Chapter 7 Interactive Color Raster Graphics

Chapter Overview

Interactive Color Raster Graphics

Starting ICR Graphics Emulation

Using the ICR Protocol

- Description of the Protocol
- ASCII Encoding
- Run-Length Encoding Format
- Color Maps

ICR Graphics Windows

- Allocating Memory
- Copying a Graphics Window
- System Color Problems

Example ICR Program in C

Chapter Overview

This chapter introduces the Interactive Color Raster (ICR) protocol and describes how you may use this protocol in your programs to display color graphics with NCSA Telnet. In addition, the chapter describes how to control raster graphics windows, and display and manipulate color images. The chapter includes an example program that you may use as a template for designing programs that use the ICR protocol.

Interactive Color Raster Graphics

Interactive Color Raster (ICR) is a protocol for displaying raster graphics on your workstation screen. The ICR protocol controls its own windows through NCSA Telnet. It shares characteristics of the Tektronix graphics terminal emulation protocol. For example, escape sequences are used to control the display.

Using ICR, you can write mainframe programs to display color images in

their own windows on your Macintosh screen, and you can apply the full range of 256 colors out of a palette of 16 million colors to your graphics displays. The ICR protocol is intended for use on a Macintosh with 256-color capability.

Starting and Quitting ICR Graphics Emulation

To use ICR, you need a program that runs on the remote, or host, computer and gives all of the appropriate commands to conduct the ICR graphics emulation. To create an ICR program, work from the protocol description contained in the section entitled "Using the ICR Protocol" and the example program contained in the section entitled "Example Program for ICR in C."

When the protocol command for creating a window arrives from the host, NCSA Telnet creates a Macintosh window for it. All human-readable text continues to go to the VT102 window and the graphics commands are sent to the proper graphics window.

The ICR program on the remote computer may choose to take the window away itself. If it does not, you can dispose of a graphics window by clicking in the close box, which is located in the upper-left corner of the window's title bar. If you exit NCSA Telnet while some windows remain open, the windows close automatically.

Using the ICR Protocol

To use ICR, you write a program that issues graphics commands to NCSA Telnet. NCSA Telnet receives these commands, interprets them, creates or destroys windows, sets the color environment, or displays raster graphics as the program directs.

To ensure that NCSA Telnet can determine the difference between regular text and ICR graphics, all ICR graphics sequence commands begin with the escape sequence ESC^ (escape, caret).

Description of the Protocol

Each ICR command has the form:

ESC^X; ***parameters*** **^** ***data***

where

- X is one of the command characters listed in Table 7.1 and fully described in Table 7.2.
- ^ is the caret character (ASCII 94).
- *parameters* is one or more of the parameters of X. The parameters for each command are listed in Table 7.1.
- the command is terminated with a caret (^).

- each command may be followed by a data stream which goes with it.

The parameters are determined by the command character that is used (see Table 7.2). If your program omits the parameters, then NCSA Telnet supplies default values for the parameter values. Parameters are always printable ASCII and are delimited by ';'. For commands that require data, the data follows the command.

Table 7.1 ICR Commands

Command	Operation
W	Creates a window
D	Destroys a window
M	Loads a color map palette of up to 256 colors from a 24-bit palette into the graphics window
R	Indicates that run-length encoded data follows
P	Indicates that pixel data follows
I	Indicates that IMCOMP compressed data (4:1 compression) follows

Table 7.2 Commands and Command Parameters Described

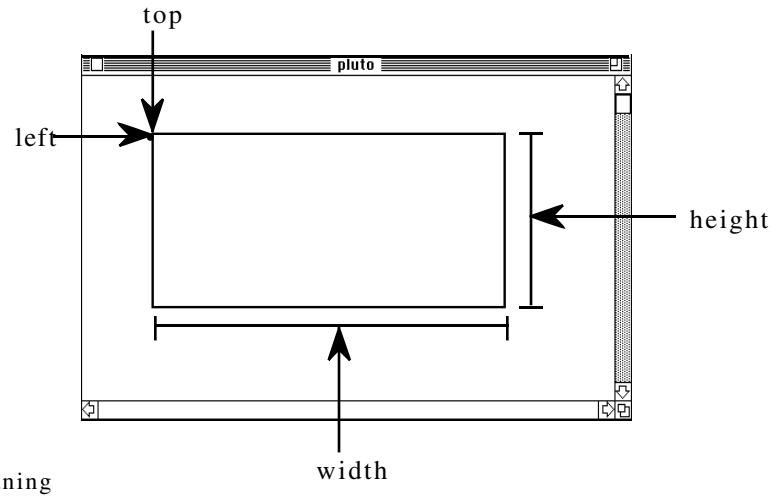
Command	Parameters	Description
W	left; top; width; height; display; windowname	Creates a window at the given location on the screen, where 0, 0 is the upper-leftmost corner of the screen. • Left, top, width, and height are integers specifying a location and size on the screen (see Figure 7.1). • Display is an integer indicating the hardware screen number (for machines with more than one screen—the parameter is not applicable for Macintoshes). • Windowname is a string used to distinguish multiple windows. The windowname assigned to a window is used by all of the other commands to specify which window to use.
D	windowname	Destroys a window by physically removing it from the screen and memory. • Windowname is the unique name assigned to a window when it is created by the W command.
M	start; length; count; windowname	Loads a color map or portion of one into the display hardware. NCSA Telnet assumes that palette entries are 8-bit R, G, and B, 3 bytes per entry, in that order. The default palette is a straight grey-scale ramp, where 0=black and 255=white. (See the section entitled "Color Maps.") • Start is an integer indicating the first entry to change. • Length is an integer indicating the number of entries to change.

4		• Count is an integer indicating the total number of bytes that are in the data portion. Count is followed by the data for the command.
R	x; y; expand; length; windowname	Specifies that the data to follow is run-length encoded. (See the section entitled “Run-Length Encoding Format.”) • x, y are integers indicating the point where the raster line starts and the data follows for length bytes of encoded data. • Expand is an integer indicating the number of times each dimension is to be expanded on the local screen. For example, an expand value of 2 makes the picture four times larger. • Length is an integer indicating the encoded length of the data, in bytes.
P	x; y; expand; length; windowname	Specifies that the data to follow is pixel data. • x, y are integers indicating the point where the raster line starts and the data follows for length bytes of pixel data. • Expand is an integer indicating the number of times each dimension is to be expanded on the local screen. For example, an expand value of 2 makes the picture four times larger. • Length is an integer indicating the length of data, in bytes, which is the same as the number of pixels to be displayed.

Table 7.2 Commands and Command Parameters Described (Continued)

Command	Parameters	Description
I	x; y; expand; length; windowname	Specifies that the data to follow is encoded with the IMCOMP compression scheme. The M command MUST be used before the picture displayed with the I command will appear correctly. • Length is an integer indicating the number of pixels per line, though one ‘I’ call represents four lines of data. The IMCOMP compression is a 4x4 SQUARE compression scheme, so each “line” of data will appear as four lines of pixels on the screen. • Y is required to increment the line numbers by fours: 0, 4, 8, 12, 16, etc.

Figure 7.1 Meaning of the Left, Top, Width, and Height Parameters



Integer	Meaning
left	the pixel value of the horizontal, or x, location of the upper-left corner of the graphics window
top	the pixel value of the vertical, or y, location of the upper-left corner of the graphics window
height	the number of pixels that comprise the vertical height of the graphics window
width	the number of pixels that comprise the horizontal width of the graphics window

ASCII Encoding

NCSA Telnet assumes that all of the parameter values are printable ASCII except ESC, which is an allowable exception on most login data streams. This means that the parameters require no special encoding, but the data values need help.

Your ICR program must encode 8-bit data values into printable ASCII for transmission. When possible, the values that fall in the printable ASCII range are passed untouched and all values outside that range are encoded as two bytes.

The following encoding works for all characters 0-255, as shown in Table 7.3.

Input:	realchar
Transmission:	specialchar followed by transchar
Encoding:	specialchar=realchar div 64 + 123
transchar=realchar	mod 64 + 32
Decoding:	realchar=(specialchar - 123)*64 + (transchar - 32)

Table 7.3 Encoding Data Values into Printable ASCII

Special	Range
123	0-63
124	64-127
125	128-191
126	192-255

Because all encoded characters are preceded by a char in the range 123-126, all regular characters that are 32-122 (inclusive) can be sent without encoding.

Warning: On CTSS, trailing spaces are trimmed. Consequently, the values 0, 32, 128, and 192 should be avoided, because they code to <special> <space>.

NOTE: In the specifications, all data lengths and counts refer to the protocol data, not the ASCII encoded data. The length fields for R, P, and M all reflect the length of the data on the originating machine before it is encoded.

Run-Length Encoding Format

The data for the run-length encoded line is first run-length compressed and then ASCII encoded. The process for deciphering, therefore, is first to decode the ASCII to binary and then to decode the run-length binary data.

Using all eight bits of the byte stream which represents the pixels in a given RLE line, start with the control character. (*n*) is the low seven bits of the byte. The high bit represents whether the following (*n*) characters are reproduced exactly (high bit=0) or whether the following single character is reproduced (*n*) times (high bit=1).

Input: 1 1 1 1 23 23 23 234 112 33 44 55 42 42 42 42
Tokenized: (128+4) 1 (128+3) 23 (5) 234 112 33 44 55 (128+4) 42
Alternate count, data, count,data

After coding into this tokenized form, the data length for the R command is known. (The length is 12 in this example). Even though the ASCII encoding takes place after this step, use the length value from this step.

ASCII result: 125 36 123 33 125 35 123 55 123 37
126 74 112 33 44 55 125 36 42

Color Maps

You can manipulate the color table for the local display with the M command. The format for the color map data is a series of color map entries. Each color map entry is three bytes, one Red, one Green, one Blue. For example, to set entries 3 through 7 of the color table, the following M command might be used:

ESC^M;3;4;12;wind^*RGBRGBRGBRGB*

where the RGBRGB... data is the list of byte values for the new entries in RGB order. The actual data transmitted over the line still has to be ASCII encoded, but the data starts out in this form. Note that the count field, which is 12 in this example, is always exactly three times the length value, which is 4 in this example.

ICR Graphics Windows

Raster graphics windows require a lot of memory—one byte for each pixel

in each graphics window on the screen. If there is insufficient memory remaining to open a new window, NCSA Telnet informs you with an alert dialog and does not create the window.

Allocating Memory

If you are using MultiFinder, you can set NCSA Telnet's allocated memory size to a larger value to prevent running out of memory. For example, if you need space for two 256x256 image windows, you need to increase the memory for NCSA Telnet by 128K—256 bytes times 256 bytes (or 64K) for each window.

Copying a Graphics Window

You can copy the contents of an ICR window onto the Macintosh Clipboard, and paste it into a program that is capable of pasting color images.

To copy the contents of a graphics window:

1. Click in the graphics window to make it frontmost.

2. Choose Copy from the Edit menu. Now you can paste the graphic into another Macintosh application.

System Color Problems

Image windows utilize the colors available for display on your Macintosh screen. When you close graphics windows, the system does not always restore the color environment to its original state, causing other windows to appear with incorrect colors. We are currently working to minimize the effects of NCSA Telnet and ICR graphics on your system's color table.

NOTE: Pressing CONTROL-C, or other methods of interrupting ICR commands, may make NCSA Telnet appear to "lock up" (see also "Telnet Options" in Chapter 4). When this occurs, press RETURN several times or enter commands until the VT102 window resumes activity. It may help to remember that when a drawing command is issued, NCSA Telnet expects an influx of a certain number (often hundreds) of bytes of image data to be used to finish drawing the current line.

Example Program for ICR in C

The sample program shown in Figure 7.2 is included on the distribution disk. It produces a test pattern on your screen if you are running an active ICR-equipped NCSA Telnet. If you do not have ICR, it produces thousands of encoded characters on your display.

Figure 7.2 Sample C Program

```
/*   icrtest
*
*  Produces a test pattern on an ICR compatible display. Demonstrates and provides example
*  code for writing ICR programs.
*
*  National Center for Supercomputing Applications
*  University of Illinois, Urbana-Champaign
*
```

```
*  by Tim Krauskopf
*  This program is in the public domain.
*
*/
#include <stdio.h>

int
       xdim=0,ydim=0;        /* size of image on disk */


char
       *malloc(),
       *testimage,
       rgb[768];                    /* storage for a palette */


main(argc,argv)
       int argc;
       char *argv[];
       {
       register int i,j;
       register char *p;

       puts("Creating test pattern");

       xdim = 150;
       ydim = 100;

       if (NULL == (testimage = malloc(xdim*ydim)))
              exit(1);

/*
```

Figure 7.2 Sample C Program (Continued)

```
*  Make the test image in a strange pattern.
*/
       p = testimage;

       for (i=0; i<ydim; i++)
              for (j=0; j<xdim; j++) {
                     *p++ = 50 + (((i & 0xfffffff8) * (j & 7))>>2);
       }

       puts("Displaying test pattern with the Interactive Color Raster protocol");

       rimage(0);            /* display remote image with [palette] */

}


/*************************************************************************/


/*  rimage
*  Remote display of the image using the ICR.
*  Just print the codes to stdout using the protocol.
*/

rimage(usepal)
       int usepal;
       {
```

```
        int i,j,newxsize;
        char *space,*thisline,*thischar;
        register unsigned char c;


/*
*  Open the window with the W command.
*/

(void)printf("\033^W;%d;%d;%d;%d;0;test window^",0,0,xdim,ydim);

/*
*  If a palette should be used, send it with the M command.
*/
        if (usepal) {
                (void)printf("\033^M;0;256;768;test window^");  /* start map */

                thischar = rgb;
                for (j=0; j<768; j++) {
                        c = *thischar++;
                        if (c > 31 && c < 123) {
                                putchar(c);
                        }
                        else {
                                putchar((c>>6)+123);
                                putchar((c & 0x3f) + 32);
                        }
                }
        }

/*
```

Figure 7.2 Sample C Program (Continued)

```
*  Send the data for the image with RLE encoding for efficiency.
*  Encode each line and send it.
*/
        space = malloc(ydim+100);
        thisline = testimage;

    for (i = 0; i < ydim; i++) {
        newxsize = rleit(thisline,space,xdim);
                thisline += xdim;                           /* increment to next line */

         (void)printf("\033^R;0;%d;%d;%d;test window^",i,1,newxsize);

         thischar = space;
         for (j = 0; j < newxsize; j++) {


/*********************************************************************/


/*  Encoding of bytes:
*
*  123 precedes #'s 0-63
*  124 precedes #'s 64-127
*  125 precedes #'s 128-191
*  126 precedes #'s 192-255
*  overall:  realchar = (specialchar - 123)*64 + (char-32)
*            specialchar = r div 64 + 123
```

```
*           char = r mod 64 + 32
*/


/*****************************************************************/


                    c = *thischar++;      /* get byte to send */

                    if (c > 31 && c < 123) {
                           putchar(c);
                    }
                    else {
                           putchar((c>>6)+123);
                           putchar((c & 0x3f) + 32);
                    }
        }
       }

       free(space);
}


/*****************************************************************/


/*  rleit
*
*  Compress the data to go out with a simple run-length encoded scheme.
*
*/
```

Figure 7.2 Example C Program (Continued)

```
rleit(buf,bufto,len)
       int len;
       char *buf,*bufto;
       {
       register char *p,*q,*cfoll,*clead;
       char *begp;
       int i;

       p = buf;
       cfoll = bufto;                      /* place to copy to */
       clead = cfoll + 1;

       begp = p;
       while (len > 0) {                   /* encode stuff until gone */

              q = p + 1;
              i = len-1;
              while (*p == *q && i+120 > len && i) {
                     q++;
                     i—;
              }

              if (q > p + 2) {             /* three in a row */
                     if (p > begp) {
                            *cfoll = p - begp;
                            cfoll = clead;
                     }
                     *cfoll++ = 128 | (q-p);             /* len of seq */
```

```
                        *cfoll++ = *p;          /* char of seq */
                        len -= q-p;                     /* subtract len of seq */
                        p = q;
                        clead = cfoll+1;
                        begp = p;
                }
                else {
                        *clead++ = *p++;        /* copy one char */
                        len--;
                        if (p > begp + 120) {
                                *cfoll = p - begp;
                                cfoll = clead++;
                                begp = p;
                        }
                }

        }
/*
*  fill in last bytecount
*/
        if (p > begp)
                *cfoll = 128 | (p - begp);
        else
                clead--;                                /* don't need count position */

        return((int)(clead - bufto));           /* how many stored as encoded */
}
```