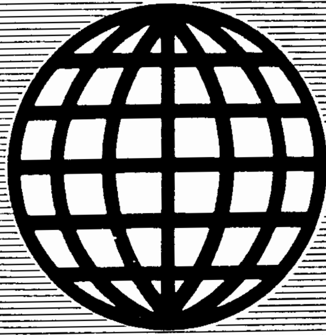


The COMPUTER JOURNAL®

Programming - User Support
Applications

Issue #27 $3.00

68000 TinyGiant

Low Cost 16-bit SBC and Operating System

The Art of Source Code Generation

Disassembling Z-80 Software

Feedback Control System Analysis

Using Root Locus Analysis and Feedback Loop Compensation

The Hitachi HD64180

New Life for 8-bit Systems

A Tutor Program for Forth

Writing a Forth Tutor in Forth

Disk Parameters

Modifying The CP/M Disk Parameter Block for Foreign Disk Formats

ISSN # 0748-9331

The C Column
A Graphics Primitive Package
by Donald Howes

It's now December 18, and I'm frantically typing this column so I can leave on the 20th to go to Canada for Christmas. By the time you read this, the holiday season will have passed. I hope that it has been a happy one for all. Hopefully, I won't still be chained to this desk.

In this column, I'll start to present code for a graphics primitive package for the IBM PC. While these particular primitives are for that machine, it shouldn't be hard to adapt the code to run on other machines, either MS-DOS based or other operating systems. Once the primitives are complete (which will be this and the next column) we can take a stab at developing some applications using them.

By hiding the machine dependent code at the lowest level, it will free us to develop the important things in a more general and machine independent fashion, aiding in the porting of our software from one system to another. This time, I'll present code for the most basic parts of the primitive package. These routines initialize and terminate the application, provide move and draw routines, and the ability to activate a pixel and clip a line (see Listings 1 and 2).

Getting Things Started (and Stopped)

To start things off, it is necessary to initialize the graphics application. The function graf_init() does this, setting the current x and y location, the maximum and minimum x and y coordinate values, getting the display mode parameters operative when the graphics application started and then setting the system into high resolution graphics mode.

I must admit, I've cheated a bit on this function, since I've hard-wired the initialization routine for an Enhanced Graphics Adapter (EGA). For those not using an IBM or compatible system, this card provides higher resolution and more colors in both the foreground and background than the Color Graphics Adapter (a palette of 16/64). It would be possible to write an initialization routine that could check whether the system had a CGA or EGA video card, but there are differences between the cards that go beyond resolution that would complicate some of the other routines (I also don't have a CGA card to test the code I would write).

The int86() function accesses the DOS interrupts and is used here to access interrupt 10H (ROM BIOS video interrupt). Users of non-IBM systems will have to find the appropriate means to do the analogous on their machines. The first call finds out what the video mode and video page are when the application starts, so that the same environment can be restored on exit. The second call sets the video card into the EGA high resolution 640×350 mode.

Once the application is complete, the function graf_term() restores the original video mode and video page using the information discovered in graf_init(). Resetting the video mode will automatically set the page to 0, so, if a video page other than the first was being used, it has to be set independently.

Where Do We Go From Here?

Now that we know how to start and stop, what do we do while we're there? Let's start with the function for setting a pixel, do_pixel(). This particular routine is very straight forward, being a single call to int86(). The parameters passed are the x,y coordinate of the pixel and the color you wish it to be (#defines for valid colors are given in graph.h, Listing 1).

Equally simple is the move() function, which updates the value of cur_x and cur_y to the x and y coordinates passed to it. Note that both the move() function and the draw() function I'll be talking about in a minute are absolute routines. That is, the x,y value passed is a screen coordinate, not an offset. These functions can be augmented by relative ones (which do use an offset) very easily. Using the move() function as an example, it would be:

```
void move_rel(xoff,yoff)
int xoff, yoff;
{
    int x, y;
    x = cur_x + (xoff);
    y = cur_y + (yoff);
    move(x,y);
}
```

A function draw_rel() would be identical, except calling draw(), instead of move().

I'll discuss draw() and clip() at the same time, since clip() is used by draw() to clip lines to the screen. The draw() function uses the Bresenham line algorithm to draw a line from the current x,y location to that specified. This algorithm is quick, since it uses only integer arithmetic to calculate which pixels are part of the line and avoids the "stair step" effect which plagues other integer based line drawing routines. An extended discussion of this routine can be found in Foley and Van Dam, pg. 433-436 (see References section below).

The clip() function is used to clip a line segment to a window, so that no time is wasted in calculating pixel positions which fall outside the window and are not displayed. The function uses the Cohen-Sutherland clipping algorithm (see Newman and Sproull, pg. 65-67) to calculate the length of the line which is visible within the window (in this case, the window is the same as the physical screen, the viewport). This clipping is an iterative process, where the line is successively truncated until both endpoints of the line are within the window. The function code() returns the 4-byte values which are used to determine when the line is within the window. In addition, the clip() function tests for the situation where line clipping results in a line where both end points are on the same side of the window and are external to it. In this case, the line cannot be displayed and the function returns.

Listing 1: Header File For Graphics Primitives

```
/* header file for graphic primitives */

#include            <stdlib.h>
#include            <stdio.h>
#include            <dos.h>

int cur_x, cur_y;                   /* current x, y coordinates */
int x_max, x_min, y_max, y_min;   /* max and min screen values */
int page, mode;                     /* video page and mode at startup */

static union REGS inreg, outreg;    /* unions for int86() */

/* defines for foreground colors */

#define         BLACK           0
#define         BLUE            1
#define         GREEN           2
#define         CYAN            3
#define         RED             4
#define         MAGENTA         5
#define         BROWN           6
#define         WHITE           7
#define         GREY            8
#define         LBLUE           9
#define         LGREEN          10
#define         LCYAN           11
#define         PINK            12
#define         LMAGENTA        13
#define         YELLOW          14
#define         BWHITE          15

void graf_init(), graf_term(), do_pixel(), move(), draw();
int clip(), code();
```

Listing 2: Graphics Primitives.

```
/**************************************************************************
*                                                                         *
*       graph.c :       graphics primitives for the IBM PC.               *
*                                                                         *
*       Copyright (C) 1986      Donald Howes                              *
*               All Rights Reserved.                                      *
*                                                                         *
*       This program may be copied for personal, non-commercial use only, *
*       provided that the copyright notice is included in all copies.     *
*                                                                         *
*       Note : most of these functions use IBM PC Video ROM BIOS calls to *
*          operate and are specific to the IBM PC and close compatibles.  *
*                                                                         *
**************************************************************************/

#include                "graph.h"

/*  graf_init   - initializes the screen

   description - sets the screen to 640x350 mode and initializes variables.
                 Also saves previous parameters for restoration.
```

C Column 27 Listing 2

```
*/
void graf_init()
{
        cur_x = cur_y = 0;                      /* set current x,y location */

        x_min = y_min = 0;                      /* minimum values for x and y */
        y_max = 349;                            /* maximum y value */
        x_max = 639;                            /* maximum x value */

        inreg.h.ah = 0x0f;                      /* get current display mode */
        int86(0x10,&inreg,&outreg);             /* ROM video call 10H */
        page = outreg.h.bh;                     /* monitor page to reset to */
        mode = outreg.h.al;                     /* monitor mode to reset to */

        inreg.h.ah = 0;
        inreg.h.al = 0x10;                      /* set EGA high res mode */
        int86(0x10,&inreg,&outreg);             /* 640 x 350 */
}       /* end of graf_init() */


/*      graf_term -             reset the system.

        description -   reset the system to the way it was before the
                                initialization call.
*/

vvoid graf_term()
{
        inreg.h.ah = 0;                                 /* reset the mode */
        inreg.h.al = mode;
        int86(0x10,&inreg,&outreg);

        if (page)               /* if page wasn't the first (0), reset */
        {
                inreg.h.ah = 5;         /* select display page */
                inreg.h.al = page;
                int86(0x10,&inreg,&outreg);
        }
}       /* end of graf_term() */

/*      do_pixel                - does a pixel

        description             - turns the given pixel to the specified color.
*/

void do_pixel(x,y,color)
int x, y, color;
{
        /* set the new color using int86() */

        inreg.h.ah = 0xc;
        inreg.h.al = color;
        inreg.x.cx = x;
        inreg.x.dx = y;
        int86(0x10,&inreg,&outreg);
}       /* end of do_pixel() */
```

C Column 27 Listing 2

```
/*      code - used by Cohen/Sutherland clipping algorithm.

        description - provides the four bit code needed to determine where
                      the line segment falls in relation to a viewport.
*/

int code(x,y)
int x, y;
{
        return(x<x_min)<<3 | (x>x_max)<<2 | (y<y_min)<<1 | (y>y_max);
}       /* end of code() */

/*      clip - clips a line.

        description - clips a line to a viewport using the Cohen/Sutherland
                      algorithm.
*/

int clip(x1,y1,x2,y2)
int x1, y1, x2, y2;
{
        void move();
        int code();
        int c1 = code(x1,y1);
        int c2 = code(x2,y2);
        int dx, dy;

        while (c1 | c2)
        {
                if (c1 & c2)
                        return;
                dx = x2 - x1;
                dy = y2 - y1;
                if (c1)
                {
                        if (x1 < x_min)
                        {
                                y1 += dy * (x_min - x1) / dx;
                                x1 = x_min;
                        }
                        else if (x1 > x_max)
                        {
                                y1 += dy * (x_max - x1) / dx;
                                x1 = x_max;
                        }
                        else if (y1 < y_min)
                        {
                                x1 += dx * (y_min - y1) / dy;
                                y1 = y_min;
                        }
                        else if (y1 > y_max)
                        {
                                x1 += dx * (y_max - y1) / dy;
                                y1 = y_max;
                        }
                }
                else
                {
                        if (x2 < x_min)
                        {
                                y2 += dy * (x_min - x2) / dx;
                                x2 = x_min;
                        }
                        else if (x2 > x_max)
                        {
                                y2 += dy * (x_max - x2) / dx;
                                x2 = x_max;
                        }
                        else if (y2 < y_min)
                        {
                                x2 += dx * (y_min - y2) / dy;
                                y2 = y_min;
                        }
                        else if (y2 > y_max)
                        {
                                x2 += dx * (y_max - y2) / dy;
                                y2 = y_max;
                        }
                }
        }
        move(x1,y1);
}       /* end of clip() */

/*  moove - performs move
        description - performs a move (does not draw) and resets the values
                      for cur_x and cur_y.
*/

void move(x,y)
int x, y;
{
        cur_x = x;
        cur_y = y;
}       /* end of move() */

/*      draw - draws a line using the Bresenham algorithm.

        description - draws a line from the current location to the location
                      referenced by (x2,y2). It also updates cur_x and cur_y.

*/

void draw(x2,y2,color)
int x2,y2,color;
{
        void do_pixel(), move();
    int dx, dy, incr1, incr2, incr3, d, x, x1, y, y1, xend, yend;

        clip(cur_x,cur_y,x2,y2);

        x1 = cur_x;
        y1 = cur_y;
    dx = abs(x2-x1);
    dy = abs(y2-y1);
```

C Column 27 Listing 2

```
  if (dy <= dx)
  {                          /* absolute value of slope is < 1 */
      if (x1 > x2)    /* start at point with smaller x co-ord. */
      {
          x = x2;
          y = y2;
          xend = x1;
          dy = y2 - y1;
      }
      else
      {
          x = x1;
          y = y1;
          xend = x2;
          dy = y2 - y1;
      }
      d = (dy >= 0) ? incr1 - dx : incr1 + dx;
      incr1 = dy << 1;
      incr2 = (dy-dx) << 1;
      incr3 = (dy+dx) << 1;

      do_pixel(x,y,color);

      while (x < xend)
      {
          x ++;
          if (d >= 0)
          {
              if (dy <= 0)          /* negative or zero slope */
                 d += incr1;
              else                  /* positive slope */
              {
                  y ++;
                  d += incr2;
              }
          }
          else
          {
              if (dy >= 0)
                 d += incr1;
              else
              {
                  y --;
                  d += incr3;
              }
          }
          do_pixel(x,y,color);
      }
  }
  else
  {
      if (y1 > y2)
      {
          y = y2;
          x = x2;
          yend = y1;
          dx = x1 - x2;
      }
      else
      {
          y = y1;
          x = x1;
          yend = y2;
                dx = x2 - x1;
      }
      d = (dx >= 0) ? incr1 - dy : incr1 + dy;
      incr1 = dx << 1;
      incr2 = (dx-dy) << 1;
      incr3 = (dx+dy) << 1;
      do_pixel(x,y,color);
      while (y < yend)
      {
          y ++;
          if (d >= 0)
          {
              if (dx <= 0)
                 d += incr1;
              else
              {
                  x ++;
                  d += incr2;
              }
          }
          else
          {
              if (dx >= 0)
                 d += incr1;
              else
              {
                  x --;
                  d += incr3;
              }
          }
          do_pixel(x,y,color);
      }
  }
      move(x2,y2);
}   /* end of draw() */
```

(C Column continued)

Graphics References

This has been a rapid introduction to graphics. For those who wish to delve a little deeper into the subject or want to try their hand at developing their own routines, here are some references you will find helpful.

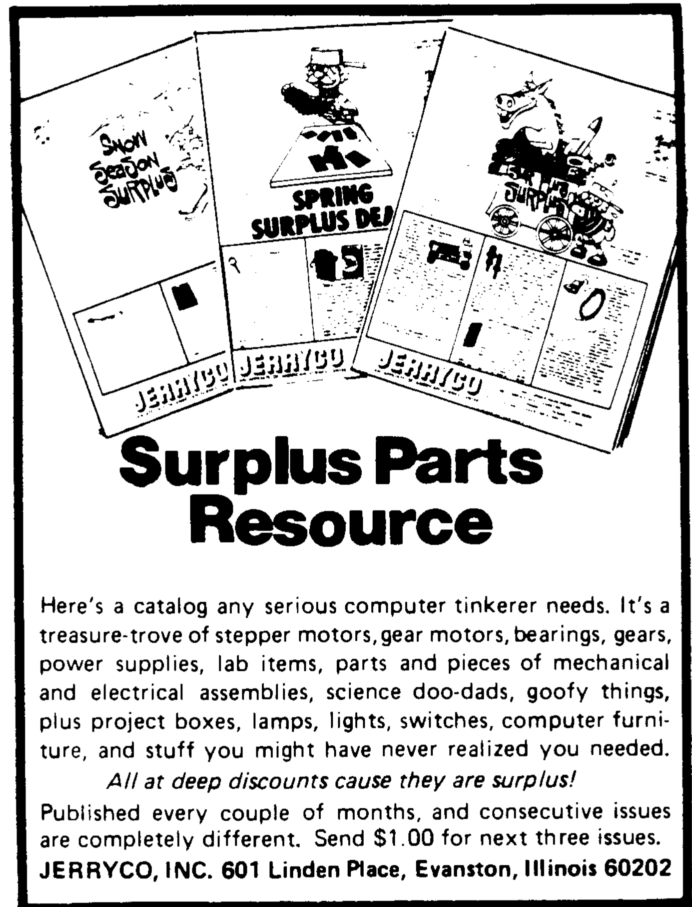
Bruce A Artwick, 1984, "Applied Concepts in Microcomputer Graphics", Prentice-Hall, Inc., Englewood Cliffs.

J.D. Foley and A. Van Dam, 1982, "Fundamentals of Interactive Computer Graphics", Addison-Wesley Publishing Company, Reading.

Steve Harrington, 1983, "Computer Graphics : A Programming Approach", McGraw-Hill Book Company, New York.

William M. Newman and Robert F. Sproull, 1979, "Principles of Interactive Computer Graphics", McGraw-Hill Book Company, New York.

That's it for this time. The next column will finish up the primitive package and we can move on to some interesting applications. ■



EVERYTHING YOU NEED... $279.00

Now it's easy to program the Heath-Zenith HERO-1® Robot with an Apple® II. HERO® Macros for the S-C Software 6800 Cross Assembler program in Heath's Robot Interpreter Language with easily remembered mnemonics. The HERO® Macros come with 30 pages of documentation.

[For example, the line 1130 MVWRIM GRIP OPEN 60 FAST instructs the HERO® to open his gripper 60 units at fast speed. Motor position is expressed in base-10.]

Transfer to HERO® with ROBI... an affordable interface for the robotics experimenter... is simple.

- ROBI is a complete package. No additional hardware required for Apple® or HERO®.
- ROBI installs quickly in an Apple® II, II+, or IIe. Once installed, no hardware changes are needed. Within minutes, you will be programming HERO®.
- With ROBI and the Cross Assembler, the programmer uses Apple®'s memory to write the program, and HERO®'s memory to run the program.
- Not "copy protected," archival copies may be made as needed.
- ROBI offers expansion potential.

VISA and MasterCard accepted

The Cross Assembler with HERO® Macros sells for $100.00; the ROBI Interface sells for $199.00. Both as a package — $279.00.

To order, or for more information, call (303) 670-6137.

BERSEARCH Information Services
26160 Edelweiss Circle
Evergreen, Colorado 80439