PAINTING TUTORIAL NOTES

Alvy Ray Smith
Computer Graphics Lab
New York Institute of Technology

SIGGRAPH '79
6-10 Aug 1979
Chicago IL

Published as Tutorial Notes at SIGGRAPH '79, '80, '81, and '82 and Infotech '79 (London). This document was reentered by Alvy Ray Smith and his son Sam in Microsoft Word on 12 Sep 2000. Spelling and punctuation are generally preserved, but trivially minor spelling errors are corrected. Otherwise additions or changes made to the original are noted inside square brackets or in footnotes.

INTRODUCTION

One of the most basic picture-making tools in computer graphics (specifically, color raster graphics) is moving or copying an image from one location to another, from one type of memory device to another type. What we shall call painting is an example of this basic operation. One image, called the brush, is copied from disk memory into main and then repeatedly copied from there into a special type of visible memory called a framebuffer at a changing position determined by the user's hand on a tablet. All the variations on simple copying available on a computer are, of course, available for the special case of painting. We shall detail several of these.

FRAMEBUFFERS

The memory devices of greatest interest in the painting context arre magnetic disks, main memory, and framebuffers. Only the framebuffer may need to be described in some detail. It is a piece of memory large enough to hold one video frame in digital form. The framebuffer contents can be viewed on an RGB (red, green, blue) monitor thirty times a second—ie, at video rate. Thus a framebuffer—also called a frame store, or a picture memory—is ordinary digital memory with the addition of circuitry to allow us to see its contents as a 2-dimensional display. Standard US video displays 480-486 lines. Comparable resolution in the horizontal direction dictates that a framebuffer contain approximately one-quarter million picture elements, called pixels. However, there are framebuffers of lower resolution available to the home hobbyist—eg, 64x64, 128x128, and 256x256. Hopefully there will be framebuffers of much higher resolution available soon—eg, 1000x1000 or 2000x2000. Some optional features a framebuffer may possess are hardware cursors, extra bit planes for graphic overlays, a colormap, video magnification, and programmable special purpose processors. Some of these—in particular, cursors and colormaps—are so important that they can be considered mandatory.

CURSORING

A person paints into a framebuffer under tablet control. Physically this means that he moves his hand about on a tablet while looking at a video monitor. To avoid looking at his hands, he must have an indicator of his tablet position on the monitor screen. One way to accomplish this is software: The point coordinates of the stylus he moves about on the tablet are sent continually to the cpu. A routine there writes a mark into the framebuffer so it appears on the monitor screen in a position corresponding to his tablet position. This mark, called a cursor, must be easily visible and must not destroy the framebuffer contents where it is written.

Displaying the cursor is a time-consuming, cycle-eating process. Hence it is convenient to have a hardware cursor built into the framebuffer. The shape of the cursor is user-defined. Its display is performed by hardware which manages to preserve the framebuffer contents under the cursor in one of several ways. It may bitwise complement the framebuffer contents and then complement again before moving to a new position. Or it may be generated in a separate memory and mixed into the video at the appropriate position. If it is not a complementing cursor, then its color may be user-selected to keep it from disappearing on certain backgrounds.

The most popular cursor shape at NYIT is an upturned arrow. The pixel at its tip is the current location of the stylus on the tablet. Other cursors typically seen are X's, +'s, or O's, centered on the current position, but any convenient shape will do. Typical hardware cursors permit easy redefinition of the cursor shape as well as on/off control (turn cursoring on/turn it off).

For software cursoring, it is convenient to have the framebuffer tell the user when the video vertical blanking occurs. (Every thirtieth of a second, a video monitor inhibits its electron guns while they are repositioned to the top of the tube in preparation for the next downscan. This is the vertical blanking interval.) The cursor can be written during this blanking so as to appear solid on the display. If it is not synchronized with the video, it will beat against the video frequency and seem to ripple, or completely disappear in some positions.

COLORMAPS

So far we have not discussed the depth of a framebuffer—the number of bits per pixel—nor how the contents of a framebuffer memory are converted to a video signal by the video processor of the framebuffer. Although home-hobbyist framebuffers exist with 1, 2, and 4 bits per pixel, the framebuffer is not a truly potent tool with less than 8 bits per pixel. This is because a human can normally distinguish between 128 and 256 levels of one color. So with 8 bits a smooth-shaded monochrome picture can be produced. It can be shown [1] that 16 bits per pixel suffice for full color pictures intended for standard US video broadcast. For full color RGB monitors, 24 bits are typically used. For sophisticated applications at NYIT, 32-bit framebuffers are used—8 bits each for R, G, and B plus additional 8 bits for matting.

In all cases, the video processor must convert the number stored in a pixel into a set of three voltages for the driving the guns of its video monitor. The as-

signment, or mapping, of numbers to voltages could be hardwired into the framebuffer but usually it is specified by a table which the user may change under program control. This table is called a colormap, or some related name. So thirty times a second, the video processor scans through the framebuffer memory. At every pixel it reads the number stored there and uses it as an index into the colormap. The set of three numbers stored for the table in that index are the RGB gun settings. At NYIT we have two types of colormap for 8-bit framebuffers. Thus both have 256 entries, where each entry consists of three numbers. In one case, each of the three numbers is 12 bits longs; 8 bits in the other. These are described as 8 bits in / 12 bits out or 8 bits in / 8 bits out, respectively. The 12-bit-out colormaps are useful for color compensation when final output is film instead of video. Typically a colormap is changed during vertical blanking so as to appear to be an instantaneous change of color.

COPYING IMAGES

As stated in the introduction, copying images from memory to memory is a basic picture synthesis tool of color computer graphics. For framebuffer / minicomputer combinations, this is usually in the form of copying a picture in a framebuffer to a disk file (called saving the picture) or copying a disk file to a framebuffer (called restoring the picture). Another common copying instance is from one framebuffer to another. For very small pictures or for cpu's with large address spaces, we might speak of copying from framebuffer to main memory or from disk to main memory as well. (In fact, for very large address spaces, we can speak of virtual framebuffers.)

We usually don't move a big block of data from one place to another without doing something to it. For example, we might lerp—also spelled lirp—one picture being restored from disk file to the picture already in a framebuffer. Lerp is short for "linear interpolation", that very common computer graphics function given by

$$\text{LERP}(\text{alpha}, A, B) = \text{alpha} * A + (1 - \text{alpha}) * B$$

where alpha is between 0.0 and 1.0. For a given alpha, the lerp of two pictures is a mixture of both of them. By varying alpha from 0.0 to 1.0, one can obtain a cross-dissolve from one picture to the other.

For another example, a third picture may be used to control the combination of two others. If this control picture consists of only two values—say 0 and 1—it can be used as a mask. A pixel in picture A replaces the corresponding pixel in picture B only where the corresponding pixel of picture C is 1. If picture C has many values, say 256, then each of its pixels may be interpreted as an alpha for the pixel-by-pixel lerp of picture A to picture B. This is called matting, or picture C is a matte.

PAINTING

Painting may be considered to be a tablet-controlled copying of an image from one memory device to another. Thus all the variations of this basic operation, some of which were described above, apply to painting as well. The simplest form of painting then is a straight copy of a small picture called the brush,

from main memory to the position in a framebuffer specified by a user with a tablet. This type of "painting" is called picture or rubberstamping. It becomes useful with the introduction of the notion of transparency. One value in the brush is designated transparent which means that it is not written to the framebuffer but its position is skipped over instead. Picture painting is useful for creating textures. For example, a brush which is several blades of grass on a transparent background can be used to paint a lawn or meadow.

The type of painting from which "painting" derives its name is the following simulation of what we normally think of as painting in the real world: The brush is thought of as a mask. Its transparent value, if it has one, is 0, say, and all other values are thought of as 1's. Then it is used, under tablet-controlled placement, to mask picture A into the picture already in the framebuffer, picture B, where picture A has all pixels of the same color. Equivalently, the user selects a color which is written into the framebuffer everywhere the corresponding brush pixel is not transparent.

There are many, many variations on the basic painting modes described above. Some of these are described in [2], so we refer the reader to this paper and proceed to describe one last important component of a basic paint program, the selection of colors. We also refer the reader to the same paper for details of a full-fledged paint program. This would include many other features than those discussed here: filling, magnification, saving and restoring pictures, making brushes, making colormaps, histories, drafting aids, etc.

PALETTES

The mode of painting which emulates real-world painting requires the user to select a color with which to paint. A convenient way to do this is provided by the display of a palette. For example, in an 8-bit framebuffer, the palette might be a 16x16 array of all 256 pixel values. At NYIT the typical palette is 4 rows of 64 small squares each, arranged in order of increasing pixel value, left to right then top to bottom. The palette may be constantly displayed on the screen or caused to appear only when necessary, temporarily overwriting the framebuffer contents. The user may be forced to select the color from anywhere in the framebuffer, including the palette. At NYIT, the philosophy has been to use a disappearing palette so as not to destroy the balance or composition of the displayed artwork, and the user is allowed to pick color from anywhere.

For deep framebuffers—eg, 24 bits per pixel—display of all possible colors on a palette is impossible. So a palette is used to display some convenient subset of the colors, say 256 of them. The user may change the selection of these colors at will.

REFERENCES

1. Alvy Ray Smith, *YIQ vs RGB*, NYIT Tech Memo 9, 2 Apr 1979.

2. Alvy Ray Smith, *Paint*, NYIT Tech Memo 7, 20 Jul 1978.