GNA95GP V2: AN ADA 95 GRAPHICS PACKAGE FOR WINDOWS 95

By

YANG HUANG

Bachelor of Science

Fudan University

Shanghai, China

1989

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1997

GNA95GP V2: AN ADA 95 GRAPHICS PACKAGE

FOR WINDOWS 95

Thesis Approved:

[illegible]

Thesis Adviser

J Chandler

[illegible]

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major adviser. Dr. K. M. George for his intelligent supervision, constructive guidance, inspiration and friendship. My sincere appreciation extends to my other committee members Dr. John P. Chandler and Dr. George E. Hedrick, whose guidance, assistance, encouragement, and friendship are also invaluable. I would like to thank Dr. K. M. George and Department of Computer Science for providing me with this research opportunity and their generous financial support.

More over, I wish to express my sincere gratitude to those who provided suggestions and assistance for this study: Mrs. Lan Li, Mr. Shan Kuang, and Mrs. Qiuye Wang.

I would also like to give my special appreciation to my wife, Jun Fu, for her strong encouragement at times of difficulty, love and understanding throughout this whole process. Thanks also go to my parents for their support and encouragement.

Finally, I would like to thank the Department of Computer Science for supporting during these two years of study.

This research project is supported by DISA DCA 100-96-10007.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER I

Introduction

Today, computer graphics is widely used in several different areas. Many graphics libraries and packages are available in public or proprietary domains. Why do we still need to develop GNA95GP, a 2D graphics package? Several factors such as programming language binding and implementation platform are related to graphics packages. GNA95GP provides a program library which is not previously available. It is based on Ada 95, a new programming language and MS Windows environment.

Based on this author's experience, the graphics packages available in the public domain are targeted for specific applications, either designed to be used with a single high-level programming language, or difficult for novices to learn. In order to use general purpose graphics libraries, such as X-Windows and MS Windows' GDI, one must learn to do window programming and must know the functions provided in these libraries. In contrast, GNA95GP is easy to learn. It absorbs the Windows overhead so that users can focus their attention on doing graphics work.

Most graphics packages available in the public domain are written in the C language or the Pascal language, such as standard graphics package Graphics Kernel System (GKS) and Programmer's Hierarchical Interactive Graphics System (PHIGS).[8, 21] Although Ada 95 is used as a primary programming language in many application areas where graphics application, such as simulation and visualization, can be found, there is no

Ada 95 graphics package available in the public domain. To this author's knowledge, GNA95GP is the first graphics package using the Ada 95 programming language.[23]

Targeted for undergraduate computer graphics courses, GNA95GP would be a useful tool. It is a free graphics package, and it is built on free software. Now GNA95GP for MS Windows 3.x is available at the World Wide Web site *http://www.cs.okstate.edu/gna95gp*[13].

After Microsoft released Windows 95, many previous Windows 3.x users upgraded their PCs' operating system to Windows 95 to take advantages of the new system. Now, Microsoft 32-bit Windows systems (Windows 95 and Windows NT) have become the standard operating systems for new IBM compatible PCs. This thesis is to upgrade GNA95GP from Windows 3.1 to Windows 95 (also to Windows NT, because the Windows development kits on which GNA95GP is based is a Win32 applications develop environment for Windows NT and Window 95. Since Windows NT operating system is not available for this research, the upgraded software has not been tested in Windows NT environment.) A secondary objective is to enhance this package under the new operating systems.

CHAPTER II

Literature Review

2.1 Review of Standard Graphics Package

Several standard graphics packages have been developed since the first graphics standard package GKS (the Graphics Kernel System) was released. Two well-known ones are GKS and its extension GKS-3D, and PHIGS (Programmer's Hierarchical Interactive Graphics System) and its extension PHIGS+. There are also some unofficial standard graphics packages, called industry standard packages, such as Adobe's PostScript, Silicon Graphics' OpenGL, and X-Consortium's X Window System.[10, 11] One of the advantages of a standard package is that it allows the application programmer to concentrate on projects without wasting time on low-level functions related to interactive graphics programming. Brief reviews of standard graphics packages are presented below:

2.1.1 GKS (Graphics Kernel System)

Graphics Kernel System (GKS) is the first standard graphics package. It is an ANSI (American National Standards Institute) standard graphics library, and is a superset of the ISO (International Organization for Standardization) graphics library. GKS adopts a layered architectures. Figure 1 presents the layer model of GKS. As we can find in this model, each layer can call the functions from a lower adjoining layer.

The top level of the model is the language-independent application interface. It is defined by the GKS standard. The interface between the application layer and the language-dependent layer is the language-dependent application layer (application oriented layer). As a language independent standard, GKS has bindings to several programming languages including FORTRAN, Pascal, C and Ada[2]. An important part of GKS is graphical resources workstations. They are graphical output devices and input devices. Each workstation has a device driver, also called workstation driver, which translates the device-independent representations of functions within the nucleus to and from the workstation-specific representations. The main features are summarized below. They are based on the works of G. Enderle, K. Kansy, G. Pfaff, F. R. David, F. James, J. Leisy, and F Andries[8, 7, 18, 19].

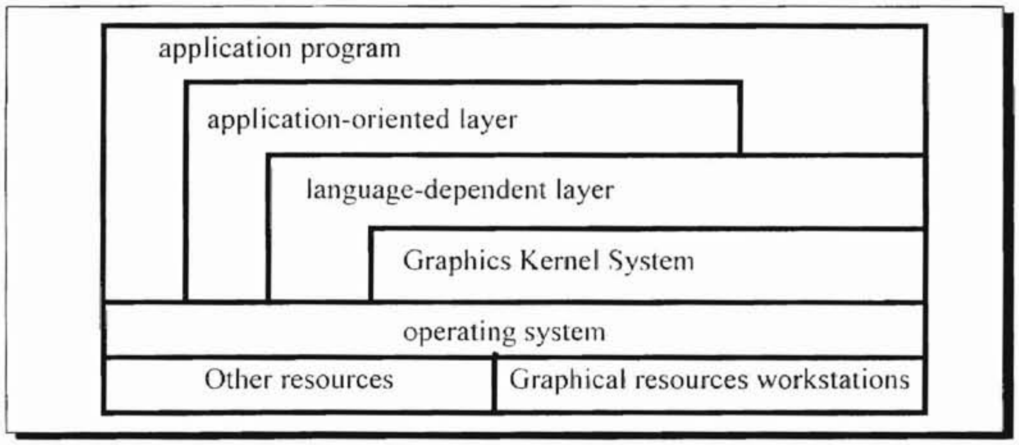

Figure 1: Layer model of GKS (adopted from [8]).

Graphics Primitives:

- **Output Primitives:** GKS provides six output primitives. They are POLYLINE (straight lines connecting a given set of points), POLYMARKER (a specified symbol centered on given positions), TEXT (character string starting from given position), FILL AREA (a polygon which can be hollow or filled with the specified uniform color, pattern, or hatch style), CELL ARRAY (an array of rectangular cells with individual colors), GENERALIZED DRAWING PRIMITIVE (GDP, special geometrical output capabilities of a workstation, such as spline curves, circular arcs, and elliptic arcs).

- **Logical Input Devices:** GKS has six logical input devices. They are LOCATOR (specify a positions by world coordinates), STROKE (a sequence of position in world coordinates), VALUATOR (input a value of type real), CHOICE (a non-negative integer selected from a set of alternatives), PICK (identify a display object, such as segment name or pick identifier), and STRING (input a character string).

Coordinate System:

GKS provides three coordinate systems, world coordinates (WC, device-independent Cartesian coordinates), normalized device coordinates (NDC, device-independent intermediate coordinate system, normalized into a range which is usually 0 to 1 in GKS), device coordinates (DC, device-dependent or workstation-dependent coordinate system).

GKS Workstations:

Graphics workstations in GKS are an abstraction of physical devices. They are divided into six categories, output workstation, input workstation, output/input workstation, workstation-independent segment storage, metafile output, and metafile input.

Segment:

GKS provides segments to group output primitives together in an appropriate data structure, which can be addressed and manipulated as a unit. Each segment is identified by a name specified by the application. Operations applied on a segment are *open, close, turn visible on and off, transform (scale, translate, rotate), copy, select, rename, highlight on and off, insertion and delete*.

Operating Modes:

All workstations can be used through three interaction modes, REQUEST (device is read only), SAMPLE (return current workstation input value without waiting for any action), EVENT (insert user input value into a queue, if a device gets input from user).

Metafile:

A record of calling graphics GKS functions which generate graphics output primitives. In GKS, a metafile is treated as a workstation which contains a sequence of items. Each item has three components, item type, item data record length, and item data record.

2.1.2 PHIGS and SPHIGS

Programmer's Hierarchical Interactive Graphics System (PHIGS) is another ANSI and ISO standard graphics package. PHIGS is designed to support highly dynamic and interactive computer graphics applications. PHIGS provides a hierarchical graphics database. The significance of this capability is that the elements of the database can be edited while they are being displayed. Such functionality is highly desirable in several different areas of applications, such as CAD or CAM (computer aided manufacturing), or command_control system. It supports several platforms. It has a variety of programming language bindings, such as FORTRAN, C and Ada.[26, 15, 29, 16, 30, 17]

SPHIGS (Simple PHIGS) is a subset of PHIGS. It has most of PHIGS's capabilities and power, but it simplifies or modifies several features.[11] SPHIGS uses a 3D floating-point world-coordinate system (so is PHIGS), rather than a 2D integer coordinate system. Therefore, it does not support direct pixel manipulation. This is one of the three major differences between SPHIGS and integer raster packages, such as SRGP[11]. The other two main differences are that SPHIGS implements the 3D viewing pipeline and maintains a database of structures, which group primitives, attributes and other information into a unit. The main features of SPHIGS are listed below[11, 7, 18, 19, 26]:

CSS (Central Structure Storage)

Central Structure Storage (CSS) is one of the significant features of SPHIGS. CSS stores a sequence of elements — primitives, appearance attributes, transformation matrices, and invocations of subordinate structures — in order to define a self contained geometric object. To store elements in CSS, an entry in CSS is created to add a new structure into CSS using a function call — **void SPH_openStructure(int** structureID). All elements specified between this function call and **void SPH_closeStructure**() function call are stored under the entry structureID in CSS. Once a structure is closed, it can be reopened for editing. At any time, only one structure can remain open. There are two additional properties of a structure. One is that the structure IDs are integers. The other is that all primitives and attributes appear only as elements of a structure. A structure either is empty or contains as many elements as required to group into one object. SPHIGS just records the structure. It does not display the structure, until the application calls **void** SPH_postRoot(**int** structureID, **int** viewIndex). This function call causes SPHIGS to perform a depth-first display traversal of the structure's elements in the CSS, executing each element in order from the first to the last. In addition, **void** SPH_unpostRoot (**int** structureID, **int** viewIndex) can be used to remove the root from the list of posted roots without deleting the entry from CSS. The information recorded in CSS helps SPHIGS to update the screen image in order to match the current status of the CSS and view table. An application either lets SPHIGS regenerate screen image automatically or call SPH_regenerateScreen() to ask a screen regeneration explicitly. Function **deleteStructure** deletes an entry or sub-entry from CSS.

Advantages and Limitation of CSS

There are three major advantages to CSS. The first is that whenever any change in CSS is made by the application, SPHIGS automatically regenerates screen image to match current status of the CSS and view table. The second is automatic pick correlation. It helps SPHIGS to determine the primitive within CSS and its relative position selected by the user. Once a structure is created in CSS, it can be edited later. The third advantage of CSS is that is provides the facility for user application to create various dynamic effects easily.

A storage facility such as CSS is neither necessary nor sufficient under some cases. Applications can do screen regeneration by themselves. Some times, it is necessary to keep all appropriate data for each object in another separate data structure. One of the reasons is that application can not manipulate CSS as it does with structures defined by itself. There are some limitations for application to query the CSS. Records in data structures other than CSS and records in CSS must be synchronized properly. It is overhead. For applications with significant structural changes between successive images, such as simulation of molecular movement, maintaining CSS is insufficient. So, many implementations of graphics packages do not have any type of structure storage, such as Windows GDI (will be discussed later). Some packages provide an option to use retained-mode (store the primitives in storage data structure first, then display them), or immediate mode (not to keep any record, and display the primitives directly). The others combine these two modes (application can keep some primitives' records in storage data structure, while display the others directly).[11] The new GNA95GP version provides the last type of storage structure.

Output and input features of SPHIGS are similar to those of GKS-3D. Output features are POLYLINE, POLYMARKER, TEXT, FILL AREA, FILL AREA SET, CELL ARRAY, GENERALIZED DRAWING PRIMITIVE (GDP). Input features are LOCATOR, STROKE, VALUATOR, CHOICE, PICK and STRING. It also uses REQUEST, SAMPLE and EVENT to handle interaction.

2.1.3 Simple Raster Graphics Package (SRGP)

As a device-independent graphics package, SRGP is designed as a raster graphics package. Its output features are adopted from Apple's QuickDraw integer raster graphics package and MIT's X Windows System. Also, the idea of input features come from GKS and SPHIGS.[11] SRGP is implemented for X Windows and PCs, using PASCAL and C.[11]

The drawing primitives of SRGP, which are similar to QuickDraw and Xlib package of X Windows, are line, marker, polygon, circle, ellipse, and text. The origin of its coordinate system is at the left bottom of the screen; x-coordinate increases from the left to the right, and y-coordinate goes up from the bottom to the top. Drawing attributes of SRGP for line are width and style. Attributes for area are filling style, filling bitmap pattern, filling pixmap pattern, and background color. Attributes for marker are style and size. Attribute for text is font. There are two modes to handle interaction, sampling and event.

Although SRGP is simple and easy to learn, and powerful for a large class of 2D graphics applications, it has some limitations. First, SRGP uses integer coordinate system. Many applications use floating-point world coordinates to store geometrical information. SRGP does not provide any scheme for mapping world coordinates to display device. Another important drawback is that SRGP doesn't keep any record of the drawn primitives. Because of this shortcoming, in some cases, for example, when a message box is removed, applications must respond to re-specify part or the entire set of primitives to SRGP.

2.2 Review of Graphics Device Interface (GDI) of MS 32-Bit Windows

Graphics Device Interface (GDI) is an integer raster graphics package of MS Windows. As a subsystem of Windows operating system, GDI provides many functions to display graphics on video displays and printers. User applications use GDI to display visual information, and they also use these to exhibit user interface, such as menus, scroll bars, icons, and mouse cursors.

Windows 95 was introduced in August 1995. One of the significant differences between Windows 95 and its previous version, Windows 3.x, is that Windows 95 is a 32-bit operating system, like Windows NT, another 32-bit window operating system of Microsoft. Due to this difference, some functions are dropped while others are modified. Although some new features are added to 32-bit Windows system, the core of GDI hasn't been changed since it was released with Windows 1.0[25]. In some ways, GDI is a high-level graphics interface language, with the capability to directly manipulate graphics devices.[25] GDI provides communication between application and device driver. It provides device-independent interface for applications while interacting with devices in a device-dependent way.

2.2.1 Coordinate System

Windows has three coordinate systems, screen coordinates (refers to the whole screen), complete windows coordinates (refers to an entire application window, includes title bar, menu, scroll bar, statue bar, and window frame), and client-area coordinates (area of application's windows except title bar, menu, scroll bar, statue bar, and window frame, the one with which we most often work). Unlike a graphics package using floating-point numbers for virtual coordinates, Windows uses signed 32-bit integer. Coordinate space of Windows is based on Cartesian coordinate space. There are two space types, logical space and physical space. One unit in logical space may represent a pixel, 0.1 mm, 0.001 mm, 0.01 inch etc. in real world, (depending on the mapping mode), but the coordinate parameters passed into GDI functions are integers (number of units). Applications use coordinate space and transformations to scale, rotate, translate, shear, and reflect graphics output.

There are four spaces used by Win32 API (Application Programming Interface), world space, page space, device space, and physical device space. World space and page space are logic spaces. World space is usually used by applications to rotate, shear, and reflect graphics output. Page space and device space work together to provide a device-independent view of the graphics hardware for applications. The back-end space of those four coordinate spaces is physical device space. It usually refers to the client part of application's window (or complete window), entire desktop, or a page of a printer. Applications specify output primitives in logic space. Using transformation algorithm, Windows maps those primitives to next coordinate space one by one until primitives are finally displayed on physical device. Figure 2 represents the relation of those coordinate spaces.

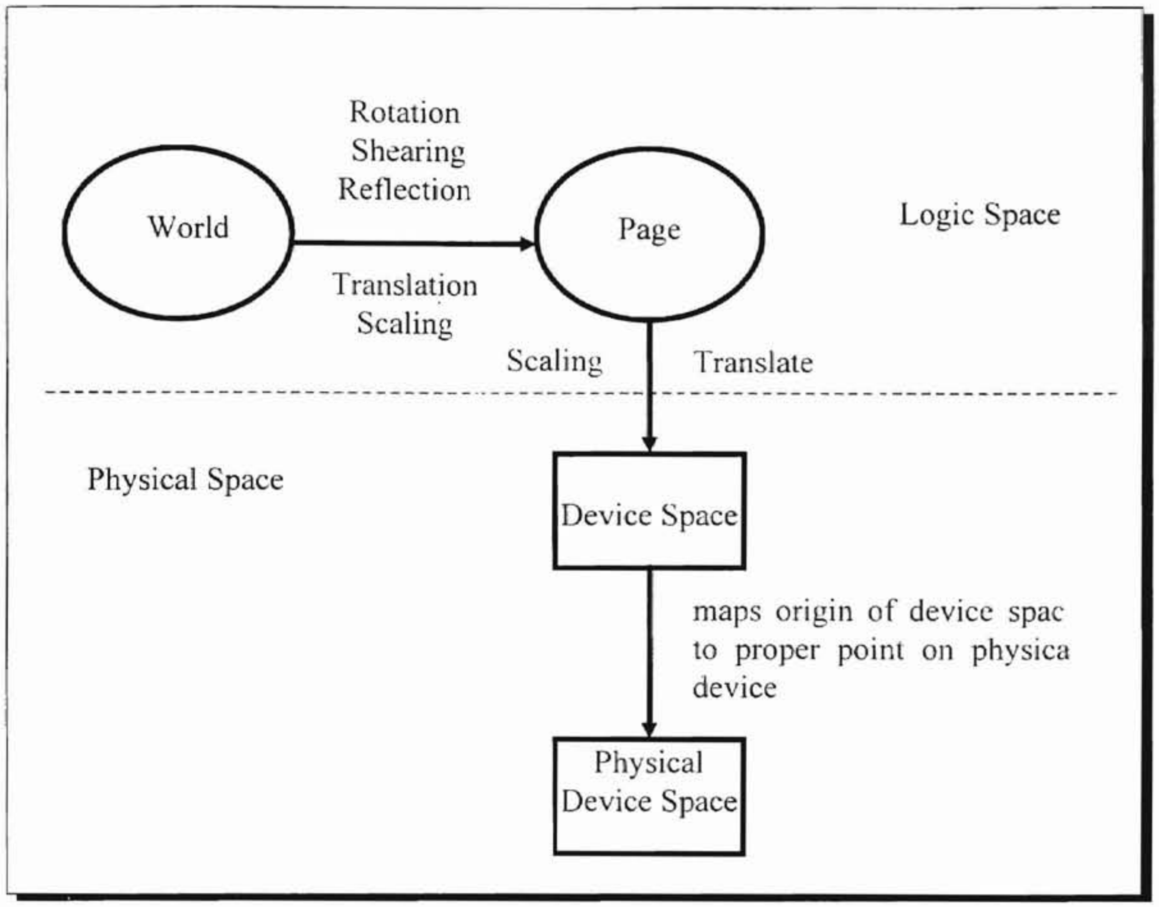

Figure 2. Relations among Four Coordinate Space

The default transformation is page space to device space transformation. It determines the mapping mode which is used by all graphics output associated with particular DC (Device Context, a data structure defining the graphics objects, their associated attributes, and the graphics modes affecting output on a device). A mapping mode is a scaling transformation that translates logic units into device units. It specifies the size of the units used for drawing operations. Mapping mode also determines the orientations of the x-axis and the y-axis. The other attributes of DC, — the window origin, the viewport origin, the window extents, and the viewport extents — are associated with mapping mode. Windows has eight mapping modes (Table 1).[25]

Applications call **SetMapMode (hdc, iMapMod)** to set the mapping mode and call **GetMapMode (hdc)** to obtain current mapping mode.

Table 1. Mapping Modes of Windows (Adopted from [25])

		Increasing value	
Mapping Mode	***Logical Unit***	***x-axis***	***y-axis***
MM_TEXT	Pixel	Right	Down
MM_LOMETRIC	0.1 mm	Right	Up
MM_HIMETRIC	0.01 mm	Right	Up
MM_LOENGLISH	0.01 in.	Right	Up
MM_HIENGLISH	0/001 in.	Right	Up
MM_TWIPS	$^{1}/_{1440}$ in.	Right	Up
MM_ISOTROPIC	Arbitrary (x = y)	Selectable	Selectable
MM_ANISOTROPIC	Arbitrary (x != y)	Selectable	Selectable

Not much work is done by the transformation from device space to physical device space. It is limited to translation and is controlled by window manager component. Its only purpose is to ensure that the origin of device space is mapped to proper position in the physical device.

World space to page space transformation is a new feature of Windows. World space to page space transformation supports rotation, translation, scaling, shear, and reflection. This transformation does not work until application first calls **SetGraphicsMode** to set graphics mode to GM_ADVANCED, and then calls

SetWorldTransform to set this transformation. Unfortunately, Windows 95 does not support GM_ADVANCED (at this time). It means that Windows 95 does not provide rotation, shear and reflection capability. But Windows NT supports GM_ADVANCED. Maybe new versions of Windows 95 will support this feature. **SetWorldTransform** receives a pointer to an **XFORM** structure, which contains appropriate values to deal with transformation. Following is the definition of **XFORM**:

```
typedef struct {
        FLOAT       eM11;
        FLOAT       eM12;
        FLOAT       eM21;
        FLOAT       eM22;
        FLOAT       eDx;
        FLOAT       eDy;
} XFORM;
```

The relation between values of XFORM elements and operations is shown in Table 2. After setting elements of XFORM to appropriate values, applications use

SetWorldTransform (HDC, XFORM);

to set appropriate world to page transformation. after this, object may be drawn in world space.

Table 2. Relation between XFORM and Transformation Operation

(Adopted from [31])

Operation	eM11	eM12	eM21	eM22	eDx	eDy
Scaling	Horizontal scaling component			Vertical scaling component		
Rotation	Cosine of rotation angle	Sine of rotation angle	Negative sine of rotation angle	Cosine rotation angle		
Reflection	Horizontal reflection component			Vertical reflection component		

shear		Horizontal proportionally constant	Vertical proportionally constant			
Translation					Horizontal translation component	Vertical translation component

2.2.2 Interaction Handling in Windows

Windows operating system itself is an interactive system. It provides many means to accept user input. Windows 95 is the first operating system to support plug and play[25]. There are two basic modes used to handle interaction, sampling and event. As a subsystem, GDI does not provide function to handle interaction, but it gets the message passed from the system, and then outputs graphics primitives on physical device. The 32-bit Windows, Windows NT and Windows 95, supports both true multitasking and multithreading. Each 32-bit thread has its own message queue and all 16-bit processes share one message queue. Every Windows application must have at least one window procedure where the application actually process the message passed from the system. The procedure is a call-back type procedure. It means that the procedure is not called by the application itself, but called by Windows system to process the messages associated the application. Most common messages are put into its message queue, and are processed in the first-come-first-out order. Some functions of API can send message that bypass the message queue and directly to window procedure. For example, ***UpdateWindow*** sends a WM_PAINT message directly to window procedure. When the application window is resized or a cover window is removed, system puts a WM_PAINT message in the application message queue. In window procedure, application can take appropriate action to respond to the message. In the new version of GNA95GP, the displaying routine is placed in windows procedure to respond to WM_PAINT message. Whenever application updates its recorded image information, it calls ***UpdateWindow*** to send a WM_PAINT message. The Windows system will put a WM_PAINT message into the application's message queue whenever the application's window is destroyed.

2.2.3 Graphics Primitives

MS Windows' GDI supports *line, curve, filled area, bitmap,* and *text*. The attributes information of these primitives are stored in Device Context (DC). MS Windows uses DC to control GDI functions' behavior.

Line and curves

GDI provides straight lines, rectangles, polygons, ellipse, arcs, pies, chords, and Bézier splines. Polyline is used to simulate other more complicated curves. The display device needs only two functions, **SetPixel** and **GetPixel** (which draw pixel), to output lines and curves. Windows uses pixel to simulate lines and curves[25].

Attributes of lines and curves are color, width, and style. They are determined by Device Context (DC) and logic pen used to draw line and curve. The default logic pen, selected into device context when Windows system creates a device context for application program, is BLACK_PEN. This logic pen draws a one pixel wide black solid line. If an application wants to draw lines using pen other than the default, it should create a logic pen first using **CreatePen** (or **ExtCreatePen**), and then select the pen into

DC using **SelectObject**. After that, Windows uses the pen selected into DC to draw all lines and curves until application de-selects the pen.

Win32 API has two types of pens, cosmetic pen and geometric pen. Geometric pen is new in Win32. Lines' (or curves') width, color, and style are specified by current pen. The current pen is defined by its attribute. Function **CreatePen** creates a cosmetic pen, which is same as the pen in Win16[25]. The parameter list for the function is given in Table 3. Following is an example usage:

HPEN hPen = CreatePen (iPenStyle, iWidth, rgbColor);

Function **ExtCreatePen** is an extension of **CreatePen**, which is new in Win32. It can create cosmetic pen or geometric pen. Table 4 contains the parameter list required for the function.

Table 3. Parameter List for *CreatePen()*

Parameters	Purpose	Options and Comments
iPenStyle	Pen's style	PS_SOLID, PS_DACH, PS_DOT, PS_DASHDOT, PS_DASHDOTDOT, PS_NULL, PS_INSIDEFRAME
iWidth	Width of pen	Cosmetic pen's width is specified in device units
rgbColor	Pen's color	

The pen's type includes PS_GEOMETRIC and PS_COSMETIC. Win32 supports all pen styles of Win16. In addition to the Win16 pen styles, it also supports PS_ALTERNATE and PS_USERSTYLE. Windows 95 does not support PS_ALTERNATE and PS_USERSTYLE styles. There are three end caps (end cap refers to the appearance of the end point of a line), round, square, and flat. "Bevel", "round", and "miter" are attributes used at corners where lines meet. Current pen position, pen, background mode, background color, and drawing mode will affect the appearance of lines.

Table 4. Parameter List for *ExtCreatePen()*

dwPenStyle	Combination of pen type (PS_GEOMETRIC, and PS_COSMETIC), style, end cap, and join attribute using bitwise OR
dwWidth	Width of pen, width of geometric pen is specified in logical units
lplb	Points to **LOGBRUSH** structure
dwStyleCount	length of lpStyle array if PS_USERSTYLE is set, otherwise must be NULL
lpStyle	Points to an array specifying user defined style

Filled Area

Any area enclosed by a series of lines or curves can be filled with current brush object selected into device context[25]. The default brush object selected into device context is WHITE_BRUSH, one of six stock brushes (stock brush — standard colored pattern of pixels used to fill an area[25]). It fills the interior area with white color, which is the same as the default background color. The other five stock brushes are LTGRAY_BRUSH, GRAY_BRUSH, DKGRAY_BRUSH, BLACK_BRUSH, and NULL_BRUSH. When Windows draws filled area, such as rectangle, ellipse, chord, pie, and polygon, it draws the outline of the area using current pen selected into DC, as it draws any lines and curves. Windows fills the area using the brush currently selected into DC. To fill the interior area using the brush other than those six stock brushes described above, application needs to create a logic brush just like creating custom logic pen, and then select it into DC. There are four functions available to create a custom brush,

CreateSolidBrush, **CreateHatchBrush**, **CreatePatternBrush**, and **CreateBrushIndirect**. **CreateSolidBrush** creates a brush filling the area with solid color, specified by its only argument. **CreateHatchBrush** uses its first argument to specify hatch mark style, and uses second argument to specify color of hatch mark. There are six hatch mark styles, HS_HORIZONTAL, HS_VERTICAL, HS_FDIAGONAL, HS_BDIAGONAL, HS_CROSS, and HS_DIAGCROSS. Windows uses background mode and background color to fill the area between the marks. **CreatePatternBrush** creates a brush based on bitmaps. Its only argument is handle of bitmap (*handle* — in Windows, refers to a special variable that refers to an object, such as a handle of a window, a handle of device context). The last function **CreateBrushIndirect** combines the first three functions together. Its only argument points to a **LOGBRUSH** (logic brush) structure, which specifies which brush style will be created, solid, hollow, hatch, or pattern. **SelectObject** is used to select a brush into DC.

Similar to drawing polyline, Windows draws polygon using an array of **POINT** structure. If the last point is not the same as the first one, Windows adds a straight line connecting these two points. The filling of interior area of polygon is affected by the filling mode. The filling modes are ALTERNATE and WINDING. ALTERNATE mode only fills those interior areas which are accessible from outside the polygon by *odd-parity* rule (same as SRGP). WINDING mode, on the other hand, fills all the interior area.

Text

Text attributes supported by MS Windows are font, color, and align. There are two font types in MS Widows, GDI fonts and device fonts. GDI fonts come with GDI package, and are stored in software format. Device fonts are device built-in fonts. GDI fonts include raster fonts, stroke fonts, and TrueType fonts. The default text color is black. **SetTextColor** sets the output text color. The default align is TA_LEFT, in which the position specified is the top left corner of the first character of the string. The other attributes of device context which affect text appearance are background mode and background color. These attributes determine how Windows fills the gap of the text.

Bitmap

Bitmap is a matrix of bits associated with the pixels of a display device. Generally, bitmaps are used to display image which needs to be drawn quickly. Most applications of bitmaps in Windows are animation. The icons, mouse cursors, and buttons are also constructed by bitmaps. There are two bitmap types, device-dependent bitmap and device-independent bitmap (DIB). DIB was introduced in Windows 3.0, which can be stored in disk files. The important difference between bitmaps and DIB is DIB contains color table.

Bitmaps, brushes, fonts, pens, and regions (will be discussed shortly) are objects of DC. If any of these types of objects is created, it must be deleted eventually. But the objects currently selected into device context should not be deleted.

2.2.4 Metafiles

Windows can store bitmaps. But it takes too much storage space to store bitmap, and bitmaps are device dependent. Metafiles are records of collection of GDI commands[25]. Due to this, graphics stored in metafile can be scaled to any size without distortion. Metafiles can exist in memory or disk. Usually, several programs can share a picture using metafiles through clipboard. Windows NT and Windows 95 support enhanced metafile format, which enhanced the old metafile format in the previous Windows. Enhanced metafile format (EMF, or Windows metafile format, WMF) supports extensive header information which helps an application to redisplay the metafile image.

2.2.5 Clipping, Region and Path

Clipping, region and path are three most powerful tools in GDI. Drawing can be restricted in clipping area regardless of the actual shape of the picture. Clipping is generally generated by a region or a path. A region is an area of any shape. Application can create a region using the primitives rectangle, polygon, and ellipse. A region can also be a combination of these regions. A more powerful way to generate clipping is path. Path is new in Win32 API, which is introduced in Windows NT and supported under Windows 95. Path can be generated by a collection of straight lines and curves, including Bézier spline. A path can contain sub-paths, which can be opened or closed. Path can be filled or be converted to region.

2.3 Review of Ada 95 and GNAT Ada 95 Free Compiler

Ada 95 is the extension of Ada which was developed in the late 1970s. It was called Ada 9X and now is called Ada 95. Ada 95 is the only object-oriented language recognized by ISO[24]. It supports package, generics, exceptions, private type, encapsulation, inheritance polymorphism and subprogram overloading. Ada 95 also has well-defined interface to other programming languages, such as C[9].

GNAT (GNU NYU Ada Translator) is a free production-quality Ada 95 compiler, which was originally developed by New York University. It is developed and distributed under GNU General Public License published by Free Software Foundation. GNAT fully supports all features of Ada 95. GNAT is a front-end and runtime system for Ada 95 and uses GCC as its back-end. GCC is the compiler of GNU (GNU is "Not UNIX") system, which is a UNIX-compatible operating system. Originally designed to compile C, GCC now is a multi-language compiler. It is used to compile C++, FORTRAN, Objective-C and Ada, with multiple front-ends and several hardware targets. GCC is fully written in C.[2, 28, 3] There are several GCC types for different platforms, such as DJGPP (for DOS), and EMX (for OS2).

GNAT is ported to several different platforms, such MS DOS, OS2, Windows NT, Windows 95, etc.. The newest released version (by this time) is GNAT 3.09, which is validated on several platforms including Window NT (same as Windows 95). The newest released version for MS DOS is GNAT 3.07 (using DJGPP version 2.0 as its back-end). There are no significant differences between this version and the prior version, GNAT 3.05, which also uses DJGPP 2.0 as its back-end[3]. Since GNAT is based on GCC, it uses different extensions of GCC as back-end for different platforms. For example, GNAT uses DJGPP in its DOS extension (from GNAT 2.06 using DJGPP 1.0 to GNAT 3.07 using DJGPP 2.0), and uses EMX (GCC for OS2) in OS2 version. Figure 3 is the overall structure of GNAT compiler.

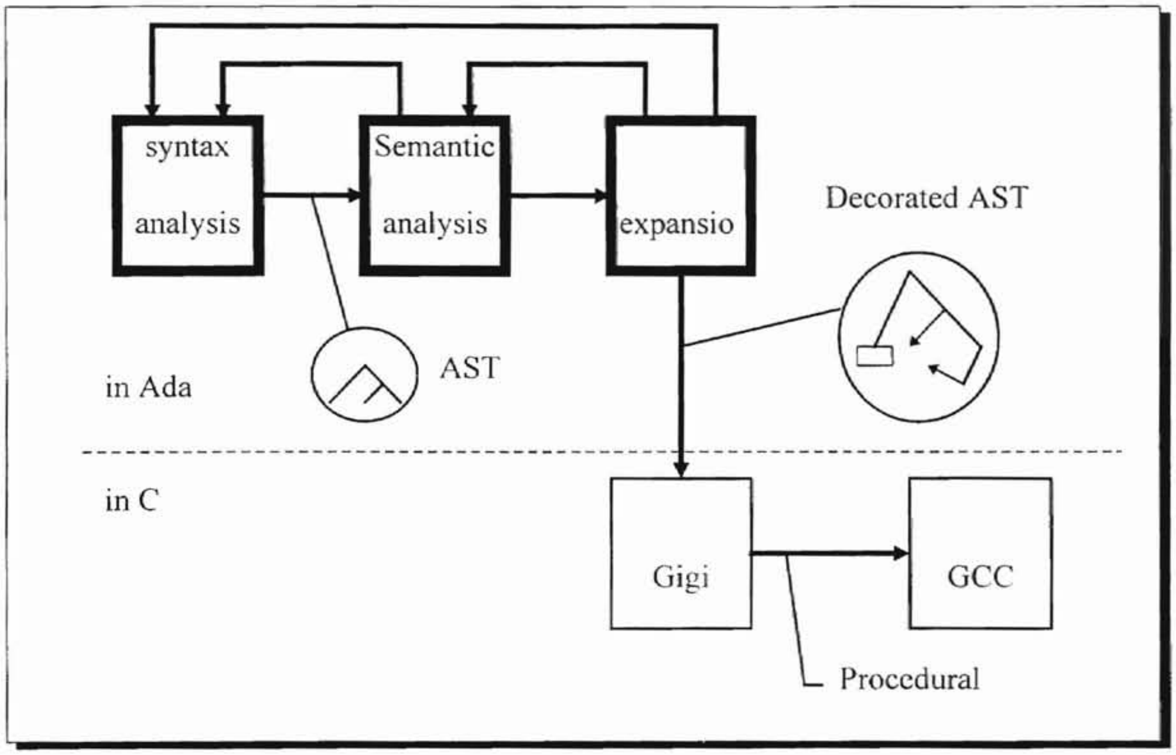

Figure 3. Overall Structure of GNAT Compiler[28]

2.4 Review of RSX Free Windows Develop Kit

RSX Windows Development Kits are written by Rainer Schnitker[27]. There are three packages, RSXWDK, RSXNT, and RSXNTDJ. RSXWDK is an environment for building 32-bit GNU-C applications for Windows 3.1. It needs DJGPP port of GNU C/C++ compiler (only supports version 1.0) to compile the application program. RSXNT and RSXNTDJ are environments for building 32-bit GNU-C application for Windows NT and Windows 95. RSXNT is the original one. It needs EMX port of GNU C/C++ compiler to compile the program. RSXNTDJ, which is a modified version of RSXNT, needs DJGPP port of GNU C/C++ compiler (supports version 2.0) to compile application programs. All these packages are free for non-commercial use. RSXNT and RSXNTDJ implement most of the features of Win32 API.

2.5 Review of Previous GNA95GP Version

GNA95GP is a free Ada 95 2D graphics package. The version for Windows 3.1x is available in public domain. GNA95GP for Windows 3.x (GNA95GP v1 for short) is based on GNAT 2.06. RSXWDK. and DJGPP 1.0. GNA95GP v1 borrows features from SRGP for output and interaction handling, and from PHIGS for storage. The front-end of GNA95GP is implemented in Ada 95. and its back-end is implemented in C. Actually, GNA95GP is a Win32 graphics binding for GNAT. The implementation of GNA95GP v1 is influenced by its environment. GNAT is based on GCC which is implemented in C. Both RSXWDK and DJGPP are implemented in C and use C/C++ as their programming language. As a binding, GNA95GP ought to be a bridge (an interface) connecting Ada 95 applications and the C development environment. GNA95GP is designed to be used in MS Windows environment whose API is also implemented in C/C++. Fortunately, GNAT provides "Pragma import" to import an object or an entity written in C and "Pragma export" to import Ada entity to C.

Although GNA95GP v1 achieved most of the design goals, it has some limitations. GNA95GP v1 only supports GNAT 2.06 which is no longer supported by GNAT support group. A lot of useful features of Windows is not supported by GNA95GP v1. such as clipping region. GNA95GP v1 has storage feature. But it seems not efficient and does not provide use-or-not-use options during programming. In some cases, it becomes pure overhead. GNA95GP v1 does not provide enough methods to query the storage structure.

CHAPTER III

GNA95GP Graphics Package for Windows 95

GNA95GP for MS Windows 95 (GNA95GP v2 for short) is an upgraded and enhanced version based on GNA95GP v1. The major work of this thesis include upgrading GNA95GP v1 from MS Windows 3.1x to MS Windows 95, and enhancing the upgraded version under MS Windows 95. The enhancement work occurs in storage part of the package. It includes redesigning storage part of the package, adding storage option, and adding more powerful and convenient tools (method to query and maintain storage database) to operate storage structure. Before presenting the work performed in this thesis, a brief overview of software dependency in GNA95GP v2 is necessary.

3.1 Software Dependency in GNA95GP v2

Like GNA95GP v1. GNA95GP v2 has been developed based on three software systems, DJGPP, GNAT Ada 95, and RSXWDK. In GNA95GP v2, we use DJGPP 200 (basic part of DJGPP version 2, which comes with GNAT 3.05 for DOS installation), GNAT Ada 95 compiler version 3.05 for DOS (GNA95GP is not tested with GNAT 3.07), and RSXNTDJ. These software systems can be obtained from Internet. Currently, these software systems can be found in the following WWW sites:

- GNAT 3.05 for DOS (include basic part of DJGPP) :

http://gnat.com or http://www.cs.okstate.edu/gna95gp/gnat305.

- RSXNTDJ:

  ftp://ftp.uni-bielefeld.de/pub/systems/msdos/misc.

The documents about these software systems can also be obtained from the above addresses.

3.2 From GNA95GP v1 to GNA95GP v2

As we mentioned in the previous chapter, one of the differences between Windows 3.1x and Windows 95 is that Windows 95 is a 32-bit operating system. That means every thing in Windows 95 is 32-bit. In Windows 95, it is unnecessary to distinguish short integer, integer, and long integer. Almost all 16-bit variables of Windows 3.x are extended to 32-bit now[20]. There is no difference between pointer and far pointer. In Windows 3.x, a far pointer is a pointer reference to the address which can not be represented in 16 bits. In Windows 3.x, holding this kind of value needs a variable declared as far pointer, for example, *char far * mystring*. Because of this change, some Windows 3.x defined date types are dropped by Windows 95, such as, FAR PASCAL, POINT FAR, etc.. Although some data type tags are still used in Windows 95, the data types they represent are changed. For example, UINT is defined as unsigned integer in both of Windows 3.1x and Windows 95. It represents an 32-bit integer in Windows 95 but 16-bit in Windows 3.x. Another example is WPARAM. Every window program for MS Windows must use this data type. Windows programming always deals with message. In Windows, type MSG is the data type for each message variable (holding message information). The type declaration is given below:

```
typedef struct tagMSG {
        HWND        hwnd;
        UINT        message;
        WPARAM      wParam;
        ...
} MSG;
```

The third member of MSG has the data type WPARAM. Every Windows application should have a window procedure (commonly called window proc, usually written as WndProc)[25], where the real action occurs. The usage of window proc is:

```
LRESULT CALLBACK  WndProc (HWND, UINT, WPARAM, LPARAM);
```

The third parameter of WndProc is also a WPARAM. In Windows 3.x, WPARAM was defined as a WORD, which was a 16-bit unsigned short integer. In Windows 95, WPARAM is redefined as a UINT, which is an 32-bit unsigned long integer, while WORD still represents a 16-bit unsigned short integer. Because WPARAM and WORD are same in Windows 3.x, there are a lot of applications which used WORD to replace WPARAM. These include GNA95GP v1. For example, in win_devi.c of GNA95GP v1, it defined a function to handle mouse button event as

```
void HandleWinButtonEvent (WORD wParam, LONG lParam) {...}.
```

In this case, WORD wParam should be replaced with WPARAM wParam. It is safe for both Windows 3.x and Windows 95. All handles in Windows 95 are 32-bit. For those applications which used exact type for handle passed or the return handle from Win32 API functions, it is a problem that need to be fixed. For example, the following code in 16-bit application:

```
{
        WORD        hBKBrush:     /* handle of brush, which determine how
                                     Windows fills area inside an object */
        /* Create a new brush */
        hBKrush = (WORD)CreateSolidBrush(RGB(R,G,B));
        ...
}
```

Changes to

```
{
        HBRUSH      hBKrush;
        /* Create a new brush */
        hBKrush = (HBRUSH) CreateSolidBrush(RGB(R,G,B));
        ...
}
```

in 32-bit application. This is also a common situation raised in Windows 3.x applications.

As a result of changing from 16-bit to 32-bit, some new data types are defined to replace old data types. For example, FAR PASCAL is replaced by WINAPI and CALLBACK. Because of this, functions WinMain and WndProc in Win_Main.c of GNA95GP v1 need to be redefined. The definition:

```
        INT PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                                         LPSTR lpszCmdParam, INT nCmdShow) {...}
        long FAR PASCAL _export WndProc (HWND hwnd, UINT message,
                                                UINT wParam, LONG lParam) {...}
```

in GNA95GP v1 are changed to:

```
        int WINAPI WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                                  LPSTR lpszCmdParam, INT nCmdShow) {...}
        LRESULT CALLBACK WndProc (HWND hwnd, UINT message,
                                          WPARAM wParam, LPARAM lParam) {...}
```

in GNA95GP v2. In Windows 3.x, a 32-bit variable returned from functions can pack two 16-bit values into it, such as coordinate value returned from GDI functions. But in

Windows 95, those functions need larger space than before to return widened values. Thus, a few functions are no longer implemented in Windows 95, and the old functions are extended to include a new additional parameter. This additional parameter of GDI functions makes room to hold or passing widened coordinate values, which is packaged into the simple variable returned from or passed to original function in Windows 3.x,. For example, in GNA95GP, draw pixel is implemented as drawing a short line (two pixel length) (see Figure 4). It moves current position to location (x, y) of the pixel, then draws a line from current position (which is (x, y)), to (x+1, y). In GNA95GP v1, it uses **MoveTo** to move current position to (x, y). In Windows 3.x, **MoveTo** is defined as:

```
DWORD MoveTo (    HDC    hdc,    /* handle of device context */
                  int    x,      /* x-coordinate of new position */
                  int    y       /* y-coordinate of new position */
                  );
```

It returns a 32-bit integer. Low-order of returned variable is logical x-coordinate of previous position, and high-order of it is logical y-coordinate of previous position. In Windows 95, all coordinate variables are 32-bit. This function is extended to **MoveToEx,** which is defined as:

```
BOOL MoveToEx (   HDC        hdc,     /* handle of device context */
                  int        x,       /* x-coordinate of new position */
                  int        y,       /* y-coordinate of new position */
                  LPPOINT    lpPoint  /* address of old position */
                  );
```

The return value is changed to indicate whether the function call succeeds or fails. An additional parameter is added to hold previous position. The new parameter is a pointer to a POINT variable, which is defined as:

```
typedef struct tagPOINT {
        long    x;
        long    y;
} POINT;
```

It has 64 bits, enough to hold two 32-bit value. If an application does not want to keep the old position, it can specify the fourth parameter as NULL.

There is, however, an exception. Windows still packages mouse location message in one 32-bit variable. Low-order of the variable is x-coordinate, and high-order of the variable is y-coordinate.

```
void Call_DrawPixel (struct ELEMENT *ptr) {
        struct param_DrawPixel          *parameter;
        parameter = (struct param_DrawPixel *)(ptr->pParam);
        MoveTo (hdc, parameter->x, parameter->y);
        LineTo (hdc, parameter->x+1, parameter->y);
}
                    Implementation of DrawPixel in GNA95GP v1

void Call_DrawPixel (struct ELEMENT *ptr) {
        struct param_DrawPixel          *parameter;
        parameter = (struct param_DrawPixel *)(ptr->pParam);
        MoveToEx (hdc, parameter->x, parameter->y, NULL);
        LineTo (hdc, parameter->x+1, parameter->y);
}
                    Implementation of DrawPixel in GNA95GP v2

                 CALL_ prefix indicates that this function is called
                   by storage part to display graphics primitive.
```

Figure 4. Implementation of Draw Pixel in GNA95GP

So far, we have discussed only about window programming using one programming language, namely C. There are also some potential problems caused by the change when an application is written in more than one language. A potential problem may raise when passing a variable among Windows C functions and functions written in other languages, such as Ada 95, especially using pass by reference. The problem is caused by the change of variable length in Windows 95.

There is no significant modification in GNA95GP after upgrading the package from Window 3.x to Windows 95. Although GNA95GP breaks the original Windows message loop and sets up its own message loop, it still depends on the messages passed from Windows operating system. GNA95GP deals with Windows' message system, gets the messages from message queue and passes them to user graphics applications. GNA95GP is written in Ada 95 and C. There are many information translations between these two parts. Thus, much of upgrading work is to investigate each line of source code of GNA95GP v1, and removing the problems that are mentioned above.

3.3 Enhancement of GNA95GP under Windows 95

Storage part of GNA95GP v1 is the weakest part of this version. Therefore, most enhancement work is focused on this part. But storage part is not an isolated part in the package. The enhancement work also covers graphics primitive part and event handling part. Actually, all three parts are integrated into one unit after the package is enhanced.

The enhancement work includes redesigning structure of storage, adding z-order concept, adding an option to allow a user to choose to use storage or not, adding more convenient, effective tools to query and maintain storage data. The z-order in this thesis is the virtual z-axis in the direction normal to the screen plane (this plane is x-y plane), from the screen plane to the user. If two or more objects overlap, the object displayed later will cover the object(s) displayed earlier. The z-order is the order of displaying. A more detailed description will be presented later. These enhancements change the entire storage component of GNA95GP v1. Before we discuss enhancement details, a comparison of GNA95GP v1 and GNA95GP v2 is presented in the next section:

3.3.1 New Features of GNA95GP v2

The first new feature in GNA95GP v2 is storage option. With this option, user can decide whether to use retained mode or immediate mode during application development. In GNA95GP v1, there is only a retained mode. All graphics primitives specified in user applications were recorded into storage structure first, and then displayed. In GNA95GP v2, all graphics primitives and their attributes, which are specified between a pair of procedure (or function) calls Gna95gp_OpenNode and Gna95gp_CloseNode are recorded into storage structure as a group of graphics object, called a *node*. It can be either empty or has a set of graphics primitives. At any time, only one node can be opened for editing. A node is similar to the structure of the previous version. The rules about node are the same as rules about structure of the previous version. Primitives recorded in storage structure can be edited later. Any graphics primitive specified outside the pair of procedure (or function) calls is not recorded in storage structure and is displayed directly. Because those primitives are not recorded, there is no way to regenerate them.

The next new feature in GNA95GP v2 is that it will automatically regenerate all primitives recorded in storage structure whenever the image is updated or damaged. The previous version supports this function partially. It can regenerate the image when an application finishes editing a structure. But it can not regenerate the image when the image is damaged by events other than application updating its storage status. GNA95GP v2 can regenerate image not only when current storage status is changed, but also when image is damaged by changing window size, and removing cover window of message box. It is possible to specify primitives' coordinate location outside the original application window. It won't be lost when application window is enlarged enough to display full image.

The third significant new feature is that a node can be selected by a given location. In GNA95GP v1, it is impossible to query the storage structure in such way. GNA95GP v2 provides a method to query the storage structure as to whether a location lies inside a graphics object or not (i.e. supports pick-correlation). In other words, a node can be selected by a cursor location. It is a new way for interaction handling in GNA95GP v2. A selected node is the front most node on which the cursor is located. The cursor can be inside the filled areas of the group or on its boundary. The selection operation returns the node's ID which is selected by cursor.

Unlike the previous version, in which the ways to edit a structure or an element are just inserting a new one and deleting an old one, GNA95GP v2 allows application to have more ways to manage its image. Rather than inserting and deleting, application can set a node or an element of current node (current node is the node opened by Gna95gp_OpenNode currently) visible or invisible, move an element around inside the current node (change the display order), reset the attributes of the current element or current node (the current element is the element of current node pointed by current index), and move current element out of current node (ungroup and re-group). After opening a node, application can change position and shape of the entire node by rotating, translating and scaling. Without opening a node, application can set node's visibility, change display order of nodes (Gna95gp_MoveNode), merge two nodes (grouping), and select a node by cursor location. Table 5 lists functions or procedures associated with storage in GNA95GP v2 and their equivalent functions or procedures in the previous version. Table 6 describes action performed by the functions or procedures associated with storage structure in GNA95GP v2.

Table 5. Functions and Procedures Associated with Storage Structure

Storage functions and procedures in GNA95GP v2	Equivalent functions and procedure in GNA95GP v1
Gna95gp_OpenNode	Gna95gp_openStructure
Gna95gp_RotateNode	N/A
Gna95gp_TranslateNode	N/A
Gna95gp_ScaleNode	N/A
Gna95gp_CloseNode	Gna95gp_closeStructure
Gna95gp_SetNodeVisible	N/A
Gna95gp_MoveNode	N/A
Gna95gp_DeleteNode	Gna95gp_deleteStructure
Gna95gp_selectNode	N/A
Gna95gp_MergeTwoNode	N/A
Gna95gp_SetElementIndex	Gna95gp_setElementIndex
Gna95gp_OffsetElementIndex	Gna95gp_offsetElementIndex
Gna95gp_ShowElement	N/A
Gna95gp_UnshowElement	N/A
Gna95gp_ShowAllElement	N/A
Gna95gp_UnshowAllElement	N/A
Gna95gp_DeleteElement	Gna95gp_deleteElement
Gna95gp_OffsetElement	N/A
Gna95gp_FrontElement	N/A

Gna95gp_BackElement	N/A
Gna95gp_MoeElementOutTo	N/A
Gna95gp_GetElementNum	N/A
Gna95gp_UpdateScreen	N/A

3.3.2 Redesign and Implementation of Storage Structure

GNA95GP is a 2D graphics package. All points are represented by x and y values. But adding a virtual z-axis (z-order) is very useful. Although there is no z-axis actually existing in the 2D display device, the different primitives' display order will generate different effect of the image. It is possible that a primitive generated later overlap the primitive generated earlier. A selection operation only selects the front most graphics object, if two or more objects overlap. To facilitate this convenience, virtual z-axis or z-order is used. The z-order is maintained by a linked list. The back end of z-order is the head of the list, and the front end of z-order is the tail of the linked list. Storage regenerates graphics primitives one by one from the head to tail of the linked list. Z-order exists among nodes, and also among elements within a node. Operations MoveNode, MergeNode, OffsetElement, FrontElement, BackElement, and MoveElementOutTo are designed to adjust z-order (see Table 6) .

Table 6. Storage Functions and Procedures in GNA95GP v2

Function or procedure name	Actions performed
Gna95gp_OpenNode	Open an existing node or create a new node; the initial point to define or redefine a node. The node is set as the current node.
Gna95gp_RotateNode	Rotates a set of primitives in a node after opening the node (a group of primitives) as the current node.

Gna95gp_TranslateNode	Translates a set of primitives grouped in a node after opening it as the current node.
Gna95gp_ScaleNode	Scales a set of primitives group in a node about the origin after opening it as the current node.
Gna95gp_CloseNode	Closes current node.
Gna95gp_SetNodeVisible	Changes visible attribute of node specified by ID.
Gna95gp_MoveNode	Offsets a node in z-order
Gna95gp_DeleteNode	Deletes a node
Gna95gp_selectNode	Selects the front most node containing the point (x, y)
Gna95gp_MergeTwoNode	Merges first node into second node. Elements of the first node are placed at the end of the second node
Gna95gp_SetElementIndex	Sets current index
Gna95gp_OffsetElementIndex	Offsets current index
Gna95gp_ShowElement	Sets element visible
Gna95gp_UnshowElement	Sets element invisible,
Gna95gp_ShowAllElement	Sets all elements visible
Gna95gp_UnshowAllElement	Sets all elements invisible
Gna95gp_DeleteElement	Deletes the current element
Gna95gp_OffsetElement	Moves current element inside the current node
Gna95gp_FrontElement	Brings current element to the front of all the other nodes
Gna95gp_BackElement	Brings current element to the background of all the other nodes
Gna95gp_MoveElementOutTo	Moves current element from the current node, into the node specified by ID
Gna95gp_GetElementNum	Gets the number of elements of the current node
Gna95gp_UpdateScreen	Sends a signal to update the entire application's screen

Sometimes, setting a node or an element visible or invisible also adjusts z-order. If an invisible node (or invisible element) is adjacent to a visible node (or visible element), application can consider them as having the same z-order. Application then can set them visible and invisible alternately at different time in order to obtain different visual effects (animation scheme).

In GNA95GP v1, it is impossible to keep a primitive or a group of primitives in the storage structure and temporally hide them. Visibility setting of nodes (or elements) makes it possible. Application can edit invisible nodes or elements in the same way as the others.

Graphics primitives are primitives with attributes. They either use default attribute settings or use their own attribute settings. GNA95GP v1 uses an attribute inheritance rule.[24] This rule is effective in GNA95GP v2 also. In the previous version, application has to delete and re-define a primitive in order to adjust an element's z-order. This causes a problem — how to maintain elements' attribute setting coherence. An *attribute-setting* is the set of attribute applicable to a drawing primitive. A *complete-attribute-setting* of an element is the attributes-setting set by the previous elements (defined as *environment-attribute-setting*) and attributes-setting set by the element itself. The attribute-setting set by the element has priority over the environment-attribute-setting. That is, for a certain attribute item, if it is set by the previous elements and the current element, the final attribute state is the state set by the current element, because the element changes the attribute setting explicitly. When moving an element, the environment-attribute-setting of the element is changed. According to attribute inheritance rule, the complete-attribute-setting of the element is also changed. That means the definition of the element is also changed. This is not the intend of the moving element operation. If the previous attribute setting scheme is used to maintain attribute coherence, an application needs to create and insert several new elements when an element is moved. The new elements are attribute setting elements. So, in GNA95GP v2, attribute setting is no longer an independent element in the node. Attribute setting is considered as a part of an element. Assume an application wants to move element A. Element B is the element that originally follows element A. After moving, element A is inserted behind element C. To maintain the coherence of attribute-settings of element A, B, and C for an element moving operation. the application only need to modify the attribute-setting in element A. the attribute-setting in element. and the attribute-setting in element C. That does not mean that each element has to include a complete-attribute-setting in its structure. When a node is created. it inherits the system default attributes as its attribute-setting. No attribute-setting information needs to be kept in the element. Once an element changes its attribute-setting, the appearance attributes state remains same until it is changed explicitly or the end of the node is reached. If only the environment-attribute-setting is used, no attribute information need to be kept in the element. When moving an element. application only have to do three things. First is to keep environment-attribute-setting unchanged for the element which immediately followed the moved element before moving. if there is one. Second is to keep the complete-attribute-setting of moved element unchanged. The last thing is. after inserting the moved element into its new position, to keep environment-attribute-setting unchanged for the element which immediately follows the new position of moved element. if there is one. To achieve all these things, application only needs to modify these three elements' attribute-setting. If the complete-attribute-setting of any one of these three elements is the same as its new environment-attribute-setting, no additional attribute information will be kept in that element. Thus, less space and time are used. An element can also be a pure attribute-setting element. that is. an element without a primitive definition.

Once elements can be moved. it is possible to group or ungroup the elements. **Gna95gp_MoveElementOutTo** and **Gna95gp_MergeTwoNode** provide these functions.

In GNA95GP v2. an application draws all recorded primitives and unrecorded primitives in the same window. A unrecorded primitive's z-order is undefined. After application updates a recorded primitive, the storage structure will automatically regenerate all recorded primitives. It is possible that the regeneration will damage the previously drawn unrecorded primitives. To minimize this damage, the actual regeneration is limited in the areas where the updated node occupied before and after updating.

To implement the redesign, a new storage structure is defined. This storage structure keeps the basic characteristics of the previous version's storage structure. The architecture of the storage structure is a two level doubly linked list. The main list is the node list. Each element in this list is a NODE structure. A NODE structure contains the head of the second level linked list — a list of primitives.

New node structure has three additional members (see figure 5).

```
typedef struct NODE *pNODE;
struct NODE
{
        int             VisibleFlag;    /* new, 0, invisible, 1, visible */
        int             NodeID;
        int             ElementNum;
        pELEMENT        pElement;       /* pointer to first element in the node */
        pELEMENT        pLastElement;   /* new, pointer to last element in the node */
        HRGN            NodeRgn;        /* new, region occupied by node, */
        pNODE           prev;
        pNODE           next;
};
```

Figure 5. Definition of node structure

Element VisibleFlag is added to indicate whether to display this node or not. When regenerating the image, display routine checks this flag first. If the node is invisible, it is bypassed. A handle of region NodeRgn is added as a member of the node structure. This member holds the identification of an area (region) of client window which contains the primitives of the node. Every time Gna95gp_CloseNode is called, this member is updated to hold new client window area occupied by updated node. The procedure Gna95gp_CloseNode invalidates an area. This area includes the areas (regions) which are occupied by the node before and after updating. This area is the area where storage regenerating routine actually draws the primitives. The third new member points to tail of element list.

```
typedef struct ELEMENT *pELEMENT;
struct ELEMENT
{
        int             AttrFlag;       /* new, 1 indicates attributes are changed,
                                        0 indicates no change occurs,
                                        pAttribute is set to NULL */
        int             VisibleFlag;    /* new, 0 invisible, 1 visible */
        int             PrimitiveID;
        pATTRIBUTE      pAttribute;     /* new, pointer to attribute setting, if changed */
        char            *pParam;
        pELEMENT        prev;
        pELEMENT        next;
};
```

Figure 6. Definition of element structure

Figure 6 gives the definition of element structure. Three new members are added in this structure. The member VisibleFalg indicates whether the element is visible or not. AttrFlag indicates whether attribute-setting is changed in the element. If attribute-setting is changed in the element, this flag is set to 1, and the third new member pAttribute is set to point to an attribute-setting structure, which is allocated dynamically to hold attribute-setting information. When display routine is called to generate the image, it checks the AttrFlag. If the flag is set to CHANGED, display routine reads the information stored in the attribute-setting structure, and calls appropriate attribute-setting routines to set the attribute. If the VisibleFlag is set to INVISIBLE, display routine bypasses the rest of the element. Actual primitive definition information is stored in an appropriate parameter structure pointed by member pParam. When rotating, translating, and scaling operations are applied to the current node, each primitive definition information, which is stored in its parameter structure pointed by pParam, is recalculated by an appropriate operation routine. New definition information is restored back to its parameter structure. Application does not need to track the information by itself. Operations of rotating, translating, and scaling are designed to be used only after a node is opened. This design makes it possible for an application to apply a sequence of rotating, translating, and scaling operations to the node before the node is redisplayed.

A new global flag StorageFlag is added. This flag and old global flag OpenFlag work together to implement storage option. The StorageFlag indicates whether the storage structure is updated or not. In GNA95GP v2, an application can update a node without opening that node, such as Gna95gp_SetNdevisible and Gna95gp_SelectNode. The OpenFlag only indicates whether a node is opened or not. When these flags are set to true, it indicates storage is being used and a node is opened to edit. When a primitive is specified, an appropriate function (one of the functions defined in output.ads) is called. This function checks these two flags. If these two are set to true, the function creates a new element, fills the element structure with proper information, and inserts the new element into the current node. If these two flags are set to be "False", the primitive specifying function directly outputs the primitive onto the display screen.

When changing window size. removing a cover or other events generated by something other than application itself, Windows invalidates the damaged area. Whenever an area of client window is invalidated by application itself or Windows system, a WM_PAINT message is sent to window procedure, and then window application can respond to the message with appropriate actions. GNA95GP rewrites the message loop, and responds to the messages inside of WinMain function, unlike common window applications responding to messages in window procedure (WndProc). GNA95GP v1 version calls displaying routine to display recorded primitives inside the procedure **Gna95gp_closeStructure** which closes editing a node. When an application closes a node, the displaying routine is directly called by the application. When using GNA95GP, the main procedure written by users is "ada_main". This is the entry of user application. The procedure "ada_main" is exported to C and called by WinMain function (the main function of window program). When an event occurs, Windows system converts the event into a message. and puts it into message queue. WinMain function gets message through GetMessage function call. WinMain then passes back the message to Windows system. Windows then sends it to the window procedure of the application. Window procedure is the place where a window application actually responds to the message. Window procedure is not called by WinMain directly[25]. WM_PAINT message is not handled by event handling routines in the previous version. All of these make it impossible to cover the damaged screen destroyed by the events such as changing window size, or removing cover window. In GNA95GP v2. the display routine is placed in window procedure whenever regeneration is needed, the damaged area is set as an invalid region. and Windows call display routine to update the invalid region.

When **Gna95gp_SelectNode** function is called, it searches the entire recorded primitives from the front node to the last node one by one according to their z-order (traverse the linked-list from tail to head). Once a node is selected, **Gna95gp_SelectNode** stops searching and returns the selected node's ID. To implement this function, the member NodeRgn of a node structure is used. An area of the application's window identified by NodeRgn contains the object represented by the node. For a filled graphics primitive (polygon, filled rectangle, and filled arc), the primitive's region is the same as the area occupied by it. For an unfilled primitive (lines and curves), such as polyline, rectangle, and elliptic arc, the primitive's region is the area bounded by the primitive including the boundary. If the primitive is not closed, Windows assumes that there is a straight line from the ending point to the starting point of the curve drawn by the primitives. The node's region is the union of all its elements' regions.

First, **Gna95gp_SelectNode** checks whether the given location is within the region of a visible node identified by NodeRgn or not. To implement this step, the Win32 API function **PtInRegion** is called. Given a region identification and a location (x, y), **PtInRegion** returns a Boolean value which indicates whether the location is inside the region. If yes, further checking continues as outlined in the next paragraph. If not, it bypasses the node, and checks the next node. This scheme shortens the search time. Usually, most nodes are excluded from further checking.

If the location is inside the node's region, **Gna95gp_SelectNode** continues to check whether the location is really on the primitives of the node or not. If **Gna95gp_SelectNode** determines that the location is on one of the elements of the node, it returns the node's ID and terminates. If not, **Gna95gp_SelectNode** goes back to the previous step. In this step, there are two cases, filled primitive, and unfilled primitive. For filled primitive, **Gna95gp_SelectNode** checks whether the location is in the region of the primitive. Under this case, two Win32 API functions are called. **CreatePolygonRgn** or **CreateEllipticRgn** is called to calculate the region of the element. **PtInRegion** is called to check whether the location is on the element. For unfilled primitive, vectors are being used. A curve is treated as a sequence of line segments. Each line segment can be considered as a straight line. The following method is used to determine if a location is on a straight line:

Let ***a*** be a vector from one end point of the line segment to the location. Let ***b*** be a vector in the direction of the line segment (see figure 7). Let ***p*** be the projection of ***a*** on ***b***, and ***e*** be normal to ***p*** such that $\boldsymbol{a} = \boldsymbol{p} + \boldsymbol{e}$. Cursor location is presented by (x, y). Line segment is represented by (x0, y0) and (x1, y1). If the length of ***e*** is less than half the width of the line, direction of ***p*** is the same as the direction of ***b***. Then if the length of ***p*** is less than the length of ***b***, location (x, y) is on the line segment (Figure 7(a)). That means the line segment is selected by the cursor. In figures 7(b) and 7(c), even though the length of ***e*** is less than half of the line width, location (x, y) is not on the line segment. In figure 7(b), the length of ***p*** is longer than the length of ***b***. In figure 7(c), the direction of vector ***p*** and vector ***b*** are opposite.

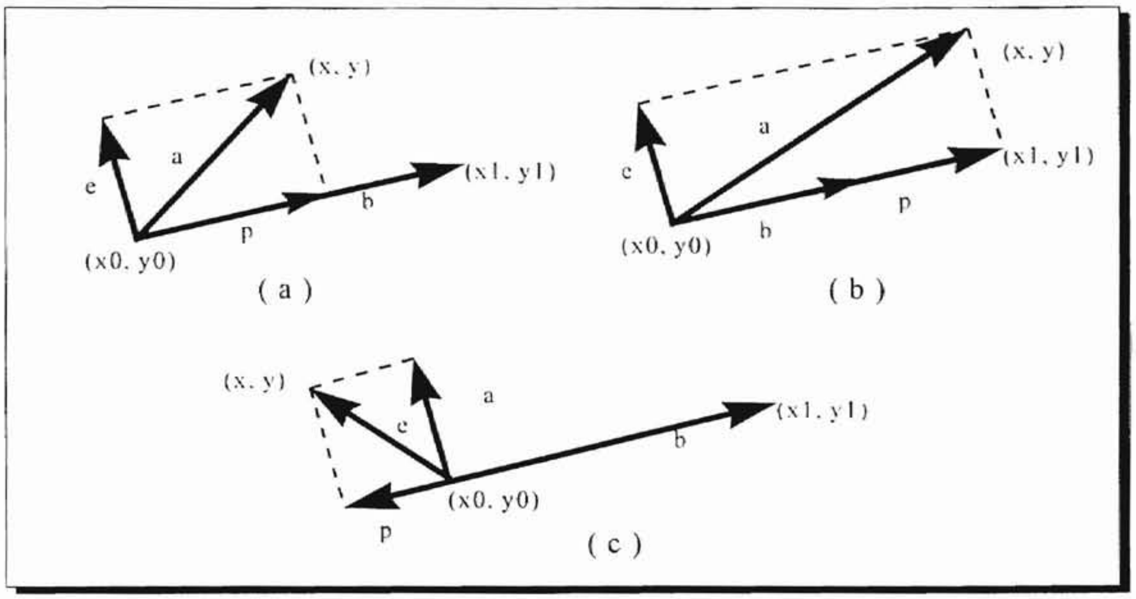

Figure 7. Using vectors for hit test (adapted from [6])

CHAPTER IV

Summary

As a free Ada 95 2D graphics package, GNA95GP is updated to be used under Windows 95 and enhanced. Z-order concept is introduced into this package. Several new features are added. More convenient tools to manage storage are provided.

GNA95GP now supports storage as an option. With this feature, an application can decide whether to record a graphics primitive at the time it specifies the primitive. Several new functions and procedures are added to support inter-node and intra-node movement of primitives. Applications can use these functions to adjust primitives' display order inside a node or among the nodes. In the new version, there are two functions to support grouping and ungrouping graphics primitives. They are Gna95gp_MoveElementOutTo, and Gna95gp_MergeTwoNode. The procedures, Gna95gp_RotateElement, Gna95gp_TranslateElement, and Gna95gp_ScaleElement provide convenient methods to edit a group of primitives. They can also be combined to implement a complicated operation, and free the application to track actual change of primitives' definition information. Gna95gp_SelectNode, which selects a node by given location, gives a new way to query the storage structure. Setting the visibility of an element or a node gives more efficient ways to change the appearance of an image. Windows' region concept is used in the storage structure to minimize unrecorded primitives' damage caused by updating recorded primitives. A convenient tool, Gna95gp_UpdateScreen, gives application a new way to clear the entire screen and regenerate all recorded primitives. Gna95gp_GetElementNum allows applications to get the number or element in the current node.

Storage structure only records the information specified by functions and procedures listed in the attribute package (attribut.ads, attribut.adb) and output package (output.ads, output.adb). A guide for installation of the package is provided in appendix A. An users' guide is provided in appendix B. A demo program is given in appendix C.

Bibliography

1. ***Ada 9X Reference Manual***, Intermetrics. Inc., Cambridge, Mass, 1994.
2. Ada Information Clearinghouse. ***AdaIC's Available Ada Bindings Report***. WWW site: http://sw-eng.falls-church.va.us/AdaIC/tools/bindings/bindings95/html/toc.html.
3. Ada Information Clearinghouse. ***Accessing the Free GNAT Compiler for Ada 95***, Web site http://sw-eng.falls-church.va.us/AdaIC/docs/flyers/gnat.htm
4. ***Ada Quality and Style: Guidelines for Professional Programmers***, Software Productivity Consortium, Inc., Herndon, Virginia, December, 1992.
5. Booch, G.. ***Software Engineering with Ada***. The Benjamin/Cummings Publishing Company, Inc.. Menlo Park, CA, 1987.
6. Crain, Dennis. ***Win32: Hit Testing Lines and Curves***, Microsoft Developer Network Technology Group, Microsoft Developer Network. April 1996, Microsoft Inc., Redmond. Wash.
7. David. F. R.. ***Computer Graphics Techniques: Theory and Practice***, Springer-Verlag New York Inc., 1990.
8. Enderle, G., Kansy, K., Pfaff, G., ***Computer Graphics Programming: GKS – The Graphics Standard***. Springer-Verlag Berlin Heidelberg New York. 1987
9. Feldman, Michael B., Koffman, Elliot B., ***Ada 95 Problem solving and Program Design***, Addison-Wesley Publishing Company. 1996.
10. Foley, J. D.. van Dam, Andries. Hughes. John F.. ***Computer Graphics: Principles and Practice***, Second Edition. Addison-Wesley Publishing Company. 1987.

11. Foley, J. D., van Dam, Andries, Feiner, Steven K., Hughes, John F., Phillips, Richard L., ***Introduction to Computer Graphics***, Addison-Wesley Publishing Company. 1994.

12. Gart, M. ***Interfacing Ada to C — Solutions to Four Problems***, TRI-Ada '95 Conference Proceedings, 28-34, November, 1995.

13. George, K. M., Li, Lan; Kuang, Shang, Huang, Yang, ***Gna95gp: Free Ada 95 2D Graphics Package***, WWW site: http://www.cs.okstate.edu/gna95gp

14. Gery, Ron, ***GDI Overview***, Microsoft Developer Networ, Technology Group, Microsoft Developer Network, April 1996, Microsoft Inc., Redmond, Wash.

15. ***Information Processing Systems — Computer Graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS), Language Bindings, FORTRAN***, American National Standards Institute (ANSI)/International Organization for Standards (ISO) 9593.1 1990.

16. ***Information Processing Systems — Computer Graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS), Language Bindings, Ada***, ANSI/ISO 9593.3:1990.

17. ***Information Processing Systems — Computer Graphics — Programmer's Hierarchical Interactive Graphics System (PHIGS), Language Bindings, C***, ANSI/ISO 9593.4:1991.

18. James, F., Leisy, J., ***Computer Graphics: the principles behind the art and science***, Franklin, Beedle & Associates, Irvine, CA. 1989.

19. James, F., Andries, D., ***Computer Graphics: Principle and Practice***, Addison-Wesley Publishing Company, Inc., 1990.

20. Kath, Randy, ***Porting 16-Bit Windows-Based Applications to Win32***, Microsoft Developer Network Technology Group, Microsoft Developer Network, April 1996, Microsoft Inc., Redmond, Wash.

21. Kosko, L., ***PHIGS Reference Manual***, O'Reilly & Associates, Inc., Sebastopol, CA, 1989.

22. Kuang, Shan, ***Event Handling in GNA95GP Graphics Package***, M. S. thesis, Department of Computer Science, Oklahoma State University, May 1997.

23. Li, Lan, Kuang, Shang. and George, K. M., ***A 2D Graphics Package in Ada 95***, 10th Annual Ada Software Engineer, Education & Training (ASEET) Symposium, Prescott, Arizona. June, 1996.

24. Li, Lan, ***Design and Implementation of A 2D Graphics Package in Ada 95***, M. S. thesis, Department of Computer Science, Oklahoma State University, Dec. 1996.

25. Petzold, Charles; Yao, Paul, ***Programming Windows 95***, Microsoft Press. 1996.

26. ***Programmer's Hierarchical Interactive Graphics System (PHIGS)***, Federal Information Processing Standards Publication (FIPS PUB) 153-1.

27. Schnitker, Rainer ***RSXNTDJ*** FTP: ftp://ftp.uni-bielefeld.de/pub/systems/msdos/misc.

28. Schonberg, Edmond, Banner, Bernard, ***The GNAT Project: A GNU-Ada 9X Compiler***, Web site: http://www.gnat.com

29. ***Tech. Corrigendum, Programmer's Hierarchical Interactive Graphics System (PHIGS), Language Bindings, FORTRAN***, ISO/IEC (International Electrotechnical Commission) 9593.1: 1990.

30. ***Tech. Corrigendum, Programmer's Hierarchical Interactive Graphics System (PHIGS), Language Bindings, Ada***, ISO/IEC 9593.3:1990.

31. ***Win32 Programming's Reference***. Microsoft Developer Network Library, April 1996, Microsoft Inc., Redmond, Wash..

32. Young, Michael J., ***Introduction To Graphics Programming For Windows 95: Vector Graphics Using C++***, AP Professional, Academic Press, Inc. 1996.

Appendix A. GNA95GP v2 Installation and Usage

A-1. Introduction

GNA95GP v2 is an upgraded version of GNA95GP v1. It is a 2D Ada 95 graphics package for MS Windows 95. GNA95GP v2 and its previous version GNA95GP v1 are all free software according to the definition of the Free Software Foundation. The primary motivation for the development of GNA95GP is support for undergraduate graphics education. GNA95GP is distributed with the expectation that it will be useful, but WITH OUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

GNA95GP v2 is developed using three software systems, GNAT 3.05 for DOS, DJGPP version 2.0, and RSXNTDJ. GNAT is a free Ada 95 compiler, which was originally developed by New York University. It is now maintained by Ada Core Technologies Inc.. DJGPP is a free environment for developing 32-bits protected mode software in C/C++ under MS-DOS, which was developed by DJ Delorie. RSXNTDJ is a free environment for creating Win32 applications for Windows 95 and Windows NT, which was developed by Rainer Schnitker. In order to use GNA95GP v2 correctly, these three software have to be installed first. It is the user's responsibility to use these software. Please read copyrights and licenses of these software before installing them.

A-2. Where to find GNA95GP v2 and other software

GNA95GP v2 is provided at WWW site: GNA95GP: Free Ada 95 2D Graphics Package For PC (URL: http://www.cs.okstate.edu/gna95gp).

GNAT 3.05 for DOS is provided by Ada Core Technologies Inc.. Presently, this software system through their WWW site: http://www.gnat.com. The basic part of GNAT 3.05 for DOS is also available at WWW site: http://www.cs.okstate.edu/gna95gp/gnat305.

GNA95GP only needs basic part of DJGPP version 2.0, called DJGPP 200. The installation of GNAT 3.05 for DOS includes the DJGPP 200. Presently, the DJGPP version 2.0 is available at WWW site: http://www.delorie.com.

Currently, RSXNTDJ is available at FTP site:

ftp://ftp.uni-bielefeld.de/pub/systems/msdos/misc.

A-3. Installation of GNA95GP v2

Before installing GNA95GP v2, a user needs to install and setup GNAT 3.05 for DOS (includes basic part of DJGPP) and RSXNTDJ correctly. These two software systems come with the instructions for installation and setup. The recommend steps to setup GNA95GP v2 are presented below:

- GNAT 3.05 for DOS installation and setup

The author recommends installation GNAT 3.0 for DOS before installing RSXNTDJ. After downloading GNAT 3.05 from Internet, one will find that GNAT 3.05 comes with two installation executables (ginstall.exe and dinstall.exe), and installation instructions (readme.txt). Read the instructions first. Then choose one executable to install GNAT 3.05. The ginstall.exe installs GNAT 3.05 from hard disk, and the dinstall.exe installs it from floppy disk. Then follow the instructions of installation executable to install GNAT 3.05. Installation program will put a GNAT 3.05 's document named "gantinfo.txt" under the sub-directory "Gnat305s" of GNAT 3.05 home directory. Read that document, and follow the instructions to test GNAT 3.05.

- RSXNTDJ insulation

Download the file "rsxntdj1.zip" from its FTP site. After decompressing the file, read the file "rsxntdj.hlp", which is under the "doc" sub-directory of RSXNTDJ home directory. Follow the instructions to setup RSXNTDJ. Make sure that the path of RSXNTDJ library precedes the library of GNAT 3.05, for example:

```
SET LIBRARY_PATH=D:\RSXNTDJ\LIB;D:\GNAT305\LIB;\GNAT305\LIB\ADALIB
```

If you run the command 'gcc -v' you should get the message:

```
Reading specs from d:\rsxntdj\lib\specs
gcc version 2.7.2
```

If you install RSXNTDJ in a directory other than "D:\RSXNTDJ", read the instructions in "rsxntdj.hlp" to modify the "SPECS" file in "\RSXNTDJ\LIB" directory.

- GNA95GP v2 insulation

1. Download GNA95GP v2 from "GNA95GP: Free Ada 95 2D Graphics Package" web site.
2. Decompress the "gna95gp2.zip" file.
3. Read the "readme.txt" file.

4. Run the executable "setup.exe", and follow the instructions. Executable "setup.exe" will setup GNA95GP correctly on you machine and create a batch file "gna95gp.bat" under GNA95GP home directory.

• Uninstall GNA95GP

Delete entire GNA95GP directory.

A-4. Usage of GNA95GP v2

To build an executable of an Ada program using GNA95GP, the initial environment memory must be least 2048. This can be set by the user using Windows 95 tools. Then, open a DOS box under Windows. Run batch file "gna95gp.bat", which is created by GNA95GP installation, to set environment variables. Now, a user will be able to build an executable from Ada source program. GNA95GP provides three convenient tools to build executable from Ada programs. They are ca.bat, gplink.exe, and gpmake.exe. Assume there is an Ada program named "**myprog.adb**". The following commands issued from the DOS prompt will build an executable file.

1. ***ca myprog.adb***

This command compiles the "myprog.adb", and generates an object file "**myprog.o**".

2. ***gplink myprog.o***

This command links the object file "myprog.o" with GNA95GP library, and generates an executable file, whose default name is the first input file's name (excluding extension) with the extension ".exe". In this example, the default name of output executable is "myprog.exe". The "-o" option of ***gplink.exe*** is used to specify user specified name. For example, the executable name of "myprog.o" will change to "myexec.exe" if the command used is "***gplink -o myexec myprog.o***".

3. ***gpmake.exe***

gpmake.exe is a tool which combines functions of ***ca.bat*** and ***gplink.exe*** into one. It has two options, "-c" and "-o". They can not be used together. The option "-c" means compiling the Ada program only. The option "-o" is the same as the option "-o" of ***gplink.exe***. Table 7 is the usage of GNA95GP's compilation tools and their equivalent commands.

Table 7. GNA95GP's Compiler Tools and Equivalent Commands

Usage of GNA95GP Compiler Tools	Equivalent Command
ca myprog.adb	*gcc -c -O myprog.adb*[1]
gplink myprog.o	*gcc -Zwin32 -o myprog.tmp myprog.o*[2]
gplink -o myexec myprog.o	*gcc -Zwin32 -o myexec.tmp myprog.o*[2]
gpmake -c myprog.adb	*ca myprog.adb*
gpmake myprog.o	*gplink myprog.o*
gpmake -o myexec myprog.o	*gplink -o myexec myprog.o*
gpmake myprog.adb	*ca myprog.adb*
	gplink myprog.o
gpmake -o myexec myprog.adb	*ca myprog.adb*
	gplink -o myexec myprog.o

1. See "gnatinfo.txt" of GNAT 3.05 for detail
2. See "rsxntdj.hlp" of RSXNTDJ for detail.

The executable can be run from DOS prompt. After the file "rsxnt.dll" is copied from "bin" sub-directory of RSXNTDJ into "Windows\system" directory, the executable can be run directly from Windows.

Appendix B. User's Guide for GNA95GP v2 Storage

This user's guide is targeted for storage section of GNA95GP v2. The complete document is included in the GNA95GP v2 installation.

B-1. Introduction

Storage component of GNA95GP is a set of functions and procedures implemented in Ada 95 and C that support graphics applications to store graphics primitives, maintain recorded information, and display recorded primitives. Retained mode or immediate mode can be chosen during the programming. Compared to the previous version, storage section of GNA95GP v2 provides several more powerful, efficient, and convenient tools. With this storage section, GNA95GP now supports several new features. Applications can adjust displaying order of nodes or that of elements in a node, temporarily hide nodes or elements from displaying, groups and ungroup primitives, edit a node by rotating, translating, and scaling. All of the features provided by the storage section of the previous version are also supported in this storage section.

B-2. Limitation of GNA95GP v2 Storage Componet

Not all information specified by GNA95GP functions and procedures can be recorded in storage structure. Only the primitives and attributes specified by the functions and procedures defined in the attribute and output packages can be recorded in storage structure when these functions or procedures appear between the pair of function calls, Gna95gp_OpenNode and Gna95gp_CloseNode. These functions and procedures are listed below:

Functions and procedures defined in attribute package:

- gna95gp_setLineStyle(lineStyle : gna95gp_lineStyle);
- gna95gp_setLineWidth(lineWidth: integer);
- gna95gp_setColor(color: integer);
- gna95gp_setBackgroundColor(color: integer);
- gna95gp_setFillStyle(fillStyle: gna95gp_drawStyle);
- gna95gp_setFillBitmapPattern(patternIndex: integer);
- gna95gp_setWriteMode(mode: gna95gp_writeMode);

Functions and procedures defined in output package:

- Gna95gp_DrawPixel(x: in XCoord; y: in YCoord);
- Gna95gp_DrawMarker(x:XCoord; y: YCoord; MarkerSize: integer;
  MarkerStyle: Gna95gp_MarkerStyle);
- Gna95gp_line (p1: in tagPoint; p2: in tagPoint);
- Gna95gp_RectanglePoint(leftbottomPoint: in tagPoint; rightupPoint: in tagPoint);
- Gna95gp_Rectangle(rect: in tagRectangle);
- Gna95gp_polyLineCoord(vertexCount: in PosInt: X_Array: in VErtexXCoordList;
  Y_Array: in VErtexYCoordList);
- Gna95gp_PolyLine(vertexCount: in PosInt; vertics: in VErtexList);
- Gna95gp_Polygon(vertexCount: in PosInt; vertics: in VErtexList);

- Gna95gp_FilledRectanglePoint(leftbottomPoint: in tagPoint;

  rightupPoint: in tagPoint);

- Gna95gp_FilledRectangle(rect: in tagRectangle);

- Gna95gp_ellipseArc(Px: integer; Py: integer; Rl: integer; Rs: integer;

  Ang: float; StartAng: float; EndAng: float);

- Gna95gp_FilledEllipse(Px: integer; Py: integer; Rl: integer;

  Rs: integer; Ang: float; StartAng: float; EndAng: float);

- Gna95gp_text(point: in tagPoint; text: String).

B-3 Usage of Storage Structure

To record primitives and attributes in storage structure, application needs to call Gna95gp_OpenNode to define starting point explicitly. Any primitive or attribute specified by function or procedure listed in the previous section after this point and before the end point will be recorded in storage structure, and will be displayed by storage displaying routines. Gna95gp_CloseNode defines the end point, and closes the recording. At any time, only one node can be open for editing. Any primitive or attribute specified outside the definition block will be displayed immediately on the screen, and no information about that is recorded.

In GNA95GP v2, when storage section record the primitives, attribute is no longer an independent element in a node. It becomes a part of primitive element in the storage structure. The attribute-settings specified immediately after a primitive definition is considered as a part of the primitive. Like the previous version, the appearance attributes state remains the same until it is changed explicitly or end of the node is reached. The first element of a node can be a pure attribute-setting element. That means an element only contains attribute setting. The visibility of the element is set as invisible and the pointer to primitive definition is set as NULL. Attribute settings of unrecorded primitives in this version are the same as in the previous version.

The usage of storage functions and procedures are presented below.

Function Gna95gp_OpenNode(ID: integer) return integer:
Open a existing node or create a new node; The beginning point to define or redefine a node. The node is set as current node.
Parameters: Node ID, if found in storage structure, an existing node is opened, otherwise, a new node is created.
Return: integer, indicate whether the function succeeded or not. 1, if succeeds; otherwise 0.

procedure Gna95gp_RotateNode(flag: integer; angle: float):
Rotate a set of primitives group in current node after open this node as current node.
Parameters: flag, if 1, rotate about x axis; if 2, rotate about y axis, otherwise, rotate about z axis; angle, rotate angle in degree.
Return: none.

procedure Gna95gp_TranslateNode(dx, dy: integer):
Translate a set of primitives group in current node after opening this node as current node.
Parameter: dx, offset along x axis: dy, offset along y axis.
Return: none.

procedure Gna95gp_ScaleNode(sx, sy: float):
Scale a set primitives group in current node about origin point after opening this node as current node.
Parameter: sx, scale ratio along with x axis; sy, scale ratio along with
Return: none.

procedure Gna95gp_CloseNode:
Close current node.
Parameters: none.
Return: none.

procedure ***Gna95gp_SetNodeVisible(ID, flag: integer)***:
Change visible attribute of node specified by ID.
Parameters: ID, node ID; flag, if 1, set the node visible; if 0, set it invisible.
Return: none.

procedure ***Gna95gp_MoveNode(ID, offset: integer)***:
Offset the node specified by ID;
Parameters: ID, node ID; if offset > 0, move forward, if offset < 0, move backward.
Return: none.

procedure ***Gna95gp_DeleteNode(ID: integer)***:
Delete the node specified by ID.
Parameters: node ID;
Return: none.

function ***Gna95gp_SelectNode(x, y: integer) return integer***;
Select the front most node by location (x, y).
Parameters: location of cursor.
Return: selected node ID; 0, if no node is selected.

function ***Gna95gp_MergeTwoNode(nodeA, nodeB: integer) return integer***:
Merge nodeA into nodeB. Elements of node A are placed at end of node B's elements list.
Parameters: ID of nodeA and ID of nodeB.
Return: 1, if succeeds; otherwise, 0.

procedure ***Gna95gp_SetElementIndex(ind: integer)***:
Set current element index to ind.
Parameters: index application wants to set
Return: none.

procedure ***Gna95gp_OffsetElementIndex(offset: integer)***:
Offset current element index.
Parameters: offset current index; if offset < 0, move backward; if offset > 0, move forward.
Return: none.

procedure ***Gna95gp_ShowElement(index: integer)***:
Set the element specified by index visible.
Parameters: element index.
Return: none.

procedure ***Gna95gp_UnshowElement(index: integer)***:
Set the element specified by index invisible.

Parameters: element index.
Return: none.

procedure Gna95gp_ShowAllElement:
Set all elements of current node visible.
Parameters: none.
Return: none.

procedure Gna95gp_UnshowAllElement:
Set all elements of current node invisible.
Parameters: none.
Return: none.

function Gna95gp_DeleteElement return integer:
Delete current element.
Parameters: none.
Return: 1, if succeeds; otherwise 0.

procedure Gna95gp_OffsetElement(offset: integer):
Move current element around inside the current node.
Parameters: offset, distance from current position. if offset > 0, move forward; if offset < 0, move backward.
Return: none.

procedure Gna95gp_FrontElement:
Move current element to the front of node.
Parameters: none.
Return: none.

procedure Gna95gp_BackElement:
Move current element to the background of node.
Parameters: none.
Return: none.

function Gna95gp_MoveElementOutTo(ID: integer) return integer:
Move current element from the current node, into the node specified by ID.
Parameters: node ID, if found in node list, the element is moved to the end of the node; otherwise, a new node is created and the element is moved into the new node
Return: 1, if succeeds, otherwise, 0.

function Gna95gp_GetElementNum return integer:
Get number of the elements in current node;
Parameter: none;
Return: integer, number of elements in current node.

procedure *Gna95gp_UpdateScreen*:
Send signal to update entire screen
Parameters: none.
Return: none.

Appendix C Demo Program Using GNA95GP v2

The program presented below is a demo program using GNA95GP v2. It displays Celsius and Fahrenheit measures as thermographs. Mouse click inside a thermographs, will display appropriate degree in both thermographs and the appropriate values in boxes above the thermographs. Mouse click "EXIT" box will exit the program. Mouse click the locations outside those areas, application will display warning message.

```
with io;
use io;
with init;
use init;
with devices;
use devices;
with WinType;
use WinType;
with head;
use head;
with Output;
use Output;
with objtype;
use objtype;
with attrtype;
use attrtype;
with attribute;
use attribute;
with structure;
use structure;


procedure ada_main is

        pragma suppress(All_Checks);

        C_Meter: tagRectangle;
        F_Meter: tagRectangle;
        C_txtBox: tagRectangle;
        F_txtBox: tagRectangle;
        Exit_Box: tagRectangle;
        C_Text: String(1..3);
        F_Text: String(1..3);
        C_Value: integer;
        c0: integer;
        f0: integer;
        F_Value: integer;
        x: integer;
        y: integer;
        text_point: tagPoint;
        ctext :tagPoint;
        ftext: tagPoint;
        warn_text: String := "Sorry! Try again";
        timeout: integer;
        measure: deluxe_locator_measure;
        device: inputDevice;
        dtext: tagPoint;
        cy1: integer;
        cy2: integer;
        fy1: integer;
        fy2: integer;
        ID: integer;
        flag: integer;

begin

        Gnat95gp_init;

        ----------------------------------------------------------------------
        -- create window and set device, mode
        ----------------------------------------------------------------------

        Gna95gp_CreatWindow("ADA WINDOWS--Celsius Fahrenheit Translator", 639, 479);
        Gna95gp_setInputMode(LOCATOR,EVENT);
        Gna95gp_setInputMode(KEYBOARD,INACTIVE);


        ----------------------------------------------------------------------
        -- make set rectangles' position
        ----------------------------------------------------------------------
        -- rectangle for Celsius thermometer
        C_Meter := makeRect_Coord(200, 340, 250, 140);
        cy1 := 340;
        c0 := 10;
        -- for Fahrenheit thermometer
        F_Meter := makeRect_Coord(400, 340, 450, 140);
        fy1 := 340;
        f0 := 50;
        -- for Celsius value display
        C_txtBox := makeRect_Coord(200, 80, 250, 50);

        -- for Fahrenheit value display
```

```
      F_txtBox := makeRect_Coord(400, 80, 450, 50);

      -- make rectangle box for "exit"
      Exit_Box := makeRect_Coord(100, 360, 160, 330);

      -- make text point
      text_point := makePoint(20, 100);
      ctext := makePoint(200, 360);
      ftext := makePoint(400, 360);

      if Gna95gp_OpenNode(9) = 1 then
-------------------------------------------------------------------
--draw rectangle
-------------------------------------------------------------------
              attribute.gna95gp_setColor(4);
              Gna95gp_FilledRectangle(C_Meter);
              Gna95gp_CloseNode;
      end if;

      if Gna95gp_OpenNode(10) = 1 then
              Gna95gp_FilledRectangle(F_Meter);
              attribute.gna95gp_setColor(4);
              Gna95gp_CloseNode;
      end if;
      -------------------------------------------------------------------
--write text
-------------------------------------------------------------------
      if Gna95gp_OpenNode(3)=1  then
              Gna95gp_text(ctext, "Celsius");
              Gna95gp_text(ftext, "Fahrenheit");
              Gna95gp_CloseNode;
      end if;
      -------------------------------------------------------------------
      --draw lines for Celsius thermometer degree
      -------------------------------------------------------------------
      if Gna95gp_OpenNode(4) = 1 then
              Gna95gp_lineCoord(250, 340, 260, 340);
              Gna95gp_lineCoord(250, 320, 255, 320);
              Gna95gp_lineCoord(250, 300, 260, 300);
              Gna95gp_lineCoord(250, 280, 255, 280);
              Gna95gp_lineCoord(250, 260, 260, 260);
              Gna95gp_lineCoord(250, 240, 255, 240);
              Gna95gp_lineCoord(250, 220, 260, 220);
              Gna95gp_lineCoord(250, 200, 255, 200);
              Gna95gp_lineCoord(250, 180, 260, 180);
              Gna95gp_lineCoord(250, 160, 255, 160);
              Gna95gp_lineCoord(250, 140, 260, 140);

      -------------------------------------------------------------
```

```
    -- display degree
    ------------------------------------------------------------
            dtext :=makePoint(265, 330);
            Gna95gp_text(dtext, "-20");
            dtext :=makePoint(265, 290);
            Gna95gp_text(dtext, "0");
            dtext :=makePoint(265, 250);
            Gna95gp_text(dtext, "20");
            dtext :=makePoint(265, 210);
            Gna95gp_text(dtext, "40");
            dtext :=makePoint(265, 170);
            Gna95gp_text(dtext, "60");
            dtext :=makePoint(265, 130);
            Gna95gp_text(dtext, "80");

------------------------------------------------------------
--degree for Fahrenheit
------------------------------------------------------------
            Gna95gp_lineCoord(450, 340, 455, 340);
            Gna95gp_lineCoord(450, 330, 455, 330);
            Gna95gp_lineCoord(450, 320, 460, 320);
            Gna95gp_lineCoord(450, 310, 455, 310);
            Gna95gp_lineCoord(450, 300, 455, 300);
            Gna95gp_lineCoord(450, 290, 455, 290);
            Gna95gp_lineCoord(450, 280, 460, 280);
            Gna95gp_lineCoord(450, 270, 455, 270);
            Gna95gp_lineCoord(450, 260, 455, 260);
            Gna95gp_lineCoord(450, 250, 455, 250);
            Gna95gp_lineCoord(450, 240, 460, 240);
            Gna95gp_lineCoord(450, 230, 455, 230);
            Gna95gp_lineCoord(450, 220, 455, 220);
            Gna95gp_lineCoord(450, 210, 455, 210);
            Gna95gp_lineCoord(450, 200, 460, 200);
            Gna95gp_lineCoord(450, 190, 455, 190);
            Gna95gp_lineCoord(450, 180, 455, 180);
            Gna95gp_lineCoord(450, 170, 455, 170);
            Gna95gp_lineCoord(450, 160, 460, 160);
            Gna95gp_lineCoord(450, 150, 455, 150);
            Gna95gp_lineCoord(450, 140, 455, 140);

------------------------------------------------------------
--display degree value
------------------------------------------------------------
            dtext :=makePoint(465, 310);
            Gna95gp_text(dtext, "0");
            dtext :=makePoint(465, 270);
            Gna95gp_text(dtext, "40");
            dtext :=makePoint(465, 230);
            Gna95gp_text(dtext, "80");
```

```
        dtext :=makePoint(465, 190);
        Gna95gp_text(dtext, "120");
        dtext :=makePoint(465, 150);
        Gna95gp_text(dtext, "160");
        Gna95gp_CloseNode;
end if;

if Gna95gp_OpenNode(5) = 1 then
        Gna95gp_FilledRectangle(Exit_Box);
        gna95gp_setColor(3);
        dtext := makePoint(115, 335);
        Gna95gp_text(dtext, "EXIT");
        gna95gp_setColor(4);
        gna95gp_setBackgroundColor(3);
        Gna95gp_CloseNode;
end if;

if Gna95gp_OpenNode(6) = 1 then
        Gna95gp_lineCoord(200, 340, 250, 340);
        Gna95gp_setColor(6);
        Gna95gp_CloseNode;
end if;

if Gna95gp_OpenNode(7) = 1 then
        Gna95gp_lineCoord(400, 340, 450, 340);
        Gna95gp_setColor(6);
        Gna95gp_CloseNode;
end if;

if Gna95gp_OpenNode(8) = 1 then
        Gna95gp_text(text_point, "Sorry, Try again!");
        attribute.gna95gp_setColor(2);
        Gna95gp_CloseNode;
        Gna95gp_SetNodeVisible(8, 0);
end if;


timeout := -1;
flag := 0;

loop
        --get events
        device := Gna95gp_waitEvent(timeout);

        if (device = LOCATOR) then

                --get mouse position
                measure := Gna95gp_getDeluxeLocator;
                if measure.button_chord(1) = DOWN then
```

```
                    x := measure.position.x;
                    y := measure.position.y;

------------------------------------------------------------
-- if click mouse inside Celsius or Fahrenheit box
-- display degree and value in graphics box and text box
------------------------------------------------------------
                    ID := Gna95gp_SelectNode(x, y);
                end if;

                if ID = 9 OR ID = 6 then
                    flag := 0;
                    Gna95gp_SetNodeVisible(8, 0);
                    cy2 := y;

                    if Gna95gp_OpenNode(6) = 1 then
                        Gna95gp_TranslateNode(0, cy2-cy1);
                        Gna95gp_CloseNode;
                        cy1 := cy2;
                    end if;

                    Gna95gp_FilledRectangle(C_txtBox);
                    Gna95gp_FilledRectangle(F_txtBox);
                    C_Value := (340-y)/2-20;
                    if (C_Value > 38) and (c0 < 38) then
                        c0 := C_Value;
                        if Gna95gp_OpenNode(9) = 1 then
                            Gna95gp_SetElementIndex(1);
                            Gna95gp_setColor(2);
                            Gna95gp_CloseNode;
                        end if;
                    end if;
                    if (C_Value < 38) and (c0 > 38) then
                        c0 := C_Value;
                        if Gna95gp_OpenNode(9) = 1 then
                            Gna95gp_SetElementIndex(1);
                            Gna95gp_setColor(4);
                            Gna95gp_CloseNode;
                        end if;
                    end if;
                    Gna95gp_num_to_string(C_Text, C_Value);
                    dtext := makePoint(201, 55);
                    Gna95gp_setColor(4);
                    Gna95gp_text(dtext, C_Text);

                    F_Value := 9 * C_Value / 5 + 32;
                    if (F_Value > 100) and (f0 < 100) then
                        f0 := F_Value;
                        if Gna95gp_OpenNode(10) = 1 then
```

```
                                Gna95gp_setColor(2);
                                Gna95gp_CloseNode;
                        end if;
                end if;
                if (F_Value < 100) and (f0 > 100) then
                        f0 := F_Value;
                        if Gna95gp_OpenNode(10) = 1 then
                                Gna95gp_setColor(4);
                                Gna95gp_CloseNode;
                        end if;
                end if;
                Gna95gp_num_to_string(F_Text, F_Value);
                dtext := makePoint(401, 55);
                Gna95gp_setColor(4);
                Gna95gp_text(dtext, F_Text);
                fy2 :=340- (F_Value + 20);

                if Gna95gp_OpenNode(7) = 1 then
                        Gna95gp_TranslateNode(0, fy2-fy1);
                        Gna95gp_CloseNode;
                        fy1 := fy2;
                end if;


        elsif ID = 10 OR ID = 7 then
                flag := 0;
                Gna95gp_SetNodeVisible(8, 0);
                fy2 := y;

                if Gna95gp_OpenNode(7) = 1 then
                        Gna95gp_TranslateNode(0, fy2-fy1);
                        Gna95gp_CloseNode;
                        fy1 := fy2;
                end if;

                Gna95gp_FilledRectangle(C_txtBox);
                Gna95gp_FilledRectangle(F_txtBox);
                F_Value := 340-y-20;
                if (F_Value > 100) and (f0 < 100) then
                        f0 := F_Value;
                        if Gna95gp_OpenNode(10) = 1 then
                                Gna95gp_setColor(2);
                                Gna95gp_CloseNode;
                        end if;
                end if;
                if (F_Value < 100) and (f0 > 100) then
                        f0 := F_Value;
                        if Gna95gp_OpenNode(10) = 1 then
                                Gna95gp_setColor(4);
                        Gna95gp_CloseNode;
                    end if;
                end if;
                Gna95gp_num_to_string(F_Text, F_Value);
                dtext := makePoint(401, 55);
                Gna95gp_setColor(4);
                Gna95gp_text(dtext, F_Text);

                C_Value := (F_Value-32)*5/9;
                if (C_Value > 38) and (c0 < 38) then
                    c0 := C_Value;
                    if Gna95gp_OpenNode(9) = 1 then
                        Gna95gp_SetElementIndex(1);
                        Gna95gp_setColor(2);
                        Gna95gp_CloseNode;
                    end if;
                end if;
                if (C_Value < 38) and (c0 > 38) then
                    c0 := C_Value;
                    if Gna95gp_OpenNode(9) = 1 then
                        Gna95gp_SetElementIndex(1);
                        Gna95gp_setColor(4);
                        Gna95gp_CloseNode;
                    end if;
                end if;
                Gna95gp_num_to_string(C_Text, C_Value);
                dtext := makePoint(201, 55);
                Gna95gp_setColor(4);
                Gna95gp_text(dtext, C_Text);
                cy2 :=340-(C_Value + 20) * 2;

                if Gna95gp_OpenNode(6) = 1 then
                    Gna95gp_TranslateNode(0, cy2-cy1);
                    Gna95gp_CloseNode;
                    cy1 := cy2;
                end if;

            elsif ID = 5 then
                exit;
            else
                if flag = 0 then
                    Gna95gp_SetNodeVisible(8, 1);
                    flag := 1;
                end if;
            end if;

        end if;

    end loop;
```

```
end ada_main;

pragma export (C, ada_main, "ada_main");
```

VITA

Yang Huang

Candidate for the Degree of

Master of Science

Thesis: GNA95GP V2: AN ADA 95 GRAPHICS PACKAGE FOR WINDOWS 95

Major Field: Computer Science

Biographical:

Education:

Received Bachelor of Science degree in Polymer Chemistry from Fudan University, Shanghai, China in July 1989. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in July 1997.