AN ENGINEERING GRAPHICS APPLICATION SYSTEM

AN ENGINEERING GRAPHICS APPLICATION SYSTEM

By

HONG ZHOU, B.Eng., M.Eng.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (1992) **McMASTER UNIVERSITY**

(Computation) Hamilton, Ontario

TITLE: An Engineering Graphics Application System

AUTHOR: Hong Zhou, B.Eng.(Chongqing University, P.R.China)

M.Eng.(Chongqing University, P.R.China)

SUPERVISOR: Dr. W.F.Smyth

NUMBER OF PAGES: viii, 80

ABSTRACT

EGAS is the acronym for the Engineering Graphics Application system, developed specifically to facilitate engineering design. It provides interactive drafting tools for civil engineering, mechanical engineering and architectural graphics, and is particularly useful to structural engineers. EGAS has been developed based on X in a Unix environment.

ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. W.F. Smyth for his supervising and help throughout this project. I am grateful for valuable comments to my first reader, Dr. P.J. Ryan. A special thanks also goes to Mr. D. Trottier for the system support and advice.

This work is dedicated to my husband Qiang and my dear parents for all their love and understanding.

TABLE OF CONTENTS

CHAPTER ONE

INTRODUCTION

1.1 Overview of This Project

Computer graphics, a method of making drawings with a computer, has come into broad usage with more growth expected. One application of computer graphics is computer-aided drafting for engineering design. With computer-aided drafting, the computer is used as a tool to assist the user with the routine phases of the drafting, but must not restrict in any way the creativity or options available to the user.

In engineering drafting, even for modest structures, large amounts of data such as points, lines, dimensions, loading stresses are involved. Manual work is not only inefficient, but also has the disadvantage that created drawings are difficult to modify. The computer is therefore introduced for entering, storing, retrieving, displaying and manipulating these data. This project is designed for such a purpose.

1.2 Description of EGAS

EGAS is the acronym for the Engineering Graphics Application System, developed specifically to facilitate engineering design. It provides interactive drafting tools for civil engineering, mechanical engineering and architectural graphics, and is particularly useful to structural engineers. The whole system is like a toolbox with many features to make design and drawing of engineering structures easier.

EGAS is based on a series of built-in functions. The functions are designed to help in quickly producing the drawing which the user wants. They also offer features that let the user easily correct drawing errors and make even large revisions without redoing an entire drawing. Utilities are provided for storing and retrieving created drawings too.

All of the functions are organized into nested menus which provide various options and on-line help facilities at each stage. EGAS provides the user with a flexible way to access all of its functions. Unlike some of the menu-driven packages, EGAS does not require the user to exit from a lower level of nested menus before the user can make a selection on a higher level one. All menus which are visible on the screen are accessible by the user, no matter which level the menu belongs to. (See section 2.1.1.)

A special feature of EGAS is that it allows the user to create or delete a set of points or lines simultaneously. The points or lines in one set usually have some common attributes, such as n points in a certain direction with the same distance between them. The point set can be specified either by the user or by system default. So also the line set. (See section 2.3.)

Another powerful feature of EGAS is that it lets the user insert an existing drawing (stored on disk) into the drawing currently being created. Thus the user can interactively construct a drawing "part", store it in a regular EGAS drawing file, then insert as many copies of it as the user likes in subsequent drawings. (See section 2.2.4.)

It is the goal of EGAS to produce precise final drawings; however, EGAS deals with reference lines only at this stage of development.

No technical computer knowledge is required to use EGAS effectively; practice and thorough understanding of its features are the keys to proficiency.

1.3 About The Hardware and Software

EGAS runs on a SUN SPARC workstation. A graphics monitor is used to display the drawings created by EGAS. Everything done to the drawing is shown on the monitor, so the user can watch the progress each step of the way. A three-button mouse is used as a graphics input device, with which the user communicates with the system. It is operated by moving it around the tabletop while a cursor symbol on the screen follows the movement.

The program is implemented in C, and uses the X window system. The X window system, called X for short, is a network-based graphics windowing system for workstations. It was developed by MIT and has been adopted as an industry standard. X can run on many different types of systems. EGAS has been developed based on X in a UNIX environment.

1.4 Organization of This Write-Up

The current chapter is an overview introduction of this project. The rest of this report is organized as follows:

Chapter 2 describes various features of EGAS in detail, concentrating on what functions EGAS provides and how to use them. Examples will be given. This chapter is somewhat like a users guide.

Chapter 3 discusses programming and implementation techniques. An introduction to X gives general X concepts, including the basics of the X toolkit and the X widget set. There is discussion of how X is used to develop EGAS.

Chapter 4 describes some of the algorithms involved in the EGAS programs, focusing on the calculation of certain built-in functions.

Chapter 5 discusses the project in general. It provides some EGAS examples and makes some suggestions for future work.

All the references are listed at the end of the write-up.

Appendix A contains detailed procedures for installing EGAS on a user's system and setting the operating environment.

Appendix B provides instructions for using EGAS. It includes all of the documents used by the HELP facility.

CHAPTER TWO

FEATURES OF EGAS

2.1 Getting Started with EGAS

It is assumed that EGAS has been installed on the user's system properly, with all the required files available. The operating environment also has been set as required (refer to appendix A for detailed procedures).

To run EGAS, type: **egas**

followed by a RETURN.

2.1.1 Screen Displays

After EGAS is started, the initial screen configuration is as shown in Fig. 2.0. The main menu appears on top of the screen; this provides access to various EGAS functions. The side menu, which is used for setting values for EGAS system parameters, is at the upper right corner of the screen. The blank screen under the main menu is the drafting area. A cursor symbol also appears on the screen; its position is controlled by the position of the mouse on the tabletop. The cursor is called the Screen Pointer in EGAS.

When a main menu selection is made, a pull-down menu appears below the main menu title, as shown in Fig.2.1. The pull-down menu is called a submenu in EGAS.

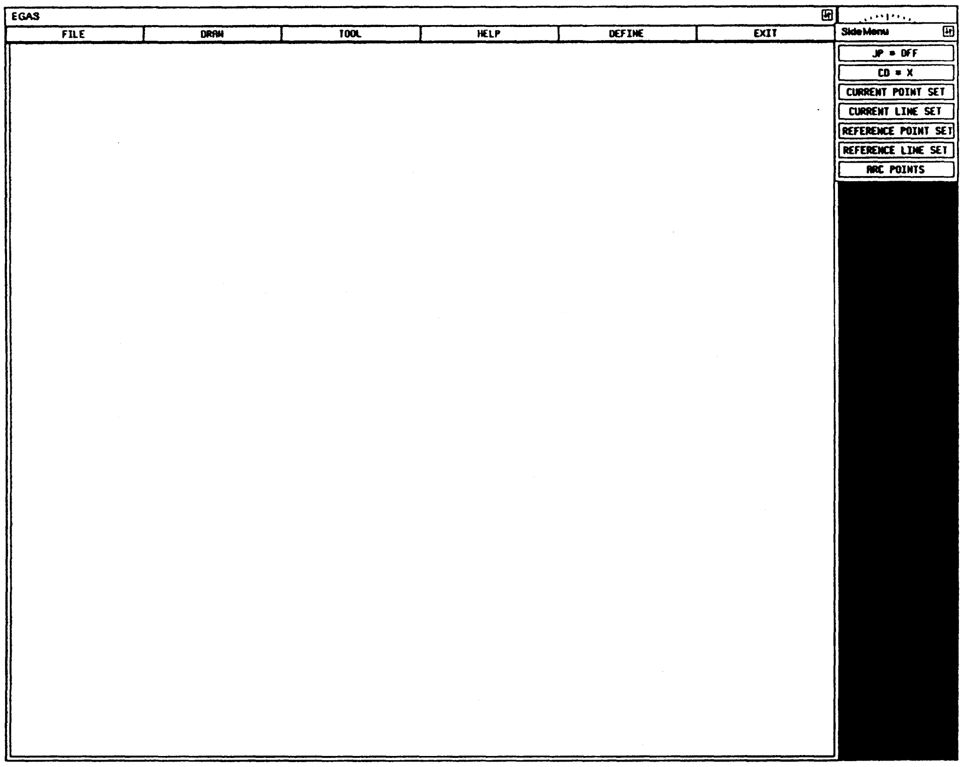


Fig. 2.0 The Initial Screen Configuration of EGAS

There is a special submenu in the lower corner of the screen, which is called displace menu. It is used for setting the values of variables. It only appears when the DRAW option has been selected, since those variables are required only by one of the drawing commands -- DISPLACEMENT.

All menus have equal priority in terms of menu access, although each has different properties. The user may choose a menu option from any one of the EGAS

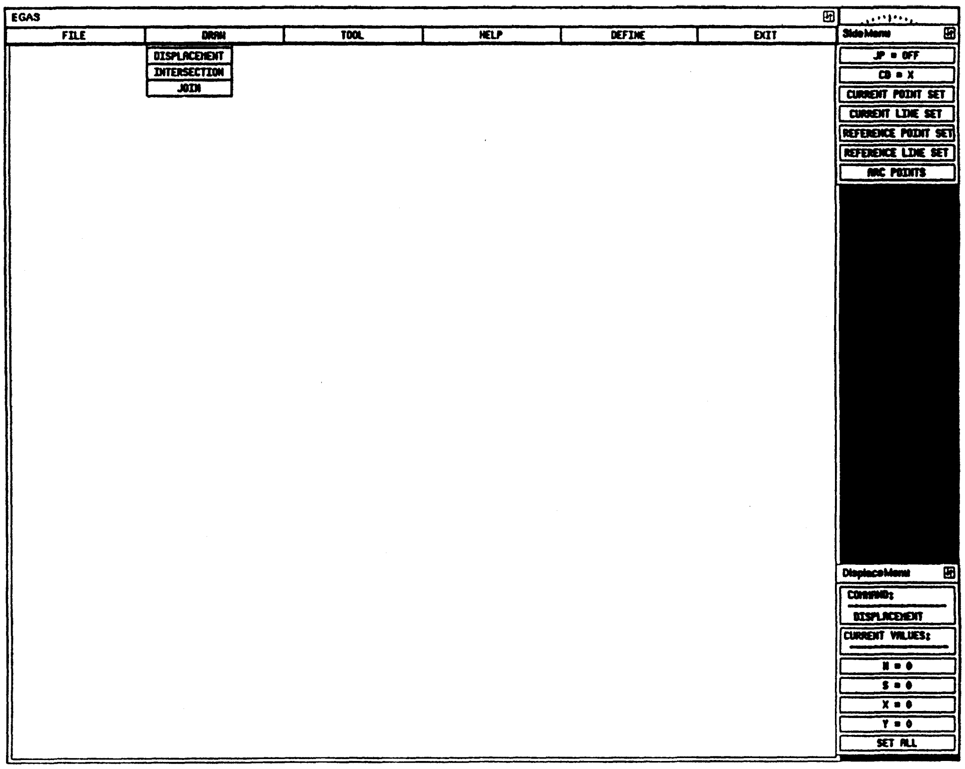


Fig. 2.1 The Submenu in EGAS

menu as long as it is visible on the screen. It is not necessary to exit from the current submenu before a main menu or a side menu selection is made.

2.1.2 User Interface

EGAS is a menu driven package. All functions are organized in nested menus. No command needs to be typed in from the keyboard. A mouse is the only pointing device used in EGAS, which makes the entry of commands from the menus very easy.

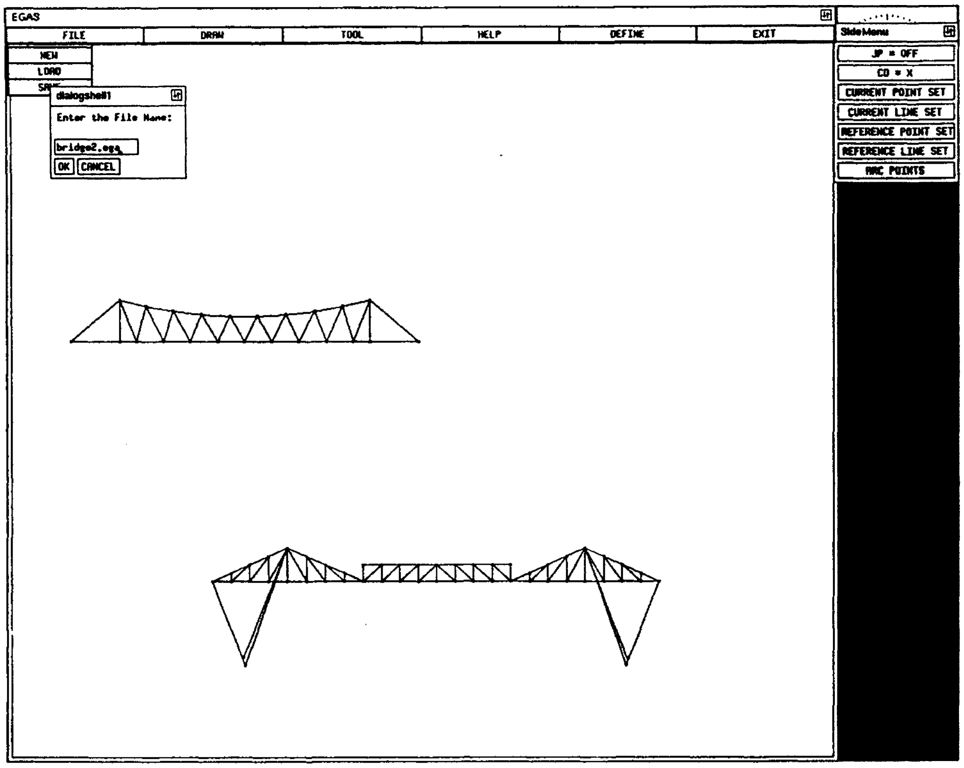
The three mouse buttons have a consistent use in EGAS. The left button is used for menu selections or screen picking, such as selecting a point on the display. A menu selection is made simply by moving the Screen Pointer into the desired menu box and pressing the button. Screen picking is similar to menu selection: the only difference is that the Screen Pointer is moved to the desired position in the display instead of being moved into the menu box.

The middle button is used as a drafting pen: pressing the button draws a single point on the display when the Screen Pointer is in the drafting area. If the middle button is pressed while the Screen Pointer is at some place on the screen other than the drafting area, there is no effect.

The right button is not used by EGAS.

Most EGAS functions involve parameters: some of them are system-defined while others need to be specified by the user, such as the number of points to be created. All the user-defined data are entered through a specially designed pop-up window called the dialog-box.

A dialog-box appears on the screen when an associate menu selection is made, as shown in Fig.2.2. Then the user can enter a value from the keyboard, and the value is displayed in the dialog-box while it is being typed in. A displayed value in the dialog-box can be changed or deleted by pressing the BACKSPACE key. The value can either be a character string or an integer.



Enter the File Name:

bridge2.egs

Fig. 2.2 The Dialog-Box

To register the value, click on the OK box; to cancel the value, click on the CANCEL box. No access to any menu box is allowed when a dialog-box is on the screen. The dialog-box must be popped down, either by registering the value or by cancelling it, before a menu selection is acceptable.

You may already have noticed that in the EGAS screen, there is a part on the right side which remains dark. Actually, that part is a portion of the terminal screen and does not belong to any EGAS windows. By keeping it along with the EGAS displays, the user can access all the utilities provided by X itself, such as printing windows or doing a screen dump through the X program Xdpr.

2.2 The Main Functions

After EGAS is loaded, the following six options are listed in the main menu:

FILE DRAW TOOL DEFINE HELP EXIT

FILE is used to create drawing files.

DRAW and TOOL are used to provide interactive drawing functions, including functions for creating or modifying drawings. To start a new drawing from scratch, choose DRAW. To make changes to an existing drawing, choose TOOL.

DEFINE is used to define the units and accuracy for a EGAS drawing. A unit corresponds to either the IMPERIAL or the METRIC system and may be specified by the user. (DEFINE has not been implemented at this moment of development. For more details about it, see section 5.3.)

HELP provides documentation for EGAS.

EXIT terminates the execution of EGAS.

2.2.1 Save and Retrieve Drawing

The submenu of FILE contains functions associated with file processing. There are three options:

1. **NEW**: starts a new drawing.

2. **LOAD**: retrieves an EGAS data file specified by the user and displays the drawing on the screen.

3. **SAVE**: saves the created drawing to an EGAS data file.

When LOAD or SAVE is selected, a dialog-box appears on the screen which prompts the user to enter the file name. There is no special format required for the name of an EGAS file; all the names which are valid in UNIX will be accepted by EGAS.

Trying to retrieve a file which does not exist causes the dialog-box to vanish from the screen but has no effect on the display. Trying to save a drawing to an existing file rewrites that file; EGAS will prompt the user with a warning message before it performs the rewrite.

2.2.2 Create New Drawing

Three basic graphics primitives are used to construct a drawing in EGAS: a point, a line and an arc of a circle. A primitive appears on the drafting area of the screen after it is drawn by the user.

The most convenient way to draw a single point or a single line is by using the mouse. A point is created by moving the Screen Pointer to the desired location and pressing the middle button on the mouse. A line is created by creating its two end points while the system parameter Join Point (JP) has been set to ON (see section 2.3.1 for specification of JP).

EGAS also provides a set of built-in drawing functions, all contained in the submenu of DRAW. These are as follows:

1. **DISPLACEMENT**: creates points or points and lines according to the user's needs;
2. **INTERSECTION**: creates points at the intersections of two sets of lines;
3. **JOIN**: creates a set of lines by joining two sets of points, or creates an arc of a circle by joining three points.

The above three functions have one thing in common: they all require specific information from the user. After the information is supplied, a set of primitives may be created simultaneously.

Section 2.3 will discuss most of the cases each drawing function may apply to. For detailed instructions for using these functions, refer to Appendix B.

2.2.3 Manipulate Existing Drawing

Interactive drawing tools are provided by EGAS for manipulating a created drawing. Three operations have been defined under the main menu item TOOL. They are:

1. **INSERT DRAWING**: inserts an EGAS drawing saved in a file into the drawing currently being worked on;

2. **DELETE**: deletes part of the drawing;

3. **CLEAR DRAWING**: deletes the entire drawing;

4. **REDRAW**: retrieves the drawing deleted by CLEAR DRAWING.

The user is prompted to give a file name and a Scale Factor (SF) when INSERT DRAWING is selected. The specified file is the one which contains the previously saved drawing, and SF is used to specify the size of the drawing. The drawing in the file is copied into the current drawing with its size reduced and its upper left corner placed at the insertion point. The first point in the Current Point Set (CPS) is used as the insertion point. (For a description of CPS, see below.)

The value of SF should be an integer in the range [1..10]. When SF equals one, the drawing is inserted with its original size; otherwise, it is inserted with its size reduced by the factor. No enlargement is allowed by EGAS when doing a drawing insertion.

A particular drawing becomes a part of the current drawing after it has been inserted. With this function, the user can create his own library of "parts" interactively, and insert them into the current drawing at any specified location. A part is nothing special; any drawing created by EGAS and saved in an EGAS file may be used as a part. A part can be inserted into one drawing as many times as the user wants.

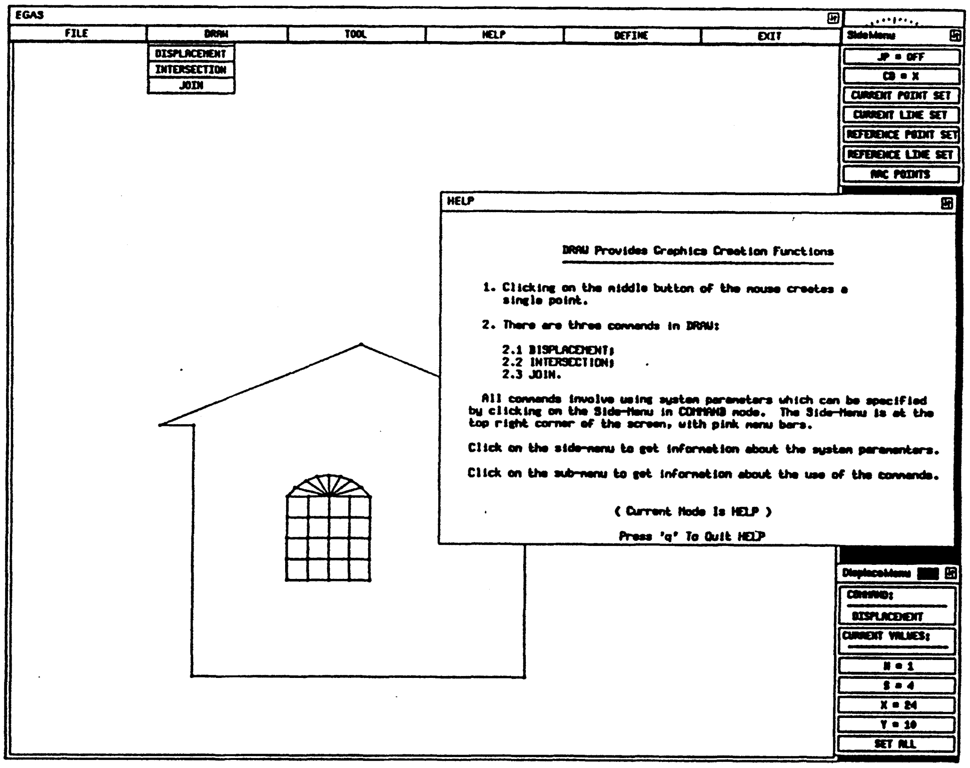
It is required that a current object set be defined before function DELETE is performed. The current object set can be one of Current Point Set (CPS), Current Line Set (CLS) or Arc Point Set (APS). (See section 2.3.1 for the specifications of CPS, CLS and APS.)

To delete a set of points, first specify them as the CPS, then choose DELETE. The method of deleting a set of lines is similar. If an APS is defined as the current object set, then the arc which connects the points will be deleted, instead of the APS itself.

After CLEAR DRAWING is performed, the data for the deleted drawing will be memorized by the system and can be retrieved by selecting REDRAW. If any drawing has been created between the performance of CLEAR DRAWING and REDRAW, the most recently created drawing and the deleted drawing will be overlapped on the display.

2.2.4 On-Line Help

EGAS provides an on-line help facility which makes available a description of any function at any stage during the running EGAS.



DRAW Provides Graphics Creation Functions

1. Clicking on the middle button of the mouse creates a single point.

2. There are three commands in DRAW:

   2.1 DISPLACEMENT;
   2.2 INTERSECTION;
   2.3 JOIN.

All commands involve using system parameters which can be specified by clicking on the Side-Menu in COMMAND mode. The Side-Menu is at the top right corner of the screen, with pink menu bars.

Click on the side-menu to get information about the system paramenters.

Click on the sub-menu to get information about the use of the commands.

(Current Mode Is HELP)

Press 'q' To Quit HELP

Fig 2.3 HELP in EGAS

There are two modes in EGAS. One is COMMAND mode, the other is HELP mode. The system is in COMMAND mode by default. Clicking on the HELP box in the main menu changes the system to HELP mode.

When EGAS is in HELP mode, any menu selection brings out the corresponding help document.

Fig 2.3 shows an example of HELP in EGAS.

2.2.5 Exit EGAS

EXIT is used to terminate EGAS and return control to the operating system.

2.3 Commands for Creating A New Drawing

Certain parameters are introduced before further discussion of drawing functions.

2.3.1 Definitions of Parameters:

Most EGAS functions require some parameters, the values of which need to be specified. All the parameters have a consistent use within EGAS. Here are their specifications:

DEFINITION	POSSIBLE VALUES
JP = Join Point	ON, OFF;
CD = Current Direction	X, -X, Y, -Y;

CPS = Current Point Set	0, Ordered list;
CLS = Current Line Set	0, Ordered list;
RPS = Reference Point Set	0, Ordered list;
RLS = Reference Line Set	0, Ordered list;
APS = Arc Point Set	0, Three ordered points;

The above parameters are listed in a side-menu, which stays on screen during the whole process of running EGAS. The values of JP and CD can be toggled by clicking on the menu box. The values of the rest of the parameters are set in two steps; first picking up the desired points on the screen, then clicking on menu box to define. (See Appendix B for details.) All the objects contained in a set may be referenced by the set's name after they are defined by the user. EGAS functions may either apply to an object set or use it as a reference; for example, all the points in CPS may be erased by a single command DELETE or new points may be created whose coordinates are calculated using the coordinates of those in CPS. How the system behaves depends on how the user makes use of the commands listed in the menu.

In addition to system parameters, four variables are used by the command DISPLACEMENT. They are:

N: Number of points or lines to be created;

S: Spacing between points or lines to be created;

X: Distance in the X direction;

Y: Distance in the Y direction.

In the current program, the measuring of a unit length given by the user is fixed: it corresponds to a distance of 10 pixels on the screen. The method of defining a variable is through a dialog-box. (See Appendix B for details.) The values of these four variables are displayed on the displace-menu (given on the screen as "Displce Menu'). When any value is changed by the user, the display will automatically be updated.

2.3.2 DISPLACEMENT

DISPLACEMENT is a function for creating points and lines. Different combinations of parameters give different displacement. Here are the general cases:

1. CLS + N: Creates N equally-spaced points in each line of the CLS (see Fig 2.4).

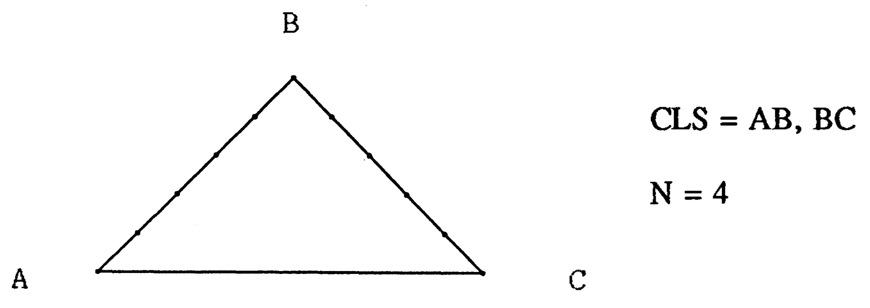
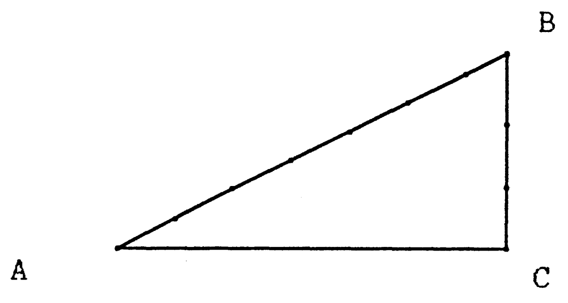


Fig. 2.4 DISPLACEMENT - Action 1

2. CLS + S: Creates as many points as possible spaced by S on each line of the CLS (see Fig 2.5).

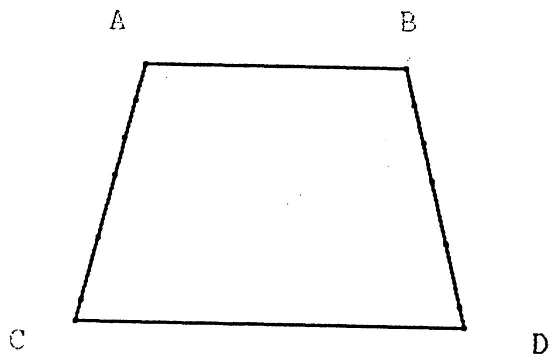


CLS = AB, BC

S = 5

Fig. 2.5 DISPLACEMENT - Action 2

3. CLS + N + S: Creates N points spaced by S on each line of the CLS; it has a "follow-on" feature which allows a subsequent N,S pair (see Fig. 2.6).



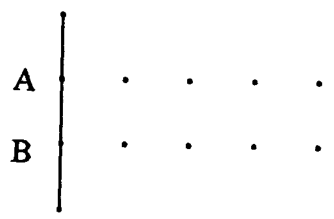
CLS = AC, BD;

N = 3, S = 5;

Followed by N = 2; S = 8;

Fig 2.6 DISPLACEMENT - Action 3

4. CPS + CD + N + S: Creates N points spaced by S in CD from each point of the CPS (see Fig. 2.7).



CPS = A, B

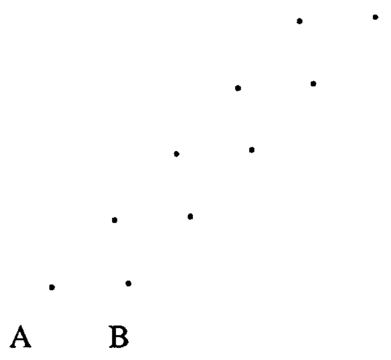
CD = X

N = 4

S = 2

Fig 2.7 DISPLACEMENT - Action 4

5. CPS + N + X + Y: N points displaced by (X, Y) from the preceding one starting with each point of the CPS (see Fig. 2.8).



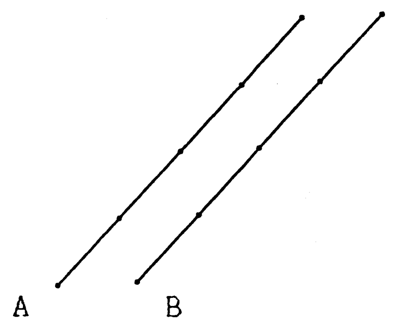
CPS = A, B

N = 4

X = 10 Y = 10

Fig. 2.8 DISPLACEMENT - Action 5

One thing to be mentioned is that the above two examples have assumed JP = OFF while performing the displacement actions. If JP = ON, the created points are joined to CPS by a set of straight lines (see Fig. 2.9).



JP = ON

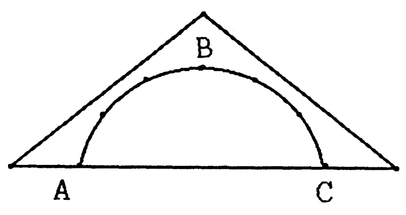
CPS = A, B

N = 4

X = 10 Y = 10

Fig. 2.9 DISPLACEMENT - Action 5

6. APS + N : creates N equally-spaced points on an arc of a circle, which is defined by points of APS (see Fig.2.10).



APS = A, B, C

N = 5

Fig. 2.10 DISPLACEMENT - Action 6

7. APS + S : Creates as many points as possible spaced by S on APS (see Fig. 2.11).

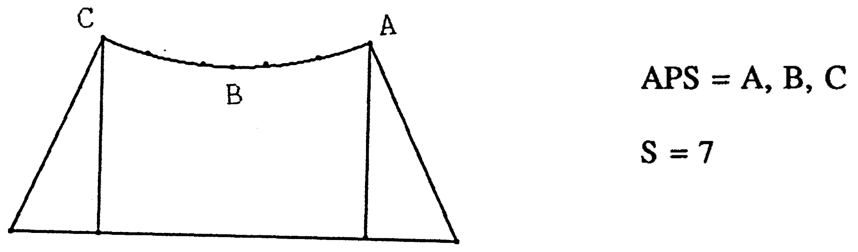


Fig. 2.11 DISPLACEMENT - Action 6

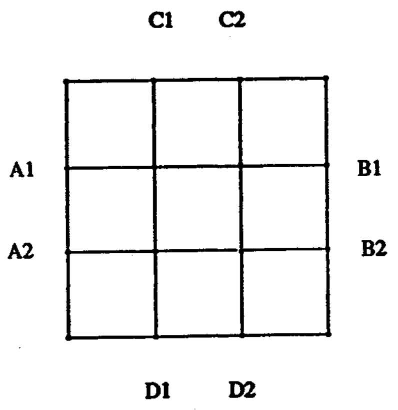
When using DISPLACEMENT, the system determines which one of the described actions will be performed according to the information supplied by the user. To choose one of them, the user needs to set to zero those variables not required for the current action, define a current object (point or line) set, and then click on the menu bar of DISPLACEMENT.

From the point of view of executing DISPLACEMENT, CPS, CLS and APS have a property called mutual exclusion: only one of them can be the valid current object set at a time. This rule applies to the most recently defined object set. However, the value of any current object set is retained until it is changed by the user. For example, after APS is defined, CPS is no longer being taken into account by DISPLACEMENT when it is trying to find out the proper action, but the content of CPS remains and can be used, for example, by command JOIN.

2. INTERSECTION

In general, INTERSECTION creates points at the intersection of every current line with every reference line. The particular cases are as follows:

1. CLS + RLS: (see Fig. 2.12).



CLS = A1B1, A2B2

RLS = C1D1, C2D2

Fig. 2.12 INTERSECTION - Action 1

2. CPS + CD + RLS: (see Fig. 2.13).

CPS = A1, A2, A3

CLS = BC

CD = X

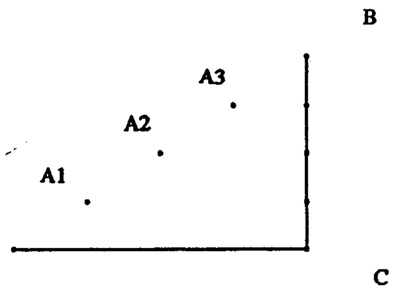


Fig. 2.13. INTERSECTION - Action 2

3. CLS + RPS + CD: (similar to Action 2).

To run INTERSECTION, first specify the required parameters, then click on the INTERSECTION menu box.

3. JOIN

This is the only EGAS drawing function designed only for creating lines, either straight lines or an arc of a circle. The way JOIN works is relatively simple, as shown in the two possible cases:

1. CPS + RPS: Current points and reference points are joined by straight lines, first CPS to first RPS, second CPS to second RPS, etc. (see Fig 2.14).

Note that if the number of points contained in CPS and RPS are not equal, EGAS takes the smaller one. For example, if there are 5 points in CPS and 6 points in RPS, only 5 lines will be created between CPS and RPS.

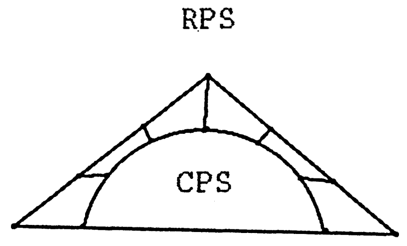


Fig. 2.14 JOIN - Straight Lines

2. Arc Points: Three points are joined by an arc of a circle whose centre and radius are defined by the three points (see Fig. 2.15).

APS = A, B, C

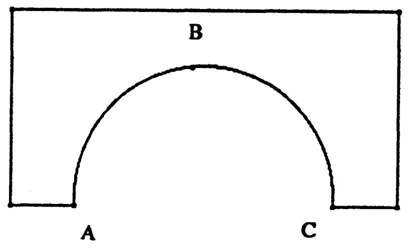
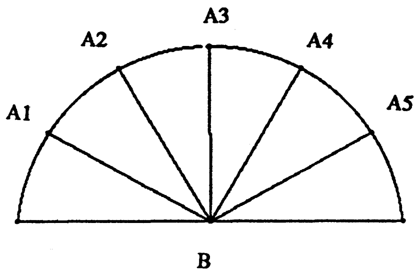


Fig. 2.15 JOIN - Arc of A Circle

The three points of APS must be selected in clockwise order around the arc.

One point may be used more than once in CPS or RPS, this is done by clicking on it repeatedly, Fig. 2.16 shows an example of this case.

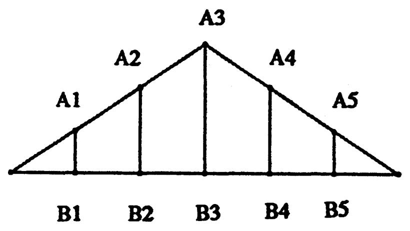


CPS = A1, A2, A3,
A4, A5

RPS = B, B, B,
B, B

Fig. 2.16 One Point Is Used More Than Once In RPS

Redefining CPS does not alter the content of RPS, or vice versa. Fig. 2.17A and Fig. 2.17B show examples of this.

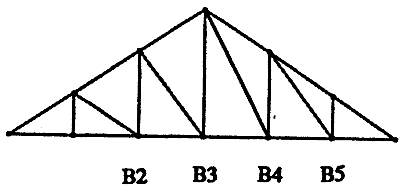


CPS = A1, A2, A3, A4, A5

RPS = B1, B2, B3, B4, B5

Fig.2.17A Join CPS and RPS

As shown in Fig.2.17A, CPS is define as A1, A2, A3, A4, A5, and RPS B1, B2, B3, B4, B5. When RPS is redefined as B2, B3, B4, B5, CPS remains the same as before, then Command JOIN connects 4 lines between CPS and newly-defined RPS, as shown in Fig.2.17B.



RPS = B2, B3, B4, B5

Fig.2.17B Redefine RPS, JOIN

CHAPTER THREE

PROGRAMMING AND IMPLEMENTATION TECHNIQUES

3.1 Introduction To X

X is the software environment which EGAS has been implemented in; therefore, it is necessary to introduce some general concepts of X before we start to discuss the major issue of this chapter. To gain a conceptual understanding of X, it is helpful to look at X's history and note the initial goals of the X designers.

In 1984, MIT officials were faced with a problem common to the business and academic computing worlds: they were the owners of a motley set of incompatible workstations acquired through donation and purchase. The goal was to build a network of graphical workstations that could be used as teaching aids. Faced with a patchwork of operating systems and hardware vendors, MIT officials decided to form Project Athena, an MIT development team working in association with DEC and IBM. [3]

Project Athena's solution was to design a network that would run local applications while being able to call on remote sources. This network was roughly based on a Stanford University software environment called W. By linking together IBMs, DECs and other disparate workstations through a graphical networking environment, the designers created the first operating environment that was truly hardware and vendor independent, the X Window System. Instead of developing a different graphical interface for each workstation hardware vendor's product, X gave MIT a way to program workstation applications using a common and portable interface to workstation hardware [3].

X was a success. In January 1987, a dozen vendors announced a cooperative effort to standardize and promote X. Almost all workstation vendors have, by now, accepted the X Window System as a standard interface to their workstation hardware [1].

X is a large and complex system. Much of that complexity comes from the fact that X attempts to deal with virtually every type of computer graphics display available. X cannot be explained completely even in a book: our introduction to X will concentrate on the typical X display and the general X architecture.

3.1.1 Overview of X Display and X Architecture

X is typically run on a workstation with a large screen. Like other windowing systems, X divides the screen into multiple input and output areas called windows, and allows the user to work with multiple programs simultaneously, each in a separate window.

The operations performed within a window can vary greatly, depending on the type of program running it. Certain windows accept input from the user: they may function as terminals. Other windows simply display information, such as the time of day or a picture of the characters in a particular font, etc.

The windows an application programmer probably uses most frequently are terminal emulators, windows that function as standard terminals. The terminal emulator included with the standard release of X is called xterm. In an xterm window, the user can do anything he might do in a regular terminal: enter commands, run editing sessions, compile programs, etc.

X takes user input from a pointer. The pointer is usually a mouse but could just as well be a track-ball or a tablet. The pointer allows the user to control a program without using the keyboard, by pointing at objects drawn on the screen such as menus and command buttons. Of course, X also handles keyboard input. The pointer directs keyboard input from window to window. Only one window at a time can receive keyboard input.

Most window systems are kernel-based: that is, they are closely tied to the operating system itself and can only run on a discrete system such as a workstation. X is not part of any operating system, but is instead composed entirely of "user-level" programs.

The X Window System architecture is based on a simple client/server relationship in which the display server is the program that provides display capabilities and tracks client input, while clients are application programs that perform specific tasks.

An important X concept which needs to be introduced here is the "event". An X event is a data structure sent by the server that describes something that just happened that may be of interest to the application. There are two major categories of events: user input and window system side effects. For example, the user pressing a keyboard key or clicking a mouse button generates an event; a window being moved on the screen also

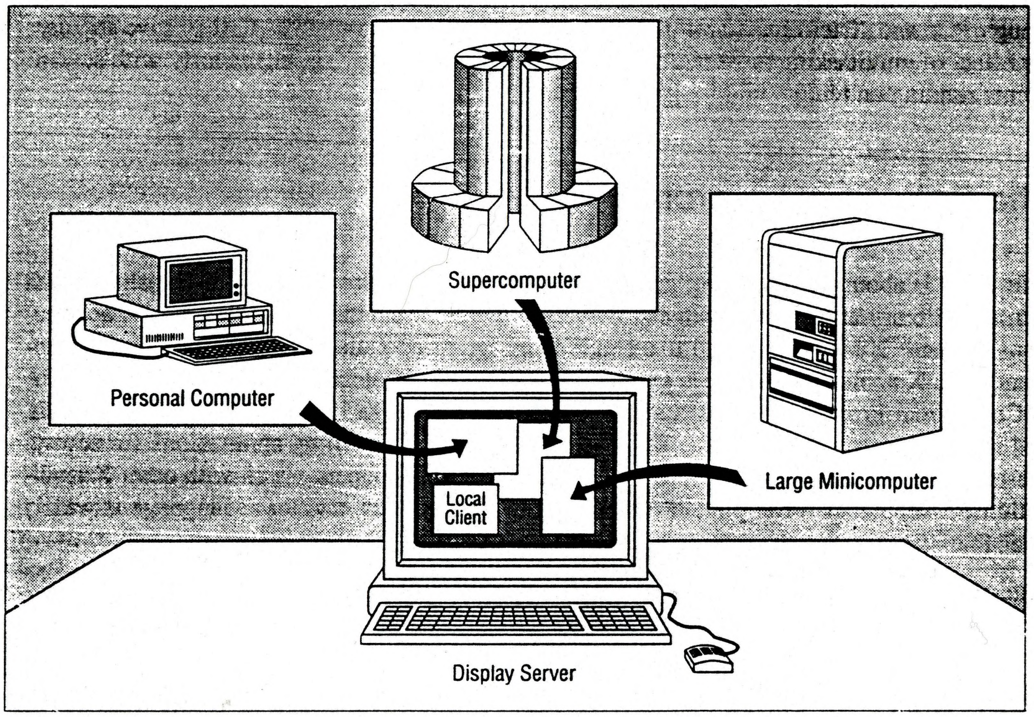


Fig 3.0 Applications Can Run On Any System Across the Network

generates events -- possibly in other applications as well if the movement changes the visible portions of their windows. It is the server's job to distribute events to the various windows on the screen.

Because X is a networked environment, the client and server don't necessarily compute on the same system. Instead, X allows distributed processing. For example, a relatively low-powered PC or workstation may be used as a display server to interact with clients that are running on a more powerful remote system. All user input and displayed output occur on the PC or workstation server and are communicated across the network using the X protocol. Fig.3.0 shows a diagram of such a network.

Another advantage of the client-server model is that since the server is entirely responsible for interacting with the hardware, only the server program must be machine specific, X client application programs can be easily ported from system to systems.

3.1.2 Linkage From C To X

Most X applications are written in the C programming language, as is EGAS.

X is not restricted to a single language or operating system: the only requirement of an X application is that it generate and receive X protocol messages. X is defined in terms of a network protocol. The network protocol specifies the format and meaning of messages passed over a display connection between application and workstation.

C programmes gain access to X workstations by calling Xlib, the X programming interface.

Xlib is the subroutine package that maps to X protocol requests or provides utility functions. What Xlib actually does is convert the C language function calls to the X protocol request that implements the given function. An X application programmer is unlikely to be exposed to the X protocol directly, since Xlib efficiently hides the protocol details.

Although Xlib provides full access to the capabilities of the X protocol, it does little to make programming easier. More and more application programs use the high-level X Toolkit to mask some of the complexity of the X protocol. And this is the way in which EGAS is written.

3.1.3 About X Toolkit

X Toolkit is the collective name for two C-language subroutine libraries: Xt intrinsics library and a widget set library, which are designed to simplify the development of X applications using reusable components.

A widget is a pre-built user-interface component. Typical widgets include scrollbars, menus, dialog boxes, drawing areas, etc. Each widget is made up of its own X window, but most of the work that goes on in that window has already been taken care of - all the application programmer has to do is assemble the widgets and write application-specific code that will be called in response to events in the widgets.

The Xt Intrinsics library consists of routines for using and building widgets. Xt

Intrinsics are written using Xlib. Both Xt Intrinsics and Xlib are required by any X implementation that allows programming of X applications.

There are several widget sets provided by system vendors to implement their particular user-interface styles. The one EGAS uses is Xaw, which stands for X Athena Widgets. (Xaw was developed by MIT's Project Athena.) It is based on Xt and provides a small number of widgets that can be used to write application programs.

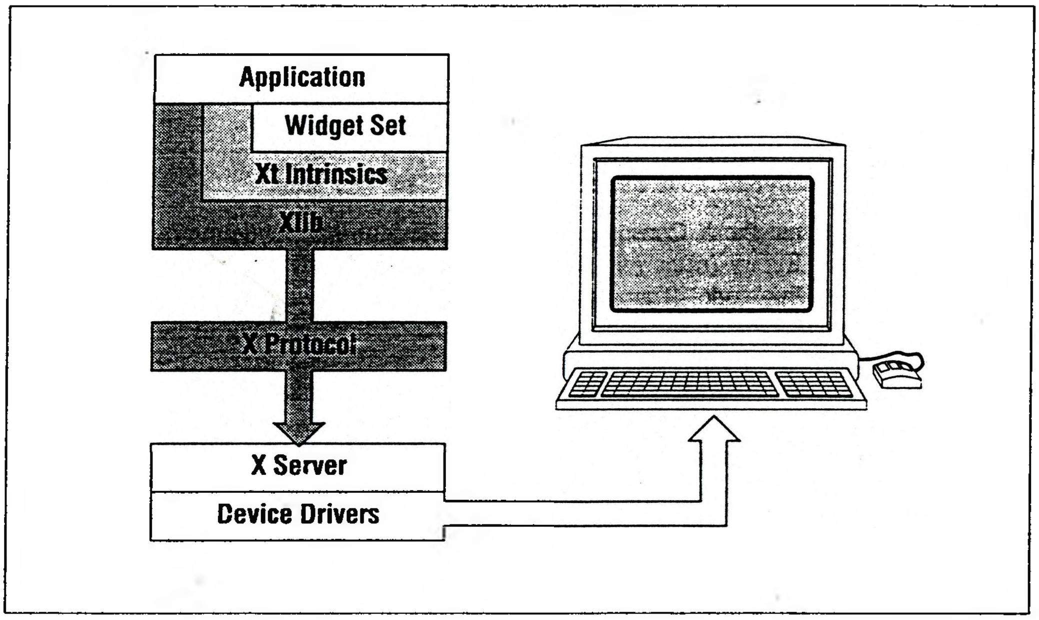


Fig. 3.1 The Software Architecture of X Toolkit based applications.

Fig.3.1 shows the layering of software in an application that uses Xt Intrinsics and a widget set. Notice that the Intrinsics are based upon Xlib, the lowest-level C-language interface to X. Applications often need to call Xlib directly to accomplish certain tasks such as drawing. Xt does not provide its own graphics calls, nor does it provide access to every X protocol feature.

3.2 Programming with Widgets in EGAS

As we have seen, the purpose of Xt is to provide an object-oriented layer that supports the user-interface
abstraction -- a reusable component, the widget. The use of widgets separates application code from user-interface code, an important programming principle which EGAS follows.

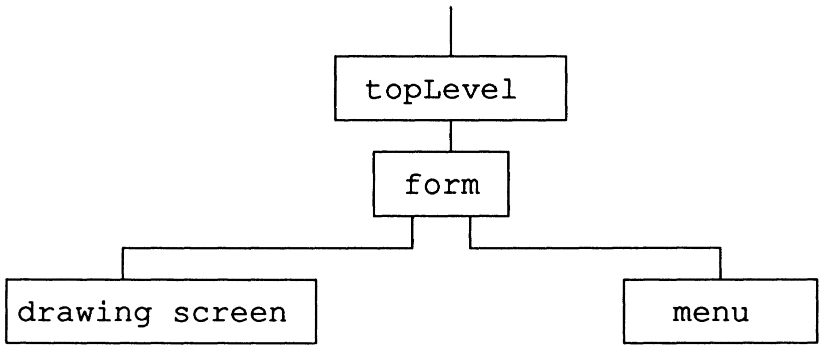


Fig. 3.2 Main Structure of EGAS

While implementing EGAS, we tried to use the existing widgets in Xaw to simplify the work as much as possible. Fig. 3.2 shows the main structure of EGAS.

The topLevel is a shell widget, a special widget created by the call to initialize the

Toolkit. It is used as the parent of all other application widgets (with the exception of pop-ups, see section 3.3.2), and includes special functionality that allows it to interact with the window manager. The topLevel is not visible when EGAS runs, since it is overlaid by the drawing screen.

The form is a Constraint widget; like composite widgets, it manages the layout of children. The difference is that a Constraint widget lets the application provide layout information for each child. This is a more powerful way to arrange children because it allows the programmer to provide different rules for how each child will be laid out.

The drawing screen is a special-purpose application window. Its characteristics are not supported by an existing widget, but by application code. Most of the EGAS drawing functions are implemented in this window, which is the user's drafting area.

The menu is a Composite widget, which is used to contain other widgets, such as Command widget, Label widget, etc. These widgets are called child widgets of Composite widgets. The Menu together with its child widgets form the user interface of EGAS. Composite widgets play an important role in a widget set, since they insulate the programmer from having to place each widget individually. Composite widgets automatically adjust the layout of their children when child widgets are added or removed; this provides a great flexibility to the programmer during the implementation of EGAS.

In the next two sections, we will go further into the programming techniques for the user interface and the drawing screen.

3.3 Implementing the User Interface

Menus are one of the most important user-interface elements in window-based applications. In the development of EGAS, the first task was to implement the menus.

As we have described in Chapter Two, EGAS is a menu driven package. All functions are organized in nested menus, including the main menu, a side menu and submenus. From the programmer's point of view, those menus are built upon a hierarchy of widgets. The root of the hierarchy is the shell widget topLevel, followed by the composite widget menu. Each menu box displayed on the screen is represented by a command widget in the hierarchy. A command widget initiates some application action when it is clicked on with the mouse.

One way that the program arranges for widgets to invoke application functions is by registering application functions with Xt. Once the program is running, Xt will call these functions in response to occurrences in the corresponding widgets.

EGAS uses callbacks to link widgets and application functions, which is one of the three mechanisms available in Xt. Generally speaking, a widget declares one or more callback lists as resources; the corresponding application functions are added to these call back lists, which will be invoked whenever the predefined callback conditions are met. For example, in EGAS, the condition for a command widget is being clicked by the user;

a command widget invokes a certain function when the user clicks on it.

Callbacks are resources, so that the programmer can set or change the application functions without affecting the widget itself. In this way the development of the user interface and the application functions are isolated from each other.

3.3.1 Pop Up Dialog-Box

One of the important X mechanisms used to implement the user interface is the Pop up.

Pop ups are widgets that are not visible until a certain user command is given, or a situation arises in which the program requires user input; even then they are visible for only a short period.

More precisely, Pop ups are not a kind of widget, but rather a way of using widgets. Any widget can be used as a pop up. A special parent widget called a TransientShell must be created first. Then the widget to appear in the pop up is created as the child of TransientShell; and this widget may be a simple widget or a composite widget with children. However, TransientShell must have only one child.

In EGAS, all of the user-defined data are entered by means of the dialog-box, which is a typical use of Pop ups. When a dialog-box appears on the screen, the actual widget popped up is an Athena Dialog widget. The Dialog widget is a widget designed to prompt for auxiliary input from a user; for example, when a file name is required from the user.

The Dialog widget creates its own widget children. This helps to combine existing widgets in a standard, useful way. Dialog widgets in EGAS contain four areas. The first line contains a Label widget providing a description of the function of the Dialog widget; for example, the string "Enter Value for X:". The second line contains a Text widget into which the user types input. The third line contains two Command widgets that let the user confirm or cancel the dialog input.

Different values of the Label widget and various application programs for Command widgets are used for each dialog-box in EGAS. But only one TransientShell widget and one Dialog widget are defined for common use, since it never occurs that two dialog-boxs appear on the screen simultaneously. At the time when a dialog-box needs to be popped up, the position of the widget, usually determined by the position of the menu box which calls the pop-up, is returned first by an Xt function XtTranslateCoords(); then the position where the Dialog widget will be placed is set accordingly by the Xt function XtVaSetValues(). And different values of the Label widget and different entries to application programs of Command widgets are also set accordingly. In this way, the dialog-box is always placed on the screen right beside the corresponding menu box with the proper message.

One characteristic of Xt worth noting here is widget independence. That is, each widget operates to a large degree independently of the application. This is because a widget is an "object" in the Object Oriented Programming sense. Xt dispatches events to a widget, which performs the appropriate actions according to the design of its class, without application help. Widgets handle the consequences when the values of their resources are changed. For example, a Label widget does not depend on the application that created it to determine its size. By default, Label will choose a size large enough to accommodate the current string in the current font. In the case of application changes the text of the Label with a call to XtSetValues, the Label widget itself will attempt to keep its own window large enough to accommodate the current string. This characteristic provides a great convenience to the programmer during the development of dialog-boxes in EGAS.

3.4 Drawing Screen Design

EGAS is an engineering application graphics system; the core of the package is a drawing screen on which the user can create any drawing on.

3.4.1 Bitmap Concepts

Fundamentally, the X workstation's screen is made up of small square discrete picture elements, or pixels. Taken together, the pixels form an array, or raster. All drawing operations ultimately reduce to illuminating the right pixels in the pixel array.

This method of drawing is often called bitmapped graphics.

Of course, it is not required to draw pixel by pixel in X. X provides facilities for drawing objects such as points, lines, rectangles and arcs.

3.4.2 Building a Drawing Screen Based On a Widget

Although there are many standard user interface elements defined in the widget set, application programs often need to call Xlib directly to accomplish certain tasks such as drawing. Xt does not provide its own graphics calls. Thus it is necessary to create a custom window in EGAS, which has features not supported by any existing widget.

It is possible to create a normal X window and draw into it; however, this method loses advantages of Xt, such as Xt's event compression (a technique which allows some events to be ingnored or repacked before they are given to a widget), event filtering (a method which removes events that some widgets can work without it, thus improving performance) and other features. To avoid this, the application should do all its drawings in widgets.

Discussion of the details of how to write application widgets is beyond the scope of this write-up (See reference [2, chapter 5&6] for detail). However, the basic idea is that widgets are never written from scratch. Applications always start from template files that contain the framework for a widget without the specific code, or even better, from an existing widget that has some similar characteristics.

EGAS has built the drawing screen based on a widget called BitmapEdit. The code for the BitmapEdit widget was originally written in an application, and later moved into a widget [2]. While developing the drawing screen, we didn't have to type in the framework of the widget; the only thing we did was to insert our application code into the BitmapEdit widget properly. It is in the application code that graphics function calls to Xlib are found, as well as all the built-in drawing tools. In this way, EGAS has created its special-purpose application window as a widget. The beauty of the widget framework is that the code within it is very modular -- each module has a specific place and purpose. Another obvious advantage of building drawing screen as a widget is that it is easier to process user inputs through Xt event handlers, such as translations (translation tables registered to map event sequences in one widget to actions in another. See reference [2, chapter 7] for more detail).

3.5 On-Line Help

In EGAS, a help widget of type Box is defined for displaying the help text. It is not a Text widget but a Box widget, because the Text widget allows the user to type in characters from the keyboard, just like a normal text editor does, and this is not what HELP is supposed to be like. A Box widget prevents the displayed text from being altered by the user.

It is possible to display the text on the drawing screen, but there is a problem in redrawing the graphics after the system exits from HELP. By defining a separate widget, Xt automatically handles this problem.

The Help window vanishes from the screen when the user presses 'q' on the keyboard. This has been implemented by means of callback. A procedure which pops down the window is registered in the callback list of the Help widget; in this list the condition for executing this procedure is set to 'button press q', so that whenever 'q' is pressed, the condition is meet, the Help window disappears. Since this procedure is only registered for the Help widget, pressing q has no effect on other widgets in the system.

Also, there are several ways to store the text of HELP. A common way to draw text in X is to store the text in a variable of type string, then draw the string on the screen. Instead of hardcoding the text inside the program, EGAS keeps all the help information as ordinary text files in a subdirectory called /Help under the working directory. Each file for a function in EGAS's menus. The file names are contained in an array in the program. By giving the proper index, a required file is loaded into the system through reading each line of the file, and then the text is drawn on the screen.

3.6 The Application-Default File

In Xt, the resource mechanism allows widgets and applications to be customized, such as setting the size and background colour of the application windows, or even changing the callbacks of a widget. Resources may be declared by either a widget or an application, and can be set from any one of several sources, including a user's resource file, the command line, or an application-specific default file.

Each resource of a widget has a default value determined by the widget class that declared the resource. However, in many cases, the application wants a different default value, but still wants the user to be able to change the value of that resource.

The application can provide a default value for resources by hardcoding them in the source code. However, changing these settings would require recompiling the source. A better way to do it is to provide an application-default resource file, which on UNIX systems is usually stored in the directory /usr/lib/X11/app-defaults.

The application-default file used in EGAS is named XEgas (See Appendix A for more detail). With it, not only can the developer change the resources of EGAS easily, but the user can even make changes to the layout of EGAS, such as changing the background colour of the menu boxs, without recompiling the program.

CHAPTER FOUR

GEOMETRICAL TRANSFORMATIONS AND ALGORITHMS

4.1 Analytic Geometrical Calculations of Built-In Functions

As we have mentioned in Chapter One, EGAS is based on a series of built-in functions. In this chapter, we will discuss how those functions have been implemented as mathematical calculations. By looking at a selected set of functions, we will introduce the main two-dimensional geometrical transformations used in EGAS.

4.1.1 Coordinate Representations

EGAS is designed to use Cartesian coordinate systems. Two Cartesian systems are used, one required by the X display, the other is defined for the application program to locate points in the drawing. The coordinates referenced by the application are called world coordinates, and the coordinates used by the X display are called screen coordinates. In both systems, an X coordinate specifies horizontal location and a Y coordinate specifies vertical location. For the world coordinates, the coordinate origin is defined at the lower left corner of the drawing screen, with X values increasing to the right and Y values increasing upwards. For the screen system, the coordinate origin is defined to be the upper left corner of the screen, so that the Y values are inverted. The values of screen coordinates are expressed in number of pixels while the values of world coordinates are based on the units of the drawing. A transformation is performed by EGAS to convert user coordinates to screen values. However, our following discussions about the geometrical calculations will be based on a general Cartesian coordinate system, regardless of the details of the transformation between these two coordinate systems.

4.1.2 Divide Lines Into Equal Portions

Recall DISPLACEMENT-Action 1: CLS + N ; it creates N equally spaced points in each CLS. This has been accomplished as follows.

A line in CLS is specified by its two end points. As shown in Fig 4.0, assume $P_1(X_1, Y_1)$ and $P_2(X_2,Y_2)$ are the two end points of one line in CLS and $Q(U_1,V_1)$ is the first point to be created on this line, nearest to $P_1(X_1, Y_1)$.

Let $dx = (X_2-X_1)/(N+1)$;

$dy = (Y_2-Y_1)/(N+1)$.

We have: $U_1 = X_1 + dx$;

$V_1 = Y_1 + dy$.

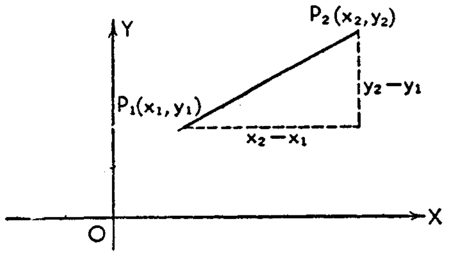


Fig. 4.0 Divide Line by N

Generally speaking, for the N points to be created on the same line, (U_i, V_i) can be calculated by the following algorithm:

$U_0 = X_1$; $V_0 = Y_1$;

```
For i=1 to N do
 begin
   $U_i = U_{i-1} + dx$;
   $V_i = V_{i-1} + dy$;
 end
```

Applying the above algorithm to each line in CLS, we get all of the points which divide CLS into equal portions.

4.1.3 Divide Lines Into Specified Spacing

DISPLACEMENT-Action 2, expressed as CLS + S, is another EGAS drawing function which creates as many points as possible spaced by S on each line in CLS. This has been done based on the following algorithm.

As shown in Fig.4.1, $P_1(X_1, Y_1)$, $P_2(X_2,Y_2)$ are the two end points of a line in CLS, dx is the X increment and dy is the Y increment.

Then we have:

$$dx = S * \frac{(X_2-X_1)}{\sqrt{(Y_2-Y_1)^2 + (X_2-X_1)^2}}$$

$$dy = S * \frac{(Y_2-Y_1)}{\sqrt{(Y_2-Y_1)^2 + (X_2-X_1)^2}} ;$$

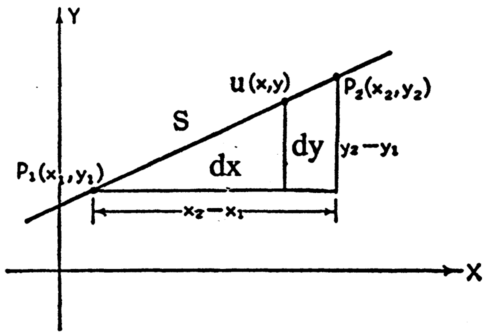


Fig. 4.1 Divide Line by S

therefore, for the points on a line created by this command, the coordinates (U_i, V_i) are calculated by:

```
i = 1;
While ( i*dx <= $(X_2-X_1)$ ) do
  begin
    $U_i = U_{i-1} + dx$ ;
    $V_i = V_{i-1} + dy$ ;
    i = i+1;
  end
```

Applying the above algorithm to each line in CLS, we get all of the points which divide CLS into a specified spacing.

Note that when dx equals to zero, it is the case of the line is vertical and when dy equals to zero, it is the case of the line is horizontal.

4.1.4 Set of Points with Equal X-Y Distance

DISPLACEMENT-Action 5, expressed as CPS + N + X + Y, creates N points distanced by (X,Y) from the preceding one starting with each point of CPS.

Assume CPS[X_i] and CPS[Y_i] represent the coordinates of the ith point in CPS, and two dimensional array P[i,j] contains the points to be generated. Then, one column of P[ ] is calculated by :

```
For j = 1 to N do
```

```
begin
  P[i,j].x = CPS[$X_i$] + i*X;
  P[i,j].y = CPS[$Y_i$] + j*Y;
end
```

Applying the above algorithm to each point in CPS, we will get coordinates of all points in P[i,j].

4.1.5 Points of Intersection of Lines

INTERSECTION is one of EGAS's drawing functions. In general, INTERSECTION creates points at the intersection of two sets of lines.

As we know, if two lines have a common point, the coordinates of this point satisfy the equations of the lines and are found by solving their simultaneous equations. Therefore, to implement INTERSECTION, the first step is to get the equations of the lines.

A line is determined either by two distinct points on it or by one point on it with the line's direction. INTERSECTION in EGAS accepts both kinds of line specification: when CLS+RLS is given, the lines are determined by two points on them; when CLS+CPS+CD, or CLS+RPS+CD is given, the lines are determined by a point and the direction.

Let $P_1(X_1, Y_1)$ and $P_2(X_2,Y_2)$ be the two distinct points on a line, then we get the equation of the line as:

$$\frac{Y - Y_1}{X - X_1} = \frac{Y_2 - Y_1}{X_2 - X_1} \qquad (1)$$

Let $P_1(X_1, Y_1)$ be a given point on the line, and K be its slope (K can be derived from the direction).

We get the equation of the line as:

$$Y - Y_1 = K*(X - X_1) \qquad (2)$$

Both equations (1) and (2) can be converted to the general equation of a straight line with the form: $aX + bY + c = 0$.

Assume $a_1X + b_1Y + c_1 = 0$ represents a line in CLS;

$a_2X + b_2Y + c_2 = 0$ represents a line in RLS,

the solution of the simultaneous equations gives the coordinates of the point at the intersection of these two lines: $X = A/C$; $Y = B/C$, where

$$A = \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix} \qquad B = - \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} \qquad C = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$$

Applying the above solution to each pair of lines in CLS and RLS (or CLS + RPS + CD), we get the coordinates for all the points at the intersections of these two sets of lines. Note that if C were zero it would be impossible to solve for X, Y. This is the case in which the two lines are parallel. It means the two lines do not have an intersection.

4.1.6 An Arc of A Circle

JOIN is an EGAS drawing function which creates lines rather than points, either straight lines or an arc of a circle. In case an arc is created, three points must be specified by the user, since a circle is uniquely determined by three points which are not collinear.

As long as three points $P_1(X_1, Y_1)$, $P_2(X_2,Y_2)$ and $P_3(X_3,Y_3)$ are known, we can find the equation of the circle as follows:

Take the equation in general form: $X^2 + Y^2 + aX + bY + c = 0$

Substituting the coordinates of the three points in this equation we get the three linear equations in the three unknowns a, b, c:

$$aX_1 + bY_1 + c = - X_1^2 - Y_1^2 ;$$

$$aX_2 + bY_2 + c = - X_2^2 - Y_2^2;$$

$$aX_3 + bY_3 + c = - X_3^2 - Y_3^2.$$

The solution of these three equations for a, b and c will yield the coefficients of the equation of the circle:

$a = A/D$; $b = B/D$; $c = C/D$, where

$$D = \begin{vmatrix} X_1 & Y_1 & 1 \\ X_2 & Y_2 & 1 \\ X_3 & Y_3 & 1 \end{vmatrix}; \quad A = \begin{vmatrix} d_1 & Y_1 & 1 \\ d_2 & Y_2 & 1 \\ d_3 & Y_3 & 1 \end{vmatrix}; \quad B = \begin{vmatrix} X_1 & d_1 & 1 \\ X_2 & d_2 & 1 \\ X_3 & d_3 & 1 \end{vmatrix};$$

$$C = \begin{vmatrix} X_1 & Y_1 & d_1 \\ X_2 & Y_2 & d_2 \\ X_3 & Y_3 & d_3 \end{vmatrix}; \qquad \begin{aligned} d_1 &= - X_1^2 - Y_1^2 ; \\ d_2 &= - X_2^2 - Y_2^2 ; \\ d_3 &= - X_3^2 - Y_3^2 . \end{aligned}$$

Note that if D were zero it would be impossible to solve for a, b, c. This is the case in which the three points are collinear.

The standard form of the equation of a circle is then derived from the general form as follows:

$$\left(x + \frac{a}{2} \right)^2 + \left(y + \frac{b}{2} \right)^2 = r^2$$

This is the equation of a circle with centre at (-a/2, -b/2) and radius

$r = (1/2)*\sqrt{a^2 + b^2 - 4c}$

as shown in Fig. 4.2.

As long as the circle is defined, the arc which connects the three given points is defined as part of it.

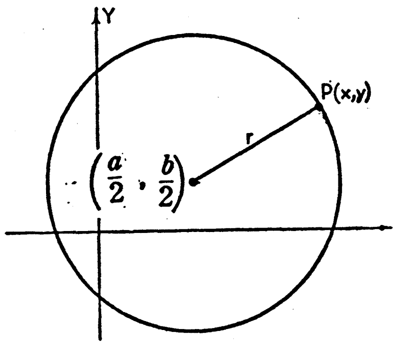


Fig.4.2 A Circle

Note that for most of the calculations described above, there is a "round-off" problem, since coordinates can be calculated only as integers. To solve this problem, we used variables of type "double" during the calculation, and convert them back to integers just before we pass them to Xlib functions to perform the drawing. The conversion used is as follows:

double X, Y;

integer x, y;

$$x = \lfloor X + 0.5 \rfloor; \qquad y = \lfloor Y + 0.5 \rfloor;$$

4.2 Save Drawing and EGAS File Format

One very important fact about X should be mentioned before we start to talk about saving drawings in EGAS. Once the workstation handles an application's request to draw an object (such as a line), the workstation has no memory of the fact that a line is drawn, only that certain pixels are "lit". Once the drawing is complete, the workstations's memory of what was drawn is contained entirely in the pixel raster. Therefore, the application program must establish its own method to memorize and save the created drawing.

A common way to do this is to record the drawing in terms of drawing operations. A procedure specification is added to a display list each time a drawing operation is performed, and each element in the list can be stored into a file later on when a save function is required. EGAS uses a similar but simpler way for saving drawings, as explained below.

A data record is defined corresponding to each graphical primitive in EGAS. While creating a drawing, instead of procedure specifications, it is the geometrical attributes of the primitives which are added to appropriate records. The following data records are introduced by EGAS programs:

Points_Created:	contains X,Y coordinates for all the created points;
Lines_Created:	contains points in pairs which are end points of created lines;
Arcs_Created:	contains lists of three ordered points which determine arcs.

There are also three records named **Points_Deleted, Lines_Deleted** and **Arcs_Deleted** in EGAS; they have the same structures respectively, as those specified above and are used for recording graphical primitives when any deletion is made.

Function SAVE stores data in those records into a file. Function LOAD reads them from the file and calls EGAS procedures to redraw the display on the screen. When records are loaded, the coordinates are passed as parameters to X routines which perform the required creation/deletion operations.

Note that data in records are expressed in terms of screen coordinates, which are integers. The data of a drawing stored in a file are in terms of world coordinates; they are also integers in the current system, but they are not necessarily integers in general. It depends on the units and accuracy of a drawing, which may vary for different drawings and are specified by the user. Transformations are made before writing data to and after reading data from a file.

An advantage of doing this is that it is convenient to retrieve a drawing from world coordinates format into system according to current units and accuracy, especially when inserting a drawing into specific place with certain scaling factor. Another advantage is that the saved drawing is hardware independent.

CHAPTER FIVE

DISCUSSION

Until now, we have introduced our software package EGAS, an engineering graphics application system, in terms of three aspects. At first, we presented the main features of EGAS, and then we talked about the programming techniques involved in this project (including X concepts) followed by discussions of the mathematical calculations and algorithms used in EGAS. In this chapter, we will present some examples of the execution of EGAS and then give several suggestions about the future work.

5.1 EGAS Examples

All of the EGAS features mentioned in Chapter 2 have been implemented and tested except the one for defining units and accuracy. Following are some drawings created by EGAS.

As we have said in the introduction of this report, a special feature of EGAS is that it allows the user to create or delete a set of points or lines simultaneously. This characteristic makes EGAS very efficient for building structural drawings. Fig. 5.0 is a drawing of a window. The entire drawing contains 30 points and 18 lines, and is created in only 7 steps by EGAS.

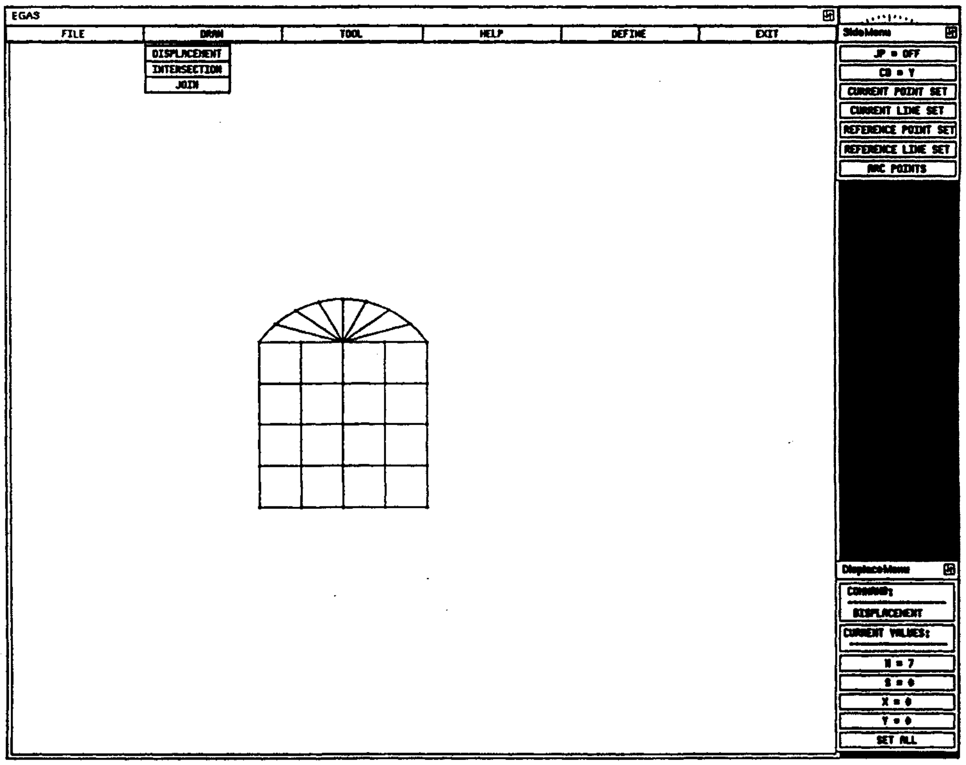


Fig. 5.0 An EGAS Drawing -- Window

The seven steps are:

1. Create the point at the bottom left corner of the window by clicking the middle mouse button;
2. Create the rest of the points in the bottom line using DISPLACEMENT with JP = ON;
3. Create the rest of the points in the square frame using DISPLACEMENT with JP = ON;

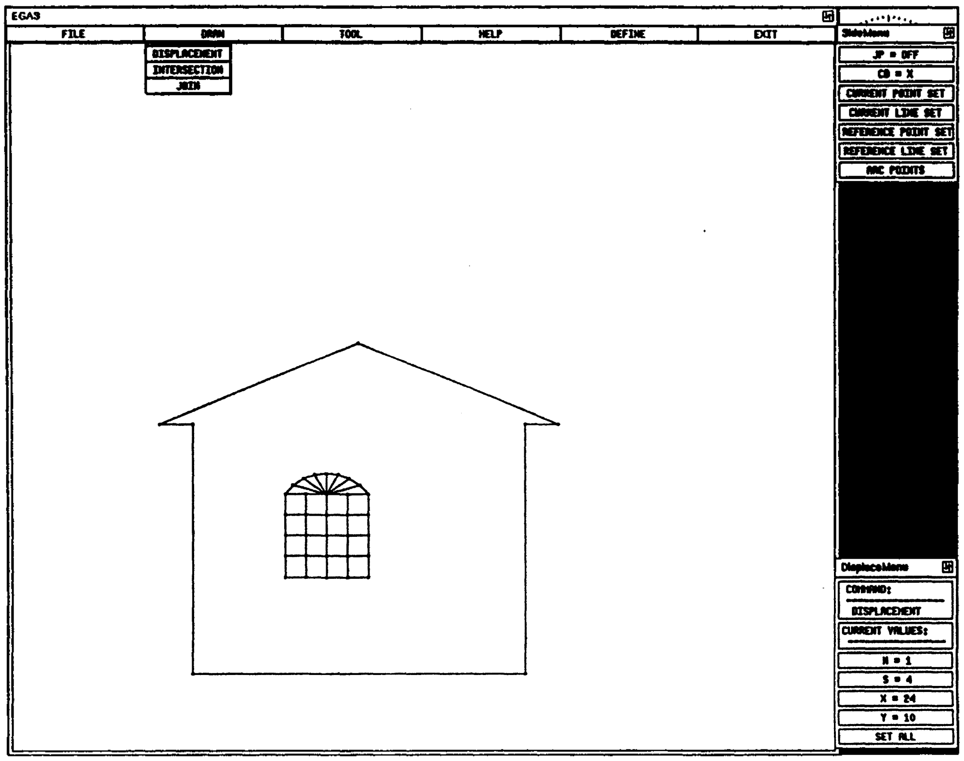
4. Create a single point at the top of the window using DISPLACEMENT with JP = OFF;

5. Create the arc by specifying APS, and then using JOIN;

6. Create the points on the arc using DISPLACEMENT;

7. Specify the centre point on the top line as RPS (repeatedly counted); and JOIN CPS and RPS.

Fig. 5.1 is an example of function INSERT; the window is a drawing saved in a file and then loaded into the current drawing -- a house.



JP = OFF
CD = X

COMMAND:
DISPLACEMENT

CURRENT VALUES:
N = 1
S = 4
X = 24
Y = 10

Fig 5.1 An Example of INSERT -- A House

Another example of the use of INSERT is shown in Fig. 5.2. Each bridge in this Fig. was separately created and saved, and then loaded into the current drawing using INSERT.

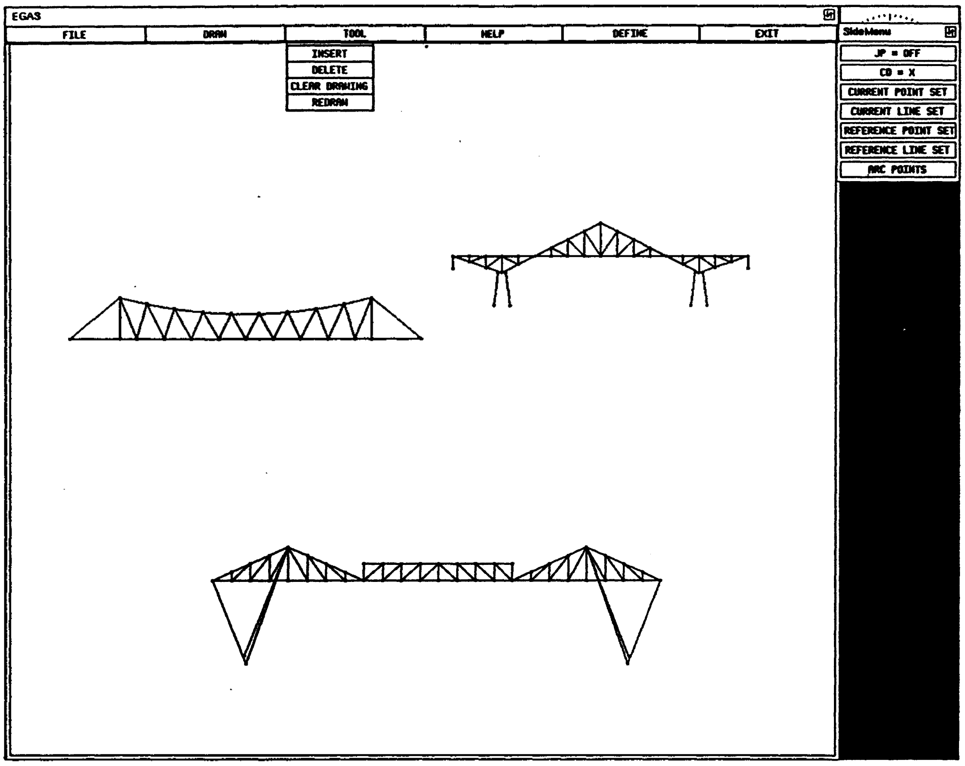


Fig. 5.2 Another Example of INSERT -- Bridges

5.2 Friendly User Interface

Providing the user with a flexible way to communicate with the system is another goal of EGAS. This has been done by implementing nested menus in such a way that all menus which are visible on the screen are accessible by the user, no matter which level the menu belongs to. In addition to allowing the user to communicate with EGAS, our system allows the user to directly communicate with the X environment during the running of EGAS.

As we have mentioned in Chapter 2, the user can access all the utilities provided by X itself while running EGAS, such as printing windows or doing a screen dump through the X program Xdpr. An obvious advantage of this is any EGAS drawing can be printed out. All of the figures used in this report are generated in this way.

5.3 Recommendations for Future Work

Further development of EGAS is planned.

At present, EGAS does not provide the user with the option to define unique units and accuracy for each drawing. Ideally, EGAS will accept a unit specification either in the IMPERIAL or the METRIC system, and for either of them, accuracy could be specified as either a decimal fraction (e.g., 0.001, 0.005) or a ratio (e.g., 1/10, 1/100, 3/16 etc.). When a unit is defined, the scale factor used in the 2transformation between world coordinates and screen coordinates is determined. As long as the units and accuracy are defined, the base increment is determined, which will be the minimum length the user can specify. For example, if the unit is defined as feet and the accuracy is 1/48, then the base increment would be 1/4 inch, which means the smallest length the user can refer to is a quarter inch in this drawing.

The entries for unit and accuracy have already been set in the menu frame in

EGAS. The two items under menu box DEFINE are two independently defined widgets, one for handling units, another for accuracy. It should not be very hard to add DEFINE function to EGAS, since the development of user interface and application code are separated from each other. What needs to be done is to write the programs which perform the appropriate mathematical conversion, and then add them as callbacks to the corresponding widgets.

Another line of further development is to extend the existing EGAS functions to three dimensions. Thus, coordinate Z will be brought into the system. For example, DISPLACEMENT-action 5: CPS + N + X + Y in the current system will then become CPS + N + X + Y + Z. The expressions of most functions may not be changed, but the meaning of them would be extended. for instance, DISPLACEMENT-action 4: CPS + CD + N + S, will not only create points in a plane but also can create points in 3D space.

An advanced software environment is required for the support of 3D graphics. One common package is PHIGS, Programmer's Hierarchical Interactive Graphics System. Different versions of PHIGS are available, such as Sun PHIGS and MIT PHIGS, both of them are completely integrated with X (both Xlib and the XToolkit), which means that the user interface we have built and the widgets we have used in the current system will still work fine with PHIGS. Major work will be given on building a mathematical transformation system which deals with 3D graphics.

Also, it would be good to introduce labels for nodes and members in EGAS drawing. Data structures will be defined along with this, therefore we will have node and member files for each drawing. And then, a node in a drawing can be specified by either using screen pointer or referring its label.

Finally, it would be useful to add scaling and rotation function to EGAS. We will again leave this for future development.

REFERENCES

[1] Oliver Jones: Introduction to the X Window System. Prentice Hall, Inc., 1989.

[2] Adrian Nye and Tim O'Reilly: X Toolkit Intrinsics Programming Manual (Second Edition). O'Reilly & Associates, Inc., 1990.

[3] Eric F. Johnson and Kevin Reichard: X Window Applications Programming. Management Information Source, Inc., 1989.

[4] Valerie Quercia and Tim O'Reilly: X Window System User's Guide. O'Reilly & Associates, Inc., 1989.

[5] James D. Foley and Andries Van Dam, etc. : Computer Graphics, Principles and Practice. Addison-Wesley Publishing Company, Inc., 1990.

[6] Donald Hearn and M. Pauline Baker: Computer Graphics. Prentice Hall, Inc., 1986.

[7] A. Robson: An Introduction to Analytical Geometry. Cambridge University Press, 1940.

[8] T. W. Barber and Rolt Hammond: Civil Engineering Design. The Technical Press Ltd., 1955.

[9] Shu Shan Li: Graph Perfect. M.Sc. Degree Project Report. Department of Computer Science and Systems, McMaster University, 1987.

APPENDIX A

HOW TO INSTALL EGAS

It is assumed that X has already been installed on your system. Version X11R4 or higher one is required, and all X toolkit functions are available. In addition, although X runs on many different types of systems, EGAS assumes that you are running it on a UNIX system.

There is a file named EGAS.tar.Z on the EGAS program disk. It is a compressed tar archive. To install EGAS, the first step is loading egas.tar.Z into your system by using kermit facility. To restore files once retrieving the archive, type:

```
uncompress egas.tar.Z
tar xf egas.tar
```

All the files contained in EGAS are installed in a subdirectory called Graph under current directory. A make file is included.

There is a application-default file named XEgas under directory Graph. It is required to set the application-default file in the system application-faults directory (usually /usr/lib/X11/app-defaults on UNIX system). If you have permission to write to that directory, you can copy it there yourself. Otherwise you can set the XAPPLRESDIR environment variable to the complete path of the directory where you installed the default

file. EGAS will not function properly without the application-default file.
A readme file under the same directory provides instructions of how to set EGAS system and the environment properly.

To run EGAS, type:

```
egas
```

followed by RETURN.

APPENDIX B

HELP FILES OF EGAS

EGAS IS AN ENGINEERING GRAPHICS APPLICATION SYSTEM

There are two modes in the system. One is COMMAND mode, the other is HELP mode. The system is in COMMAND mode by default. A click on the HELP box in the main menu changes the system to HELP mode. Pressing 'q' changes back to COMMAND mode.

1. COMMAND MODE:

Clicking on a choice in the main menu brings the specified sub-menu up.

Clicking on a choice in the sub-menu causes the system to perform the specified functions.

2. HELP MODE:

Clicking on a menu pane brings out the on-line help document for the specified function.

(Current Mode Is HELP)

Press 'q' To Quit Help

FILE

There are three commands in FILE:

1. NEW;
2. LOAD;
3. SAVE.

(Current mode is HELP)

Press 'q' to quit HELP

NEW

Starts a new drawing.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

LOAD

Retrieves an EGAS data file specified by the user, and displays the drawing on the screen. Further modification of the structure can then be made.

The user is prompted to enter the file name.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

SAVE

Stores the drawing as an EGAS data file.

The user is prompted to enter the file name.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

DRAW Provides Graphics Creation Functions

1. Clicking on the middle button of the mouse creates a single point.
2. There are three commands in DRAW:

   2.1 DISPLACEMENT;

   2.2 INTERSECTION;

   2.3 JOIN.

All commands involve using system parameters which can be specified by clicking on the Side-Menu in COMMAND mode. The Side-Menu is at the top right corner of the screen, with pink menu bars.

Click on the side-menu to get information about the system paramenters.

Click on the sub-menu to get information about the use of the commands.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

DISPLACEMENT

Creates points and lines. When EGAS asks for a displacement, it needs system and command parameters. Different combination of parameters gives different displacement.

There are seven cases:

1. CLS + N :	N equally-spaced points in each CLS;
2. CLS + S :	As many points as possible spaced by S on each CLS;
3. CLS + N + S :	N points spaced by S on each CLS; it has a "follow-on" feature which allows a subsequent N, S pair;
4. CPS + CD + N + S:	N points spaced by S in CD from each CPS; also has a "follow-on" feature;
5. CPS + N + X + Y :	N points displaced by (X,Y) from the procceding one starting with each CPS;
6. APS + N:	N equally-spaced points on an arc which is defined by APS;
7. APS + S:	As many points as possible spaced by S on an arc which is defined by APS.

Click on the side-menu to get information about the system parameters.

Click on the DisplaceMenu to get information about the command parameters.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

INTERSECTION

In each case, creates points at the intersection of every current line with every reference line. (All lines are straight.) Involves using system parameters CLS, CPS, RPS and CD.

There are cases:

1. CLS + RLS ;
2. CPS + CD + RLS;
3. CLS + RPS + CD;

To run INTERSECTION, first specify the required parameters, then click on the INTERSECTION menu bar.

Click on the side-menu to get information about system parameters.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

JOIN

Joins points to form straight lines or an arc of a circle.

There are two cases:

1. CPS + RPS : Current points and reference points are joined by straight lines, first CPS to first RPS, second CPS to second RPS, etc.

2. Arc Points : Three points are joined by an arc of a circle whose centre and radius are defined by the three points.

To run JOIN, first specify the point set, then click on the JOIN bar.

Click on the side-menu to get information about how to specify the point set.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

TOOL

EGAS provides graphics manipulation tools. They are:

1. DELETE;
2. INSERT
3. CLEAR;
4. REDRAW.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

<u>DELETE</u>

Deletes part of the drawing.

There are three cases:

1. Delete Current Point Set (CPS);
2. Delete Current Line Set (CLS);
3. Delete an arc of a circle.

To Run the DELETE:

1. Specify CPS, CLS or Arc Points using the mouse and the side-menu;
2. Click on DELETE to perform the deletion.

Click on the side-menu to get information about how to specify CPS, CLS or APS.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

<u>**INSERT**</u>

Inserts a drawing from a file into current drawing. The drawing contained in the specified file is copied into the current drawing with its size reduced and its upper left corner placed at the insertion point.

How to run INSERT:

1. Specify a point as CPS, which will be used as insertion point;
2. Click on INSERT, enter file name and a scale factor when prompted;
3. Click on OK box to run INSERT, or CANCEL box to cancel the selection.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

<u>**CLEAR DRAWING**</u>

Clears the screen.

<u>**REDRAW**</u>

Redraws the previous drawing which is erased by CLEAR DRAWING.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

JP = Join Point

When JP is on, every new created point is automatically joined to the previous created point.

When JP is off, only points are created.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

CD = Current Direction

One of the system paramenters. It is required by the function DISPLACEMENT.

EGAS provides four directions: X, -X, Y, -Y.

The value on the menu bar is the current direction. It is is changed by clicking on the menu bar while in command mode.

(Current Mode Is HELP)

Press 'q' To Quit HELP!

<u>CPS = Current Points Set</u>

One of the system parameters which may be required for creation or deletion of points or lines.

How To Specify:

1. Move the cursor to the desired position;
2. Click the left button of the mouse to pick up each point;
3. Repeat the same operation for another point until the whole set is selected;
4. Click the CPS menu bar to end the specification.

Notes:

1. The latest created point set is the CPS by system default if no CPS is specified;
2. The selection will have no effect if no point exists at the selected position.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

<u>CLS = Current Line Set</u>

One of the system parameters which may be required for creation or deletion of points and lines.

How To Specify: A line is selected by specifying the two end points of the line.

1. Move the cursor to the desired position;
2. Click the left button of the mouse to pick up the end points of a line;
3. Repeat the same operation for another line until whole set is selected;
4. Click the CLS menu bar to end the specification.

Note: The selection will have no effect if no line exists at the selected position.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

<u>**APS = Arc Point Set**</u>

APS contains three points which determine a circle. An arc of the circle identified by these three points is defined.

How To Specify:

1. Move the cursor to the desired point;
2. Press the left button of the mouse to pick one point;
3. Repeat the same operation for the other points until all three points are selected;
4. Click the Arc Points menu bar to end the specification.

Notes:

1. The three points must be selected in clockwise order around the arc;
2. Only the first three points are counted if more than three points are picked up.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

<u>RLS = Reference Line Set</u>

One of the system parameters. It is required by the function INTERSECTION.

How To Specify: A line is selected by specifying the two end points of the line.

1. Move the cursor to the desired position;
2. Click the left button of the mouse to pick up the end points of a line;
3. Repeat the same operation for another line until the whole set is selected;
4. Click the RLS menu bar to end the specification.

Note: The selection will have no effect if no line exists at the selected position.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

<u>RPS = Reference Point Set</u>

One of the system parameters which may be required by functions DISPLACEMENT, INTERSECTION and JOIN.

How To Specify:

1. Move the cursor to the desired position;
2. Click the left button of the mouse to pick up each point;
3. Repeat the same operation for another point until the whole set is selected;
4. Click the RPS menu bar to end the specification.

Notes: The selection will have no effect if no point exists at the selected position.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

<u>N, S, X, Y</u>

There are parameters which are required by DISPLACEMENT command.

N : Number of points or lines to be created;

S : Spacing of between points or lines being created;

X : X distance;

Y : Y distance.

How To Specify:

1. Click on DisplaceMenu in COMMAND mode causes a dialog-box to appear on the screen;
2. If a value is already displayed in the dialog-box, press "BACKSPACE" to delete it. Then entering a value from the keybord causes the value to be displayed on the dialog-box;
3. To register the value, click on the OK box; to cancel the value, click on the CANCEL box.

(Current Mode Is HELP)

<u>Press 'q' To Quit HELP!</u>

< THE END >