := SEQUENTIAL,
         ERROR           := CONTINUE );
   RESET (Line_Types, Error := CONTINUE) ;

   k := 0;
   WHILE NOT EOF (Line_Types) DO
     BEGIN
       k := k + 1;
       READLN (Line_Types, Read_File_String);
       j := 0;
       GV_Get_Value (j, Read_File_String, Line_Type_Number,
                     r, Error);
       LT_Store[k].LT_Number := Line_Type_Number;
       GA_Get_Annotation (j, Read_File_String,
                          Line_Type_Text, Error);
       CS_Capitalise_String (Line_Type_Text);
       Slet_Blanks (Line_Type_Text, Varying_Label);
       LT_Store[k].LT_Text := Varying_Label;
     END;
   CLOSE (Line_Types);
END; {* RLTA_Read_Line_Type_Array *}
```

```
{*************************************************************}
PROCEDURE RSTA_Read_Symbol_Type_Array;
VAR
  k : INTEGER;

BEGIN
   OPEN (FILE_VARIABLE   := Symbol_Types,
         FILE_NAME       := 'Symbol_Types.table',
         HISTORY         := OLD,
         ACCESS_METHOD   := SEQUENTIAL,
         ERROR           := CONTINUE );

   RESET (Symbol_Types, Error := CONTINUE) ;

   k := 0;
   WHILE NOT EOF (Symbol_Types) DO
     BEGIN
       k := k + 1;
       READLN (Symbol_Types, Read_File_String );
       j := 0;
       GV_Get_Value (j, Read_File_String,
                     Symbol_Type-Number, r, Error);
       ST_Store[k].ST_Number := Symbol_Type_Number;
       GA_Get_Annotation (j, Read_File_String,
                          Symbol_Type_Text, Error);
       CS_Capitalise_String (Symbol_Type_Text);
       Slet_Blanks (Symbol_Type_Text, Varying_Label);
       ST_Store[k].ST_Text := Varying_Label;
     END;
   CLOSE (Symbol_Types);
END; {* RSTA_Read_Symbol_Type_Array *}


{*************************************************************}
PROCEDURE OCLT_Open_and_Check_Line_Types;
VAR
   Retry    : BOOLEAN;
   Close_it : BOOLEAN;

BEGIN
   Close_it := FALSE;
   OK       := FALSE;
   Retry    := FALSE;
   ERASCR_Erase_Screen;
   SELGRA_Select_Graphics ('B');
   CURPOS_Cursor_Position (20,20);
   SELGRA_Select_Graphics ('O');

   OPEN (FILE_VARIABLE   := Line_Types,
         FILE_NAME       := 'Line_Types.table',
         HISTORY         := OLD,
         ACCESS_METHOD   := SEQUENTIAL,
         ERROR           := CONTINUE );
```

```
   RESET (Line_Types, Error := CONTINUE) ;

   CASE STATUS (Line_Types) OF
      -1    :  BEGIN
                 CURPOS_Cursor_Position (20,20);
                 WRITELN ('FILE IS EMPTY...CANNOT CONTINUE'+
                          ' WITH TRANSLATION' , crlf);
                 CURPOS_Cursor_Position (24,50);
                 WRITELN ('RETURN TO Continue...' );
                 OK := FALSE;
                 Close_it := TRUE;
               END;
       0    :  BEGIN
                 CURPOS_Cursor_Position (20,20);
                 OK := TRUE;
                 Close_it := TRUE;
               END;
       OTHERWISE
               BEGIN
                 CURPOS_Cursor_Position (20,20);
                 WRITELN ('FILE NOT FOUND:::::>  ' +
                          ' Line_Types.table', crlf);
                 CURPOS_Cursor_Position (24,50);
                 WRITELN ('RETURN TO Continue...' );
                 OK := FALSE;
                 Close_it := FALSE;
               END;
   END; {of case}
   IF Close_it THEN
      Close (Line_Types);
END; {* OCLT_Open_and_Check_Line_Types *}
```

```
{*************************************************************}
PROCEDURE OCST_Open_and_Check_Symbol_Types;

VAR
   Retry    : BOOLEAN;
   Close_it : BOOLEAN;

BEGIN
   Close_it := FALSE;
   OK       := FALSE;  {global for continue}
   Retry    := FALSE;
   ERASCR_Erase_Screen;
   SELGRA_Select_Graphics ('B');
   CURPOS_Cursor_Position (20,20);
   SELGRA_Select_Graphics ('O');

   OPEN (FILE_VARIABLE    := Symbol_Types,
         FILE_NAME        := 'Symbol_Types.table',
         HISTORY          := OLD,
         ACCESS_METHOD    := SEQUENTIAL,
         ERROR            := CONTINUE );

   RESET (Symbol_Types, Error := CONTINUE) ;

   CASE STATUS (Symbol_Types) OF
      -1    :  BEGIN
                 CURPOS_Cursor_Position (20,20);
                 WRITELN ('FILE IS EMPTY...CANNOT CONTINUE'+
                          ' WITH TRANSLATION' , crlf);
                 CURPOS_Cursor_Position (24,50);
                 WRITELN ('RETURN TO Continue...' );
                 OK := FALSE;
                 Close_it := TRUE;
               END;
       0    :  BEGIN
                 CURPOS_Cursor_Position (20,20);
                 OK := TRUE;
                 Close_it := TRUE;
               END;
       OTHERWISE
               BEGIN
                 CURPOS_Cursor_Position (20,20);
                 WRITELN ('FILE NOT FOUND:::::>   ' +
                          ' Symbol_Types.table', crlf);
                 CURPOS_Cursor_Position (24,50);
                 WRITELN ('RETURN TO Continue...' );
                 OK := FALSE;
                 Close_it := FALSE;
               END;
   END; {of case}
   IF Close_it THEN
      Close (Symbol_Types);
END; {* OCST_Open_and_Check_Symbol_Types *}
```

```
{**********************************************************}
PROCEDURE SKP_Setup_KGIS_Parameters;

BEGIN
   WITH coll_parm DO
      BEGIN
        node_snap_dist := 30.0;
        { 3*sd, here 30/1000 of a foot }
        edge_snap_dist := 30.0 ;
        extend_dist    := 30.0 ;
        peel_dist      := 30.0 ;
        deviation_dist :=  0.0000001 ;
        trim_dist      := 30.0 ;
      END;

   map_extents (Xlow, Ylow, Xhigh, Yhigh);
   world_TO_range (METERS, Xlow, Xhigh, Ylow, Yhigh,
                   db_range);
   display_init (db_range);
   top_window_init (db_range);
   build_tdisp;
END; {* SKP_Setup_KGIS_Parameters *}
```

```
{***********************************************************}
PROCEDURE OCG_Open_and_Check_GeoProgram_File;
VAR
   Retry    : BOOLEAN;
   Close_it : BOOLEAN;

BEGIN
   REPEAT
     Close_it := FALSE;
     OK       := FALSE;
     Retry    := FALSE;

     ERASCR_Erase_Screen;
     CURPOS_Cursor_Position (5,10);
     WRITELN
      ('Enter geofile name for translation TO KGIS: ');
     WRITELN
    (SW_File,'Enter geofile name for translation TO KGIS: ');
     READLN (GEOP_File_Name);
     WRITELN (SW_File,GEOP_File_Name);
     CS_Capitalise_String (GEOP_File_Name);
     Slet_Blanks (GEOP_File_Name, Varying_Label2);
     SELGRA_Select_Graphics ('B');
     CURPOS_Cursor_Position (20,20);
     WRITELN ('PLEASE WAIT.....');
     SELGRA_Select_Graphics ('O');

     OPEN ( FILE_VARIABLE    :=  output_file_data,
            FILE_NAME        :=  Varying_Label2,
            HISTORY          :=  OLD,
            ERROR            :=  CONTINUE );

     RESET ( Output_File_Data, ERROR := CONTINUE );

     CASE STATUS (Output_File_Data) OF
        -1    :  BEGIN
                   CURPOS_Cursor_Position (20,20);
                   WRITELN ('FILE IS EMPTY', crlf);
                   WRITELN ('Would you like TO enter' +
                          ' GEOPROGRAM name again? (Y/N) ');
                   READLN (response);
                   IF response IN ['Y', 'y'] THEN
                     BEGIN
                       Retry := TRUE;
                       Close_it := TRUE;
                     END
                   ELSE
                     BEGIN
                       Retry := FALSE;
                       OK := FALSE;
                       Close_it := TRUE;
                     END;
               END;
         0    :  BEGIN
```

```
                 CURPOS_Cursor_Position (20,20);
                 WRITELN ('GeoProgram File,OK ');
                 WRITELN (SW_File,'GeoProgram File,OK ');
                 Retry := FALSE;
                 OK := TRUE;
                 Close_it := TRUE;
               END;
       OTHERWISE
               BEGIN
                 CURPOS_Cursor_Position (20,20);
                 WRITELN ('FILE NOT FOUND:::::>  ' +
                         Varying_Label2 , crlf);
                 WRITELN ('Would you like TO enter' +
                        ' GEOPROGRAM name again? (Y/N) ');
                 READLN (response);
                 IF response IN ['Y', 'y'] THEN
                    BEGIN
                      Retry := TRUE;
                      OK := FALSE;
                    END
                 ELSE
                    BEGIN
                      Retry := FALSE;
                      OK := FALSE;
                      Close_it := FALSE;
                    END;
               END;
    END; {of case}
   UNTIL NOT Retry;
   IF Close_it THEN
      Close (Output_File_Data);
   IF OK THEN
     BEGIN
       Input_File_Name := Varying_Label2;

    OPEN ( FILE_VARIABLE   := Output_File_Data,
           FILE_NAME       := Input_File_Name,
           HISTORY         := OLD,
           ACCESS_METHOD   := Direct,
           ERROR           := CONTINUE );

    RESET (Output_File_Data, ERROR := CONTINUE );

        Output_File_Pointer := 1;
        FIND (Output_File_Data, Output_File_Pointer);
        READ (Output_File_Data, Logical_Record);
        Logical_Record_Pointer:= 1;
  END;
END; {* OCG_Open_and_Check_GeoProgram_File *}
```

```
{***********************************************************}
PROCEDURE RIKE_Request_if_KGIS_Edit;

BEGIN
   For i := 1 TO Clean_Scr DO
      WRITELN (crlf);
   WRITELN ('Do You Want to goto KGIS edit? (Y/N)');
   READLN (Response);
   IF (Response IN ['y', 'Y']) THEN
      BEGIN
         keditdrv (input, output);
      END;
END; {* RIKE_Request_if_KGIS_Edit *}


{***********************************************************}
PROCEDURE Get_First_Plot_Area;

VAR
   i, j : INTEGER;

BEGIN
  REPEAT
    FIOF_Forward_In_Output_File (Plot_String, Nchar);
    FOR i := 1 TO 10 DO
    Check_String [i] := Plot_String [i] ;
    IF Check_String = 'QUIT      ' THEN Finished:= TRUE;
  UNTIL ((Check_String = 'CHANGE TO ') AND
         (Plot_String [11] = 'P')      AND
         (Change_TO ('PLOT AREA'))) OR Finished;
  IF finished THEN
     BEGIN
       OK := FALSE;
       ERASCR_Erase_Screen;
       WRITELN (' Finished found ');
       WRITELN (SW_File,' Finished found ');
       READLN;
     END
  ELSE
     BEGIN
       j := 20;
       GV_Get_Value (j, Plot_String, i, r, Error);
       First_PA := i;
       CURPOS_Cursor_Position (16,3);
       WRITELN ('Initial Plot Area :  ', First_PA:1);
       WRITELN
           (SW_File,'Initial Plot Area :  ', First_PA:1);
       CPAM_Change_Plot_Area_Modify (First_PA);
     END;
   Finished := FALSE;

   IF OK THEN
      BEGIN
          Output_File_Pointer := 1;
```

```
          FIND (Output_File_Data, Output_File_Pointer);
          READ (Output_File_Data, Logical_Record);
          Logical_Record_Pointer:= 1;

         REPEAT
           FIOF_Forward_In_Output_File (Plot_String, Nchar);
           FOR i := 1 TO 10 DO
             Check_String [i] := Plot_String [i] ;
           IF Check_String = 'QUIT      ' THEN
             Finished:= TRUE ;
         UNTIL (Check_String = 'START     ') OR Finished;
      END;
END; {* Get_First_Plot_Area *}
```

```
{*********************************************************}
PROCEDURE Get_Max_And_Min (VAR loc: locationType);
VAR
  k : INTEGER;

BEGIN
  NChar := 0;
  Look_Forward := TRUE;
  S_X_Max := 1.0;
  S_Y_Max := 1.0;
  S_X_Min := 10000.0;
  S_Y_Min := 10000.0;
  Minimax := TRUE;
  ERASCR_Erase_Screen;
  SELGRA_Select_Graphics ('B');
  DBWIDT_Double_Width (2);
  CURPOS_Cursor_Position (2,12);
  WRITELN ('SEARCHING...');
  SELASC_Select_ASCII;
  SELGRA_Select_Graphics ('0');
  IC_INVISIBLE_CURSOR;
  REPEAT
    IF Look_Forward THEN
       BEGIN
         FIOF_Forward_In_Output_File (Plot_String, Nchar);
         FOR i:= 1 TO 10 DO
            Check_String [i] := Plot_String [i];
       END;
       CASE Check_String [1] OF
          'C' : BEGIN
                END ;
          ' ' : BEGIN
                  j := 1;
                  GV_Get_Value
                  (j, Plot_String, i, XYZ_Measured [1],
                   Error) ;
                  GV_Get_Value
                  (j, Plot_String, i, XYZ_Measured [2],
                   Error) ;
                  GV_Get_Value
                  (j, Plot_String, i, XYZ_Measured [3],
                   Error) ;
                  Translate_Coordinates (XYZ_Measured,
                   Error, loc);
                END ;
          'Q' : BEGIN
                  IF Check_String  = 'QUIT
                    THEN Finished := TRUE ;
                END ;
           OTHERWISE ;
       END ; {of case}
  UNTIL Finished;
  Finished := false;
  ERASCR_Erase_Screen;
```

```
  VC_Visible_Cursor;
  DBWIDT_Double_Width (2);
  CURPOS_Cursor_Position (2,7);
  WRITELN ('FINAL MAX AND MIN VALUES');
  WRITELN (SW_File, 'FINAL MAX AND MIN VALUES');
  CURPOS_Cursor_Position (5,8);
  WRITELN
  ('X MAX: ',S_X_Max:7:3, crlf);
  WRITELN
  (SW_File,'X MAX: ',S_X_Max:7:3, crlf);
  CURPOS_Cursor_Position (5,45);
  WRITELN (' X MIN: ',S_X_Min:7:3, crlf);
  WRITELN (SW_File,' X MIN: ',S_X_Min:7:3, crlf);
  CURPOS_Cursor_Position (8,8);
  WRITELN ('Y MAX: ',S_Y_Max:7:3, crlf);
  WRITELN (SW_File,'Y MAX: ',S_Y_Max:7:3, crlf);
  CURPOS_Cursor_Position (8,45);
  WRITELN (' Y MIN: ',S_Y_Min:7:3, crlf);
  WRITELN (SW_File,' Y MIN: ',S_Y_Min:7:3, crlf);
  CURPOS_Cursor_Position (24,50);
  WRITELN ('RETURN TO Continue...' );
  READLN;
  IF OK THEN
      BEGIN
          RESET (Output_File_Data, ERROR := CONTINUE );
          Output_File_Pointer := 1;
          FIND (Output_File_Data, Output_File_Pointer);
          READ (Output_File_Data, Logical_Record);
          Logical_Record_Pointer:= 1;

          REPEAT
            FIOF_Forward_In_Output_File
                                ( Plot_String, Nchar ) ;
            FOR i := 1 TO 10 DO
               Check_String [i] := Plot_String [i];
            IF Check_String = 'QUIT      ' THEN
               Finished := TRUE ;
          UNTIL (Check_String = 'START     ') OR Finished;
      END;
END; {* Get_Max_And_Min *}
```

```
{**********************************************************}
PROCEDURE TCF_Trans_Coord_File;

BEGIN
  CS_Capitalise_String (K_Name);
  Slet_Blanks (K_Name, Varying_Label2);
  SCREEN_WRITE_FILE := (Varying_Label2 + '.SWF');

  OPEN (FILE_VARIABLE := SW_File,
        FILE_NAME     := screen_write_file ,
        HISTORY       := NEW,
        ERROR         := CONTINUE);

        REWRITE (SW_File, ERROR := CONTINUE);
END; {* TCF_Trans_Coord_File *}


{**********************************************************}
PROCEDURE TP_Transform_Param;

BEGIN
  RLTA_Read_Line_Type_Array;
  RSTA_Read_Symbol_Type_Array;
  Get_First_Plot_Area;
END; {* TP_Transform_Param *}
```

```
{**********************************************************}
PROCEDURE TGTK_Translate_GeoFile_To_KGIS;
VAR
  k : INTEGER;

BEGIN
  Pause         := FALSE;
  Location      := terminate;
  Husk_Area     := 0;
  NChar         := 0;
  Count         := 0.0 ;
  Look_Forward := TRUE;
  Minimax       := FALSE;

  IF db_opened (Kdb_Name, READ_WRITE, Err_Code) THEN
    BEGIN
       ERASCR_Erase_Screen;
       CURPOS_Cursor_Position (24,10);
       WRITELN ('Starting Geo File spacial Data ' +
                'Translation.....', crlf);
       WRITELN (SW_File,'Starting Geo File spacial' +
               ' Data Translation.....' , crlf);
       SKP_Setup_KGIS_Parameters;
       REPEAT
         IF Look_Forward THEN
           BEGIN
             FIOF_Forward_In_Output_File ( Plot_String,
                                           Nchar );
             FOR i:= 1 TO 10 DO
                Check_String [i] := Plot_String [i];
           END;
           CASE Check_String [1] OF
              'C' : BEGIN
                      Change_Mode;
                    END;
              ' ' : BEGIN
                       J := 1;
                       GV_Get_Value
                       (j, Plot_String, i, Point_Buffer[1],
                        Error) ;
                       GV_Get_Value
                       (j, Plot_String, i, Point_Buffer[2],
                        Error) ;
                       GV_Get_Value
                       (j, Plot_String, i, Point_Buffer[3],
                        Error) ;
                    END ;
              'Q' : BEGIN
                      IF Check_String = 'QUIT      ' THEN
                        Finished := TRUE;
                    END;
                    OTHERWISE;
           END; {of case}
       UNTIL Finished;
```

```
        display_END;
        IF NOT db_closed (Err_Code) THEN
          BEGIN
            WRITELN (Err_Code , crlf);
            WRITELN (SW_File, Err_Code , crlf);
          END;
    END;
END; {* TGTK_Translate_GeoFile_To_KGIS *}
```

```
{***********************************************************}
PROCEDURE Screen_1;
VAR
   i : INTEGER;

BEGIN
   ERASCR_Erase_Screen;
   IC_Invisible_cursor;
   FOR i := 1 TO 24 DO
      DBWIDT_Double_Width (i);
   SELGRA_Select_Graphics ('O');
   CURPOS_Cursor_Position (4,13);
   WRITELN ('WELCOME TO');
   CURPOS_Cursor_Position (7,13);
   WRITELN ('GEO_TRANS:');
   CURPOS_Cursor_Position (9,6);
   WRITELN ('A TRANSLATION PROGRAM FROM');
   CURPOS_Cursor_Position (11,6);
   WRITELN ('GEOPROGRAM FILES');
   CURPOS_Cursor_Position (13,6);
   WRITELN ('TO ORACLE RDB AND KORK');
   CURPOS_Cursor_Position (15,6);
   WRITELN ('GEOGRAPHIC INFORMATION SYSTEM ');
   CURPOS_Cursor_Position (19,10);
   WRITELN ('BY Steve Schilling');
   SELSGR_Select_Special_Graphics;
   IC_Invisible_Cursor;
   CURPOS_Cursor_Position (1,1);
   WRITELN ('l');
   FOR i := 2 TO 39 DO
      BEGIN
         CURPOS_Cursor_Position (1,i);
         WRITELN ('q');
      END;
   CURPOS_Cursor_Position (1,40);
   WRITELN ('k');
   FOR i := 2 TO 23 DO
      BEGIN
         CURPOS_Cursor_Position (i,40);
         WRITELN ('x');
      END;
   CURPOS_Cursor_Position (24,40);
   WRITELN ('j');
   FOR i := 39 DOWNTO 2 DO
      BEGIN
         CURPOS_Cursor_Position (24,i);
         WRITELN ('q');
      END;
   CURPOS_Cursor_Position (24,1);
   WRITELN ('m');
   FOR i := 23 DOWNTO 2 DO
   BEGIN
      CURPOS_Cursor_Position (i,1);
      WRITELN ('x');
```

```
   END;
   SELGRA_Select_Graphics ('B');
   CURPOS_Cursor_Position (22,9);
   WRITELN (' RETURN TO CONTINUE.....');
   SELASC_Select_ASCII;
   READLN;
   SIWIDT_Single_Width (24);
   BELLRI_Bell_Ring;
   VC_Visible_Cursor;
   SELGRA_Select_Graphics ('O');
END; {* Screen_1 *}


{*************************************************************}
PROCEDURE Menu_1;

CONST
   S_Top    = 1;
   S_Bottom = 23;
   S_Left   = 1;
   S_Right  = 79;

VAR
   Direktion   : CHAR;
   Grafiks     : CHAR;
   i, j        : INTEGER;

BEGIN
   ERASCR_Erase_Screen;
   Direktion := 'U';
   CURPOS_Cursor_Position (S_Top,S_Left);
   Grafiks := 'R';
   SELGRA_Select_Graphics (Grafiks);
   FOR i := S_Top TO S_Bottom DO
      BEGIN
       CURPOS_Cursor_Position (i,1);
       WRITELN
('
 crlf);
      END;
   CURPOS_Cursor_Position (S_Top,1);
       WRITELN
('*********************************************************',
 crlf);

   CURPOS_Cursor_Position (4,20);
   WRITELN ('____________________________________', crlf);
   CURPOS_Cursor_Position (5,35);
   WRITELN ('M E N U', crlf);
   CURPOS_Cursor_Position (7,20);
   WRITELN ('TRANSLATE TO ORACLE RDBMS     :   1 ', crlf);
   CURPOS_Cursor_Position (9,20);
   WRITELN ('TRANSLATE TO KGIS             :   2 ', crlf);
   CURPOS_Cursor_Position (11,20);
```

```
   WRITELN ('QUIT                                     :   3 ', crlf);
   CURPOS_Cursor_Position (12,20);
   WRITELN ('_____________________________________', crlf);
   CURPOS_Cursor_Position (S_BOTTOM,1);
       WRITELN
('***********************************************************',
crlf);
   IC_Invisible_cursor;
   CURPOS_Cursor_Position (15,47);
   WRITELN ('CHOICE:    ');
   READLN (I_Response);
   Grafiks := 'O';
   SELGRA_Select_Graphics (Grafiks);
   CURPOS_Cursor_Position (S_Bottom,S_Right);
   VC_VISIBLE_CURSOR;
   ERASCR_Erase_Screen;
END; {* Menu_1 *}


{************************************************************}
PROCEDURE OCKD_Open_and_Check_KGIS_Database;
VAR
   Retry : BOOLEAN;

BEGIN
   REPEAT
      ERASCR_Erase_Screen;
      CURPOS_Cursor_Position (5,10);
      WRITELN ('Enter the KGIS database name: ');
      READLN (Kdb_Name);
      K_Name := Kdb_Name;
      TCF_Trans_Coord_File;
      WRITELN (SW_File,'Enter the KGIS database name: ');
      WRITELN (SW_File,Kdb_Name);
      OK := FALSE;
      Retry := FALSE;
      CURPOS_Cursor_Position (20,20);
      SELGRA_Select_Graphics ('B');
      WRITELN ('PLEASE WAIT.....');
      SELGRA_Select_Graphics ('O');

      IF db_opened ( Kdb_Name, READ_WRITE, Err_Code ) THEN
         BEGIN
           IF NOT db_closed ( Err_Code ) THEN
             BEGIN
               WRITELN (Err_Code , crlf);
               WRITELN (SW_File, Err_Code , crlf);
               OK := FALSE;
               Retry := FALSE;
             END
           ELSE
             BEGIN
               CURPOS_Cursor_Position (20,20);
               WRITELN ('KGIS Database, OK ');
```

```
              WRITELN (SW_File,'KGIS Database, OK ');
              OK := TRUE;
              Retry := FALSE;
            END;
        END
     ELSE
        BEGIN
          ERASCR_Erase_Screen;
          CURPOS_Cursor_Position (5,1);
          WRITELN ('Could not find KGIS database : ',
                    Kdb_Name, crlf);
          WRITELN ('Would you like TO enter file name' +
                   ' again ? (Y/N) ');
          READLN (Response);
          WRITELN (SW_File,'Could not find KGIS ' +
                   'database : ', Kdb_Name, crlf);
          WRITELN (SW_File,'Would you like TO enter file'+
                   'name again ? (Y/N) ');
          WRITELN (SW_File,Response);
          IF response IN ['Y', 'y'] THEN
            Retry := TRUE
          ELSE
            BEGIN
              Retry := FALSE;
              OK    := FALSE;
            END;
        END;
   UNTIL NOT Retry;
END; {* OCKD_Open_and_Check_KGIS_Database *}


{***********************************************************}
PROCEDURE FL_File_lister (which_file : string_20);
VAR
   Close_It : BOOLEAN;

BEGIN
   CS_Capitalise_String (Which_File);
   Slet_Blanks (Which_File, Varying_Label2);

   OPEN ( FILE_VARIABLE   :=  Output_File_Data,
          FILE_NAME       :=  Varying_Label2,
          HISTORY         :=  OLD,
          ERROR           :=  CONTINUE );

   RESET ( Output_File_Data, ERROR := CONTINUE );

   CASE STATUS (Output_File_Data) OF
      -1    :  BEGIN
                 WRITELN ('FILE IS EMPTY');
                 WRITELN ('...Retry ? (Y/N) ');
                 READLN (Response);
                 IF Response IN ['Y', 'y'] THEN
                    Retry := TRUE
```

```
                ELSE
                   Retry := FALSE;
                OK := FALSE;
                Close_it := TRUE;
              END;
      0     :  BEGIN
                CURPOS_Cursor_Position (4,40);
                WRITELN ('OK, found file     :',
                          Varying_Label2, crlf);
                WRITELN (SW_File,'OK, found file      : ',
                         Varying_Label2, crlf);
                Retry := FALSE;
                OK := TRUE;
                Close_it := TRUE;
              END;
      OTHERWISE
              BEGIN
                WRITELN ('FILE NOT FOUND::::>   ' +
                          Varying_Label2 );
                WRITELN ('...Retry ? (Y/N) ');
                READLN (Response);
                IF Response IN ['Y', 'y'] THEN
                   Retry := TRUE
                ELSE
                   Retry := FALSE;
                OK := FALSE;
                Close_it := FALSE;
              END;
   END; {of case}

   IF Close_it THEN
      Close (Output_File_Data);
END;  {* FL_File_Lister *}
{*************************************************************}
PROCEDURE Display_Files;
VAR
   i,j     : INTEGER;
   Grafiks : CHAR;

BEGIN
   ERASCR_Erase_Screen;
   Grafiks := 'R';
   SELGRA_Select_Graphics (Grafiks);
   FOR i := 1 TO 7 DO
      FOR j := 43 TO 80 DO
         BEGIN
           CURPOS_Cursor_Position (i,j);
           WRITELN (' ');
         END;
   CURPOS_Cursor_Position (1,55);
   WRITELN ('FILE STATUS :');
   WRITELN (SW_File,'FILE STATUS :');
   CURPOS_Cursor_Position (2,45);
   WRITELN ('KGIS Database, ',Kdb_Name,' : OK', crlf);
```

```
   WRITELN (SW_File,'KGIS Database, ',Kdb_Name,' : OK',
            crlf);
   CURPOS_Cursor_Position (3,45);
   WRITELN ('LEGEND File, ',Leg_Name,' : OK', crlf);
   WRITELN (SW_File,'LEGEND File, ',Leg_Name,' : OK', crlf);
   CURPOS_Cursor_Position (4,45);
   WRITELN ('GeoProgram File, ',Varying_Label2,' : OK',
             crlf);
   WRITELN (SW_File,'GeoProgram File, ',Varying_Label2,
            ' : OK', crlf);
   CURPOS_Cursor_Position (5,45);
   WRITELN ('Line Types File    : OK ', crlf);
   WRITELN (SW_File,'Line Types File    : OK ', crlf);
   CURPOS_Cursor_Position (6,45);
   WRITELN ('Symbol Types File  : OK ', crlf);
   WRITELN (SW_File,'Symbol Types File  : OK ', crlf);
   Grafiks := 'O';
   SELGRA_Select_Graphics (Grafiks);
END;  {* Display_FILES *}
```

```
{***********************************************************}
{|||||||||||>               MAIN               <||||||||||}
{***********************************************************}

BEGIN
   OPEN (OUTPUT, Carriage_Control := NONE,
                  Record_Length := 512);
   Dun := FALSE;
   Screen_1;
   REPEAT
      SETSCR_Set_Scroll (1,24);
      MENU_1;
      CASE I_Response OF
      1  :  BEGIN
              ERASCR_Erase_Screen;
              CURPOS_Cursor_Position (5,10);
              WRITELN
              ('Have you prepared an Oracle table?(y/n): ');
              READLN (Response) ;
              IF (Response IN ['y', 'Y']) THEN
                 BEGIN
                    PFOT_Planes_File_To_Oracle_Translator;
                 END
              ELSE
                 BEGIN
                 END;
            END;
      2  :  BEGIN
              OCKD_Open_and_Check_KGIS_Database;
              CURPOS_Cursor_Position (24,50);
              WRITELN ('RETURN TO Continue...' );
              READLN;
              IF OK THEN
                BEGIN
                  ERASCR_Erase_Screen;
                  CURPOS_Cursor_Position (5,10);
                  WRITELN ('Enter name of legend file:' +
                         '[',Kdb_Name,']  ');
                  WRITELN (SW_File,'Enter name of legend'+
                        'file: [',Kdb_Name,']  ');
                  READLN (Leg_Name);
                  WRITELN (SW_File, Leg_Name);
                  IF (Leg_Name = '') THEN
                    Leg_Name := Kdb_Name;
                  SELGRA_Select_Graphics ('0');
                  IC_Invisible_Cursor;
                  CURPOS_Cursor_Position (20,20);
                  WRITELN ('LEGEND File : OK');
                  WRITELN (SW_File,'LEGEND File : OK');
                  CURPOS_Cursor_Position (24,50);
                  WRITELN ('RETURN TO Continue...' );
                  READLN;
                END;
                ERASCR_Erase_Screen;
```

```
                IC_Invisible_Cursor;
                IF OK THEN
                  BEGIN
                    OCG_Open_and_Check_GeoProgram_File;
                    CURPOS_Cursor_Position (24,50);
                    WRITELN ('RETURN TO Continue...' );
                    READLN;
                  END;
                SELGRA_Select_Graphics ('O');
                IF OK THEN
                  BEGIN
                    OCLT_Open_and_Check_Line_Types;
                    OCST_Open_and_Check_Symbol_Types;
                    Display_FILES;
                    CURPOS_Cursor_Position (24,50);
                    WRITELN ('RETURN TO Continue...' );
                    READLN;
                  END;
                VC_Visible_Cursor;
                ERASCR_Erase_Screen;
                TP_Transform_Param;
                Get_Max_And_Min (Loc_Buf);
                IF OK THEN
                  BEGIN
                    TGTK_Translate_GeoFile_TO_KGIS;
                    Close ( Output_File_Data );
                    ERASCR_Erase_Screen;
                    DBWIDT_Double_Width(7);
                    CURPOS_Cursor_Position (7,7);
                    WRITELN ('TRANSLATION COMPLETE', crlf);
                    WRITELN (SW_File,'TRANSLATION COMPLETE',
                             crlf);
                    CURPOS_Cursor_Position (24,50);
                    WRITELN ('RETURN TO Continue...' );
                    READLN;
                    Close (SW_File);
                  END;
            END;
      3  :  BEGIN
              DBWIDT_Double_Width(7);
              CURPOS_Cursor_Position (7,12);
              WRITELN ('QUITTING......', crlf);
              Dun := TRUE;
            END;
      END; {of case}
   UNTIL Dun;
END.
```

☆U.S. GOVERNMENT PRINTING OFFICE: 1991—834-788

The following number is for U.S. Department of Energy OCRWM management purposes only and should not be used when ordering this publication.
NNA.910507.0002