Chapter 11
Graphics Toolkit

Overview

These words were designed to give a simple starting point for interfacing to the Macintosh QuickDraw toolbox. The words are fairly generic. An application written using them can (and has been) ported to or from other machines with different graphics environments. The majority of these words will work on both the Macintosh and Amiga versions of HMSL.

If need be, you can also access any of the advanced features of the Macintosh by using the TRAP: and :STRUCT facilities. So while these GR words do not access all of the Macintosh facilities, you are not restricted from doing so. By looking at the source code, you will see how to extend this toolkit.

Graphics Tutorial

The simplest way to do graphics in HMSL is to draw in the HMSL window. To open the HMSL window, enter:

```
HMSL.OPEN
```

This will open the window used by the HMSL screens.

To draw in this window we can set the current color, and then issue move and draw commands. These commands are based on using an imaginary color pen. To pick up the pen and reposition it you use GR.MOVE . To put the pen down and drag it in a straight line you use GR.DRAW .

```
2 GR.COLOR!
30 30 GR.MOVE
200 55 GR.DRAW   ( draw a line from  30,30 to 200,55 )
300 100 GR.DRAW  ( draw a line from 200,55 to 300,100 )
```

Now let's try drawing a filled rectangle in another color.

```
1 GR.COLOR!
20 20 100 120 GR.RECT   ( draw filled rectangle )
```

We can also output a text message, try entering:

```
3 GR.COLOR!
250 60 " Hello!" GR.XYTEXT
```

The Macintosh uses numbers to reference different colors. The Macintosh video circuitry displays a picture by reading these numbers from memory and converting them to an actual color. It uses a *color table,* or *pallette*, to figure out what color should be displayed for a given number.

A Simple Graphics Program

Let's experiment with defining a Forth word that draws random vectors. (*Vector* is the graphics industry word for a single line segment.) We can use the Forth word CHOOSE which selects a random value between 0 and whatever value is on the stack. You may want to enter these in a file so that you can modify the program when done.

```
ANEW TASK-RANDOM_LINES

WIndowTemplate MYWT

: DRAW.RAND  ( -- , draw a random vector )
   8 choose gr.color!
```

```
      400 choose    ( generate random x )
      200 choose    ( generate random y )
      gr.draw       ( draw line )
   ;

   : ?CLOSEBOX ( -- flag , eat events until the closebox hit )
      ev.get.event EV_CLOSE_WINDOW =
   ;

   : MANY.RAND ( -- , draw many random vectors )
      hmsl.open
      BEGIN
         draw.rand
         ?closebox   ( has closebox been hit )
      UNTIL
      hmsl.close
   ;
```

?CLOSEBOX will return a TRUE if the closebox is ever hit. This gives you a way out of the program.

Opening Other Windows

To demonstrate how this system works, let's open a window and do some drawing.

First we need to declare a WindowTemplate structure. This is a template that contains information about the window we desire including its width, height, location, etc. This structure will occupy some memory space in the HMSL dictionary. We refer to structures by the address of their first byte. Every byte in the computer has an address which is simply a number which uniquely identifies it. When we say the we are "passing a structure" or "passing a pointer to a structure" we mean that we are passing the address of the structure. (Pointer means address.)

HForth provides a word called WINDOW.DEFAULTS that will set the values in a WindowTemplate structure to some reasonable values.

```
WindowTemplate MYWT       ( define a structure )
MYWT WINDOW.DEFAULTS ( set default values in structure)
```

We can use FILE? to see how the WindowTemplate structure is defined. Enter:

```
FILE?  WindowTemplate
```

and hit 'Y' when asked if you want to see the source code. You will see that it is defined as follows:

```
:STRUCT WindowTemplate  \ Structure used to describe a new window
   long  wt_wStorage
   struct   rect  wt_rect
   long  wt_title
   short wt_visible
   short wt_procID
   long  wt_behind
   short wt_goAwayFlag
   long  wt_refcon
;STRUCT
```

Let's examine and change some of the default settings. The WIndowTemplate contains a rectangle that contains the top, bottom, left, and right values for the window. Enter:

```
MYWT .  ( print address of first byte just for fun )
MYWT .. WT_RECT S@ RECT_TOP .  ( set by NEWWINDOW.SETUP )
34 MYWT .. WT_RECT S! RECT_TOP  ( let's change it to 500 )
MYWT .. WT_RECT S@ RECT_TOP .  ( did it work? )
```

The 34 in the above example was the top position of the window in pixels or video dots. We can now pass that structure to the Macintosh Toolbox asking for a window.

```
MYWT GR.OPENWINDOW  .S  ( open the window )
```

Notice that a window has been opened. (It also set the window tracker to HMSL-TRACKER which will take care of the standard input events, dragging, etc. More on this in the chapter on the Macintosh Toolbox.)You may also have noticed that a value was left on the stack That was the window pointer. You should always check to make sure that this pointer is non-zero. If it is zero, it means that the window did **not** open and your program will probably crash if you pretend that it did.

If we want to draw in this window we must make it the current window by entering:

```
.S  ( is the window pointer it still on the stack? )
GR.SET.CURWINDOW
```

GR.SET.CURWINDOW will set the current window to the window specified.

When you call GR.SET.CURWINDOW, the window pointer will be stored in the variable GR-CURWINDOW. To bring this window to the front, we can call:

```
GR-CURWINDOW @ SelectWindow()
```

SelectWindow() is a Mac Toolbox that can be intermixed with HMSL graphics calls.

To finish this session we should close the window and terminate the graphics system.

```
GR.CLOSECURW
```

Generic Graphics Glossary

There are three main types of routines in this toolkit. The Control Routines initialize and terminate data structures, and are involved with opening and closing windows. The Output Primitives produce actual graphics output in the current GrafPort. The Output Attribute involve the appearance, colors, modes, etc. of the Output Primitives.

Control Routines

`GR.INIT ( -- , Initialize the Graphics Subsystem )`

Set attributes to their default values. This is called internally by HMSL.INIT

`GR.TERM ( -- , Terminate the Graphics Subsystem )`

This is called internally bu HMSL.TERM

`GR.CLOSECURW ( -- , Close CURRENT window if open)`

This command looks in the variable GR-CURWINDOW for a window pointer and closes it if one is there. It then clears GR-CURWINDOW and GR-CURRPORT.

`GR.CLOSEWINDOW ( Window -- , Close an Intuition Window )`

If this window is the same as the window stored in the variable GR-CURWINDOW then both GR-CURRPORT and GR-CURWINDOW will be cleared. This is to prevent a window from being closed twice, a fatal error, when working with multiple open windows.

`GR-CURWINDOW  ( -- addr , Variable with rel addr of Window)`

`GR.SET.CURWINDOW  ( Window -- , Sets the current Window )`

This sets the current GrafPort to this windows GrafPort. Subsequent drawing operations, therefore, will take place in this window. If you are using multiple windows, you should save your own window pointers to each of them. You can call this word to determine which of your windows will be drawn into by the GR.xxx routines.

The variables GR-CURRPORT and GR-CURWINDOW are set by this routine.

Output Primitives

GR.CLEAR (-- , Clear the current drawing surface.)

The rectangle will be based on the current windows drawing surface.

GR.DEHIGHLIGHT (x1 y1 x2 y2 -- , DEHighlight a Rect Region)

Reverse the effect of GR.HIGHLIGHT

GR.DRAW (xpix ypix -- , Draw a line.)

This will draw a line from the current position to xpix, ypix using the current attributes.

GR.HIGHLIGHT (x1 y1 x2 y2 -- , Highlight a Rectangular Region)

This will highlight a region to bring attention to it. On the Macintosh this will be done by XORing with color = 3.

GR.MOVE (xpix ypix -- , Move the current position to xpix, ypix)

GR.NUMBER (n -- , Display number at current position.)

If you need special formatting, you can format the number separately and draw using GR.TYPE .

GR.RECT (x1 y1 x2 y2 -- , Draw a rectangle)

This will draw a rectangle using the current color. The current position will be left at x1, y1.

Warning! Make sure that X2 >= X1 and that Y2 >= Y1. Otherwise the Macintosh library routine will overwrite a huge part of chip memory and you will crash. See JD:DEMO_BOXES.

GR.TEXT ($string -- , Draw a text string)

Draw text at the current position. The string must have a byte count at the given address. The current position will be left at the end of the text.

```
" Hello" GR.TEXT
```

GR.TEXTLEN (addr count -- xpixels , x size of string)

Returns length of string in pixels if drawn in current font. This can be used to right justify text by moving XPIXELS to the left of your right margin and drawing from there.

```
100 \ right margin
" Hello" COUNT GR.TEXTLEN -  \ calc x position
50 ( -- x y ) GR.MOVE  \ to start
" Hello" GR.TEXT  \ will end at x=100
```

GR.TYPE (addr count -- , Draw text, like Forth TYPE)

GR.XYTEXT (xpix ypix string -- , Draw a text string)

Draw text at the given position. This is essentially GR.TEXT that does a move first.

Output Attributes

The appearance of these output primitives can be controlled by the setting of output attributes. These attributes remain in effect until changed. The setting words are balanced by query words so that an environment can be saved, changed, and then restored by low level code. The setting words end in ! and the query words end in @ .

GR.COLOR! (color-index -- , Set the color for drawing.)

A color-index is a number that references a color defined in the current pallette.

```
3 GR.COLOR!
```

GR.MODE! (mode -- , Set the drawing mode.)

This controls the logic mode that is used to modify the pixels when drawing. We have also defined two constants GR_INSERT_MODE and GR_XOR_MODE to promote portability.

GR.COLOR@ (-- color-index , Fetch the color for drawing.)

GR.MODE@ (-- mode , Fetch the drawing mode.)

Macintosh only Graphics Routines

These are only supported on the Macintosh or work differently on the Macintosh.

GR.FONT! (fontnum -- , set current font)

Warning the Amiga accepts a Font **structure** instead of a number. Font control is not portable between machines.

4 GR.FONT! (sets monaco font)

GR.FONT@ (-- font , get font from current GrafPort)

GR.SETPORT (-- , sets GrafPort)

Draw into the window specified in GR-CURWINDOW.

WindowTemplate (<name> --INPUT-- , define a window template)

WINDOW.DEFAULTS (WindowTemplate -- , set defaults)

Graphics Input

In event driven systems, all input events should be routed through a top level routine that can handle any event that is generated. These include .i.Mouse input, 11-;mouse movement, button presses, menu picks, window close box hits, etc. The type of events generated and the way that they are handled is different in every application. For this reason, we have not included a routine that only handles mouse location input.

Event Driven Programming.

A new style of programming is evolving to meet the needs of highly interactive systems. In modern user interfaces, the user is normally free to use any input that the program offers. These might include picking from menus, moving windows around, entering graphic information, hitting the keyboard, or poking at other gadgets on the screen. The program must be ready to respond to any of these input events. A typical program is structured with a loop at the top of the program that gets input events and processes them with a case statement. The loop is exited when, for example, a CLOSEBOX is hit, or QUIT is selected from a menu. Since the user's input events control the flow of the program, this style is referred to as 'Event Driven Programming'.

Routines in HH:H4TH_EVENTS - EV.xxxx

This part of the system normally has to be developed from scratch to meet the needs of individual programs. We have included some simple routines, however, to get people started.

?CLOSEBOX (-- flag)

Checks for a CLOSEBOX hit in GR-CURWINDOW. This is handy if the only kinds of events you want are CLOSEBOX events. All other classes of events are lost. Used in the demos.

EV.2CLICK? (-- flag)

Returns TRUE if last mouse click was the second click of a double click. See JD:DEMO_CLICK

EV.FLUSH (-- , flush all events from queue)

This does EV.GET.EVENT until a EV_NULL event is received.

EV.GET.EVENT (-- class , get message class or NULL)

This is a portable word that gets the last relevant event. The event classes returned are:

```
EV_NULL
EV_MOUSE_DOWN  ( call EV.GETXY to get x,y)
EV_MOUSE_MOVE  ( requires EV.TRACK.ON )
```

```
    EV_MOUSE_UP
    EV_MENU_PICK
    EV_CLOSE_WINDOW
    EV_REFRESH
    EV_KEY      ( call EV.GET.KEY to get char )
```

`EV.GET.KEY  ( -- char , last key pressed )`

`EV.GETXY ( -- x y , get last reported mouse position )`

This just fetches the values in EV-LAST-MOUSEX and EV-LAST-MOUSEY .

`EV.TRACK.ON  ( -- , turn on mouse movement tracking )`

`EV.TRACK.OFF   ( -- , turn off mouse movement tracking )`

Advanced Graphics and Event Topics

I do not expect people to get much more into graphics than is explained in this chapter. For more information on graphics and events, I urge you to see the following files:

```
    HH:H4TH_GRAPHICS
    HH:H4TH_EVENTS
    HSYS:Mac_Calls
    HSYS:Mac_Misc
    HSYS:Mac_Structures
```

You will find many examples of how to call the mac toolbox.