Behavior Research Methods, Instruments, & Computers
1986, 18 (6), 587-594

Use of a raster framebuffer in vision research

ANDREW B. WATSON, KENNETH R. K. NIELSEN, ALLEN POIRSON, ANDREW FITZHUGH, AMYJO BILSON, KHANH NGUYEN, and ALBERT J. AHUMADA, JR.
NASA Ames Research Center, Moffett Field, California

For the study of human and animal vision, the video framebuffer is the only technology that is capable of displaying two-dimensional images with precise control of contrast, luminance, and display timing. The video framebuffer also allows precise control of color. However, this device is not designed for precise psychophysical displays, and techniques must be developed to use them in this role. In this paper, we describe a number of basic methods used in our lab. In order to be concrete, we use an Adage RDS-3000 raster display system (Adage, 1982) as an example, since that is the device we have actually used. The Adage is hosted by a PDP-11/73 under the Venix operating system. The principles generalize to other machines. Where it clarifies the issues, we have shown the syntax of the software routines involved.

Modern study of the properties of human and animal vision often requires a device capable of displaying two-dimensional images with precise control of contrast, luminance, and display timing. Precise control of color is also desirable. The video framebuffer is the only technology that provides these capabilities. We begin with a brief introduction to the basics of framebuffers and raster displays; a more complete introduction is given in the *Raster Graphics Handbook* (The Conrac Corp., 1985).

A framebuffer (FB) is a block of memory and specialized hardware that feeds the contents of the memory to a raster display (see Figure 1). FB memory is organized into P planes, each of N rows $\times$ M columns $\times$ b bits. In our system, there are four planes, red, green, blue, and alpha, each 1,024 wide $\times$ 1,024 high $\times$ 8 bits deep. A raster display is a system in which a beam traverses a screen in a regular pattern, usually in horizontal lines, from left to right, and with successive lines from top to bottom. The complete set of lines forms a rectangular *raster* on the screen. As the raster is painted on the screen, selected values are read out of the FB, transformed digitally in various ways, and passed through a digital-to-analog converter (DAC). The resulting voltage is used to brighten or dim the beam. Each separately controllable spot on the screen is known as a pixel. In our configuration, the display is 512 $\times$ 512 pixels. By setting the contents of the framebuffer to appropriate values, one can make an image appear on the screen. In a noninterlaced configuration, such as the one we describe here, the raster is typically refreshed (redrawn) once every $1/60$ of a second.

Several components may intervene between the FB and the DAC:

1. **Video path.** The data travel from the FB to the monitor along a *video path*. Various devices intervene along this path to modify the data, thus controlling the final appearance of the display screen.

2. **Frame Buffer Controller (FBC).** This device controls the manner in which the data are read out of the FB and placed on the data path. Two features of the FBC are particularly important. The *window* defines the region of the FB from which data will be read (see Figure 2). If many images are stored in memory, the window determines which image will appear on the screen. The window is typically specified by the coordinates of the upper left corner of the selected FB region, **win_x** and **win_y**. If an image is to be moved, the motion is effected by scrolling the window across the FB. The second important control is the *viewport*, which controls where on the screen the selected data will appear. It is defined by the coordinates of the upper left corner (**vw_x, vw_y**) and by width and height (**vw_width, vw_height**). Screen pixels outside the viewport are assigned a constant value (typically 0). By setting the appropriate entry in the *look-up tables* (LUTs; see below), a user may set this background area to a desired luminance. The viewport thus allows one to define and display images much smaller than

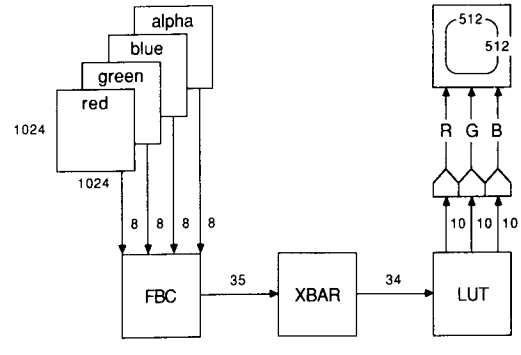

Figure 1. Overview of a raster framebuffer display system. Four planes of memory feed a video path containing a framebuffer controller, a crossbar switch, and a lookup table.

Reprint requests should be mailed to Andrew B. Watson, MS 239-3, NASA Ames Research Center, Moffett Field, CA 94035.

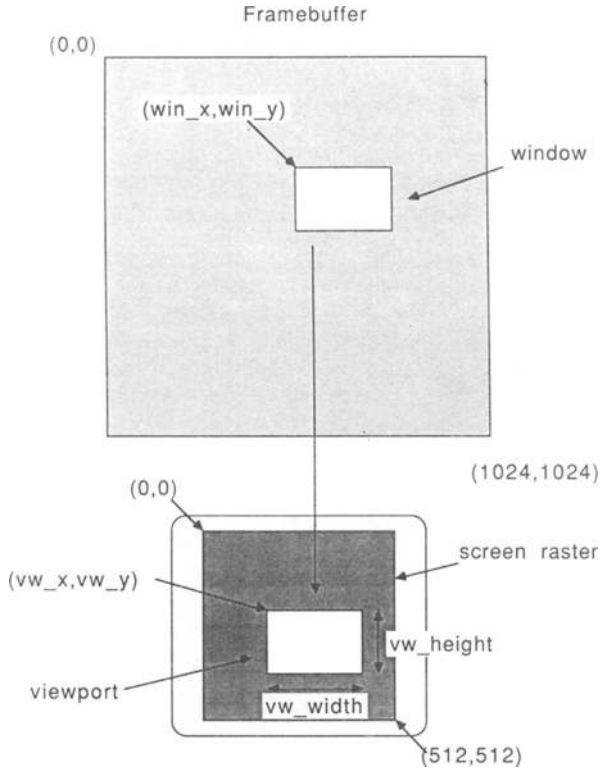

Figure 2. Illustration of the concepts of the window and viewport.

the full size of the screen. This, in turn, allows one to store a larger repertoire of images in the FB.

In our configuration, the video path, as it leaves the FBC, is 35 bits wide: 0–7 from the red plane, 8–15 from green, 16–23 from blue, and 24–31 from the alpha plane. The alpha plane, which for most purposes may be regarded as simply another plane of memory, derives its name from its use as a means of superimposing alphanumerics on an image. Bit 32 is the *in-cursor* bit, which is 1 whenever the pixel in question is in the *cursor*. The cursor is a user-definable 32×32 pixel×1 bit image that may be independently positioned anywhere in the FB. It is useful as a nondestructive fixation point that may be turned on and off and moved about without altering the FB contents. Bits 33 and 34 are the *LUT page bits*, which will be discussed below.

Crossbar switch (XBAR). A crossbar switch allows one to remap bits in the video path. For example, the crossbar allows one to change the video path bit assignments described above to map the red plane (bits 0–7) to bits 8–15. When using a monochrome display, a typical application of the crossbar is to route data from a selected plane (red, green, blue, or alpha) to the single channel driving the monochrome monitor. The video path leaving the XBAR is 34 bits wide (four 8-bit planes, in-cursor bit, LUT page bit).

Look-up tables (LUTs). The data in each of the lower three 8-bit planes of the video path act as addresses into a set of three tables, each containing 256 entries of 10 bits each. The three selected entries are sent to three 10-bit DACs, which, in turn, drive the monitor. As noted below, these tables provide a means of linearizing the display, of rapidly changing the contrast of the display without changing the FB contents, and of selecting which bit planes are to be shown. In the Adage RDS-3000, there are, in fact, 1,024 entries in each table, divided into four separate tables. The in-cursor bit and the LUT page bit select which of the four tables is used. As noted below, the presence of two LUT pages (for out-of-cursor bits) may be essential to real-time digital contrast control.

Loading images into the FB

Images are typically stored on the file system of the host computer. Before an image can be displayed, it must be copied from the file system to the FB. We do this with the routine:

image_n = load_image(file_name, start_frame, end_frame)

This routine does three things: (1) it transfers the image data from the file "**file_name**" to a region of the FB; (2) it associates an *image number* (**image_n**) with the FB region; and (3) it creates a data structure, associated with the image number, that describes the location and size of the FB region. The image number is the means used in subsequent routines to refer to a particular image. Image numbers are assigned in sequence from 0. The set of structures created may be thought of as an *FB map*. Some thought is required in packaging the images successively into the FB, especially if both monochrome and color images are intermixed.

Because the file format we use allows storage of multiple frames (as in a movie; see below), the routine allows one to specify starting and ending frame numbers (**start_frame, end_frame**). The lack of other arguments in this call indicates that the image file contains within it information regarding its width and height. This information is typically contained in a header of some standard sort. We have used the HIPS header in the current implementation (Landy, Cohen, & Sperling, 1984).

Two other routines have proven useful in our work. They are **save_fb(filename)**, which saves the FB map into a file, and **restore_fb(filename)**, which reads the FB map from a file into the appropriate data structures.

Because the transfer from host file system to FB is typically slow (several to tens of seconds), it is often necessary to load all images into the FB before the start of the experiment. Whether one takes this approach will depend on the speed of the transfer, the size of the FB, the ability to display selected regions of the FB, and the demands of the experiment. One step we have taken to accelerate the image-loading process is to create a dedicated *raw partition* on the Winchester disk that also houses the host file

system. This is, in effect, a separate file system into which images may be placed and from which they may be transferred to the FB without the substantial overhead of the host file system operations. Even with this acceleration, a $256^2 \times 8$ bit image takes about 2 sec to load. (The loading time is dependent upon the size of the host buffer allocated for the transfer.)

LUMINANCE CALIBRATION

Luminance calibration requires measuring the pixel luminance produced by each possible digital value, D, sent to the DAC. This value is not independent of the luminance of other pixels, but unfortunately completely characterizing these interactions is a formidable task. It is generally true that if some precautions are taken, the departures from independence will be modest. We measure the luminance in two ways: (1) via the digital readout of a photometer, and (2) by integrating the analog output of the photometer. The latter procedure allows us to automate the calibration of the monitor.

Integrating the Analog Luminance Waveform

As the monitor beam scans the screen, it passes momentarily under the view of the photometer. The analog output of the photometer reflects the duration of this passage and the persistence of the phosphor. An example is pictured in Figure 3. The details of this waveform are not important here, except that its integral is proportional to luminance. To compute this integral, we feed the analog signal into an analog-to-digital converter (ADC). During a single frame (16.6 msec), a sequence of 160 samples are taken at intervals of 100 μsec. The sum of these samples is taken as an estimate of the luminance, scaled by a constant. This constant can be determined by comparing the photometer readout and computed integral at some

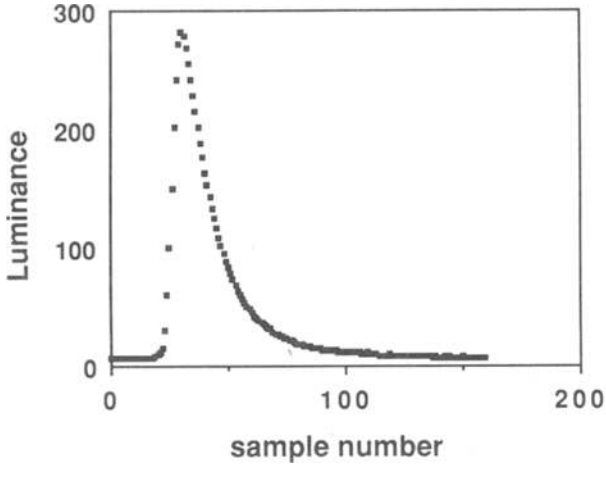

Figure 3. Samples taken by an analog-to-digital converter of the analog signal produced by a photometer during one frame of the raster. The interval between samples is 100 μsec. The time-average raster luminance is proportional to the integral of the waveform.

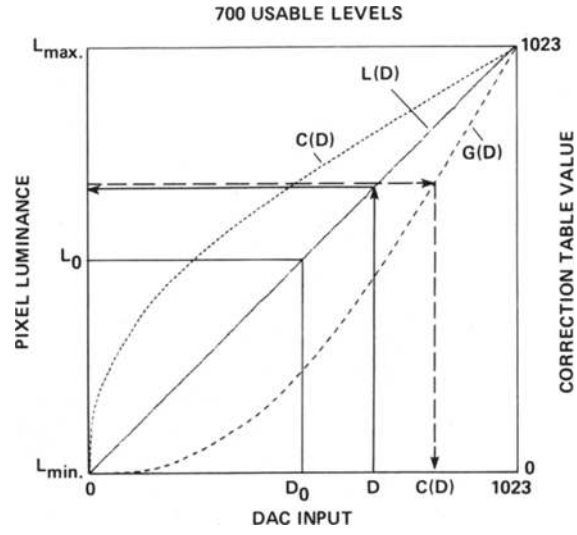

Figure 4. Construction of the correction table.

value. This sampling and computation are implemented in the routine

n =read_lum(frames, pause)

which waits for **pause** frames and then computes the (unscaled) luminance averaged over **frames** frames. The pause serves to allow any large transients to settle and is not ordinarily needed unless very large changes in luminance are made. The averaging may be used to reduce the noise in the measurements. As an example, in measuring the gamma function for the unattenuated red channel (see below), we use **frames** = 3, **pause** = 0, whereas for the attenuated blue channel, we use **frames** = 9, **pause** = 0, because the signal:noise ratio is smaller.

MEASURING THE GAMMA FUNCTION

The relationship between the voltage produced by the DAC and the luminance of the pixel is usually nonlinear. Traditionally, this function has been modeled as a power function with an exponent of *gamma*, and, for this reason, the function is often called the *gamma function*. Our procedure for measuring the gamma function is to load into the FB an image consisting of vertical black and white (0 and 255) stripes, each about 1.5 times the width of the photometer acceptance area. The photometer is centered on one stripe (e.g., 255). The value of the 255th entry in the LUT is then set in succession to all of its possible values from 0 to 1,023, and after each value is set, the resulting luminance is measured using the integrating technique described above. The sort of data obtained from this procedure is shown in Figure 4. As each value is placed in location 255 of the LUT, 1,023 minus that value is placed in location 0 of the LUT. Thus, as the stripe under the photometer gets brighter, the adjacent stripes get darker. This step is taken to approximately equalize the display power supply drain over the various settings.

The gamma function collected in this way will be used below to set the contrast of the display. It is important to realize that the precise shape of this function may vary with the monitor, with the settings of the brightness and contrast knobs on the monitor, with the gun (R, G, or B) in use, and from day to day. Since our automated procedure takes only about 5 min, we calibrate before each experimental session.

LINEARIZATION AND CONTRAST CONTROL

If the data in the FB are to represent luminance values, then the relationship between FB data and pixel luminance must be linearized. This may be accomplished by means of look-up tables between the end of the video path and the DACs (Catmull, 1979). First, we introduce the notation. Many of these quantities are indicated on Figure 4.

b	word length for image data (bits/pixel)
B	digital value of an image pixel, $0 \leq B \leq 2^b-1$
B_0	central pixel value $= 2^{b-1}$
d	word length for LUT data (and DAC)
D	digital value of an LUT entry, $0 \leq D \leq 2^d-1$
D_0	central LUT entry $= 2^{d-1}$
L_{max}	maximum luminance (when $D = 2^{d-1}$ in one channel)
L_{min}	minimum luminance (when $D = 0$ in one channel)
L_0	mean luminance $= (L_{max} + L_{min})/2$
$G(D)$	gamma function

Inverse Gamma Function

Consider Figure 4. The horizontal axis indicates the LUT value input to the DAC, ranging from 0 to 2^d-1, and the ordinate shows luminance values recorded from the screen. The gamma function $G(D)$, the nonlinear relationship between digital values and luminance, is pictured. It ranges from L_{min} to L_{max}, with a midpoint of L_0. Since there are only 2^d possible inputs D, the gamma function $G(D)$ is a table of 2^d luminance values corresponding to particular digital inputs.

We wish to arrange a linear relation between FB values and pixel luminance of the form

$$E(B) = L_0[1+A(B-B_0)/B_0]. \quad (1)$$

The parameter A is the *available contrast*, that is, the maximum contrast that can be generated on a background of L_0. Since $E(0) \geq L_{min}$,

$$A \leq A_{max} = (L_{max}-L_{min})/(L_{max}+L_{min}). \quad (2)$$

If $L_{min} = 0$, then $A_{max} = 1$. In other words, if the display can generate a black level (0 luminance), then a contrast of 100% can be produced. This linear relation says that B_0 will be mapped to L_0, regardless of the value of A. It represents a background level upon which both positive and negative signals can be superimposed. If the image is represented as a contrast signal, then pixels with zero contrast should have a digital value of B_0, and positive and negative contrasts should be increments and decrements from this value.

Gamma Correction Table

The first step in arranging the linear relation above is to construct a *gamma correction table*, $C(D)$. Consider the following linear mapping between LUT entries and luminance:

$$L(D) = L_0[1+A_{max}(D-D_0)/D_0]. \quad (3)$$

Consider how the mapping $L(D)$, as pictured in Figure 4, might be accomplished for one example input value D. Drawing a straight line up from D to the linear relation $L(D)$, and then straight over to the ordinate, we discover the desired output luminance $L(D)$. Now we need to discover what digital input value, $C(D)$, will give the desired luminance output. This value can be read off of the measured gamma function $G(D)$ by reading backward from the output value $L(D)$ to the required input $C(D)$, as indicated by the dashed lines in Figure 4. In going backward from a luminance value to a digital input, we are in effect inverting the function $G(D)$. We define the function $G^{-1}(L)$ as returning the digital value D for which $G(D)$ most nearly equals L. Then $G^{-1}(G(D)) \approx D$. We call this the *inverse gamma function*. The next step is to create a correction table consisting of inverse gamma function values for each desired luminance value $L(D)$ as defined above. The correction table is created according to the rule

$$C(D) = G^{-1}(L(D)). \quad (4)$$

In concrete terms, the algorithm to create $C(D)$ is as follows:

```
measure G(D)
for each D from 0 to 2^d-1 {
  compute L(D) from Equation 3
  find that D for which the value of G(D) is nearest
  to L(D)
  set C(D) equal to the selected D
}
```

When the table $G(D)$ is measured, it will have some noise and may not be strictly monotonic. This may be ameliorated by smoothing $G(D)$ before using it to compute the correction table, by fitting a polynomial or other smooth function or by local averaging of the table.

The sequence of values $C(D)$ should now generate outputs varying linearly between L_{min} to L_{max}. This can be confirmed by successively displaying each of the values in the correction table and recording the resulting lu-

minance. The result of this step is shown by the curve labeled $L(D)$ in Figure 4.

Usable Levels

An important measure of the table $C(D)$ is the number of *usable levels*. If the luminance were related linearly to the value D, then there would be 2^d distinct usable levels of luminance. But when the relation is nonlinear, examination of the table $C(D)$, which contains all the values of D that will ever be used, shows that some levels are used several times, whereas others are skipped over. Thus, there will be fewer usable levels in $C(D)$ than there are values of D. In the example pictured in Figure 4, there are 700 usable levels. The higher this number, the more nearly linear the display and the lower the quantization error. Judicious adjustment of the monitor contrast and gain controls can be used to optimize this measure. We generally reject a calibration if it yields fewer than 680 usable levels.

Computing the LUT

The next step is to construct an LUT $T(B)$ from the correction table. This is done using the following algorithm:

```
select a value of A ≤ Amax
for each B from 0 to 2^b − 1 {
  set J = D0[1 + (A/Amax)(B−B0)/B0]
  set T(B) = C(J)
}
```

The table $T(B)$ is then loaded into the hardware LUT. The pixel luminance is then as specified in Equation 2.

TWO-CHANNEL CONTRAST CONTROL

The word length of the DAC, and thus of the LUT entries, determines whether digital contrast control will be effective. In the Adage RDS-3000 with LUVO 30 option, the DAC word length (d) is 10 bits. If, instead, the output DACs are only 8 bits, then digital attenuation may result in large quantization errors. For example, if the system with 8-bit DACs is set up to permit contrasts of 100% ($A_{max} = 1$), then a signal presented at the minimum contrast visible to a human (about 0.5%) will have only about one or two levels. Even with 10-bit DACs, fewer than eight levels would remain. In such cases, the signal is not usually what the experimenter intended. Various methods are possible to solve this problem. The method we describe here uses two channels of a color FB (e.g., red and blue). The video outputs of the two channels are passively mixed by means of a resistor network (Figure 5), such that one channel (blue) is attenuated by a factor R. In mode 1, the image is routed via the XBAR to the red channel, the appropriate $T(B)$ created as described above, is loaded into the red LUT, and the blue LUT is loaded with zeros. Thus, mode 1 behaves as though a single channel were being used. Mode 1 is used to produce high-contrast images, from a contrast of RA_{max} to a contrast of A_{max}. In mode 2, the red LUT is loaded

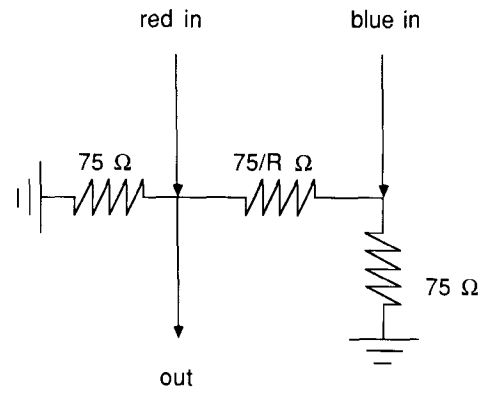

Figure 5. Resistor network used to passively mix red and blue video signals to achieve the two-channel method of contrast control.

with a constant, red_0, and an appropriate $T(B)$ is loaded into the blue LUT. The image is then directed via the XBAR to the blue channel. In this mode, the red channel produces a steady background, and the blue channel produces a small signal superimposed on the background. This mode is used whenever a contrast less than RA_{max} is required.

To determine the proper value of red_0, use the following procedure:

```
select a value of red0 between 0 and 2^d − 1
load red0 into the red LUT
load the value 2^d − 1 into the blue LUT
record the screen luminance as blue_max
load the value 0 into the blue LUT
record the screen luminance as blue_min
set blue0 = (blue_max + blue_min)/2
if blue0 < L0 then increase red0 and repeat the above
if blue0 > L0 then decrease red0 and repeat the above
```

The procedure is repeated until no further adjustments in red_0 bring $blue_0$ any closer to L_0. Then the blue channel is calibrated, and a correction table created, as in the single-channel method, except that red_0 is kept in the red LUT. In practice, the attenuated channel is usually linear, since it is producing only a small range of signals, and this may be taken advantage of to hasten the calibration procedure. The actual available contrast of the display in mode 2 will be RA; thus, it is appropriate for small contrasts, from 0 to RA_{max}.

What basis is there for selection of the value of the attenuation R in the second channel? The value of R will affect the worst-case quantization in both modes 1 and 2. At the transition between modes 1 and 2, the contrast will be RA, and in mode 1 the number of levels, N_1, will be RM_1, where M_1 is the number of usable levels in mode 1. At the minimum desired contrast C_{min} (which will be determined by the minimum contrast threshold for human observers, i.e., about 0.005) the number of levels N_2 in mode 2 will be M_2C_{min}/R, where M_2 is the maximum number of usable levels in mode 2. Table 1 shows some representative values. As R increases (and attenua-

Table 1
Minimum Usable Levels in Modes 1 and 2 as a Function of Attenuation R

R	N_1	N_2
0.01	7	512
0.05	35	102
0.1	70	51
0.15	105	34
0.2	140	26

Note—The values above assume that $M_1 = 700$, $M_2 = 1{,}024$, $C_{min} = 0.005$.

tion decreases), N_1 increases but N_2 decreases. At first glance, it might appear sensible to equate these two worst-case numbers of levels. However, the quantization error will in general be more visible at the transition between modes 1 and 2 than at C_{min}. Therefore, it makes sense to devote more levels to R than to C_{min}, that is, to make $N_1 > N_2$. This view may be tempered, however, by the user's view of whether they are likely to be more often in mode 1 or mode 2. We currently use a value R=0.1.

REAL-TIME CONTROL OF CONTRAST

It is often desirable to display an image with a contrast that varies over time. Let us call the variation the *contrast timecourse*. This may be represented as a list of floating point numbers ranging between −1 and +1, where each number represents the contrast in a single frame of the display. We have already described how to create an LUT for a particular contrast. Recall that it requires only a linear transformation of 2^b numbers, since the correction table, $T(D)$, is constructed once after each calibration. To arrange a particular timecourse then, we must take each number in turn from the list, use it to create an appropriate LUT, load the new LUT in an unused display LUT, and then, during the blanking interval preceding the next frame, switch to the new table. The Adage RDS-3000 has two complete LUTs (not including cursor LUTs), which allow us to compute and load the unused LUT during each frame (16.6 msec). If only one LUT is available, it must be loaded during the blanking interval at the end of each frame, which is usually only about 650 μsec long. If the LUT has 256 entries, a transfer rate of at least one element every 2.5 μsec is required. This does not include the time to compute the LUT, which must be done in advance.

These techniques have been implemented in the routine

cmvary(t_list, contrast, c_table1, c_table2, frames)

where **t_list** is the contrast timecourse, **contrast** is a global contrast multiplier, **c_table1** and **c_table2** are the correction tables for the red and blue channels, and **frames** is the number of frames to display.

IMAGE DISPLAY

The actual display of an image requires several steps. First, the viewport must be set to the desired size and location. In many experiments, this will be done once at the start of the session. For this we use the routine

viewport(vwc_x, vwc_y, vw_width, vw_height)

where **vwc_x** and **vwc_y** are the locations of the *center* of the viewport, and **vw_width** and **vw_height** define the size. We have found it more convenient to specify the center, rather than the coordinates of the upper left corner.

Next the XBAR must be set to select the plane in which the image lies. Then the window must be moved to the coordinates of the image. These two steps are incorporated in the routine

pos_image(image_n, x, y)

where **image_n** is the number of the selected image (or staging area; see below). The window will be centered on the coordinate **x,y**. Finally the LUT must be set to the desired sequence of values, for example, by use of the **cmvary()** routine described above. For some experiments, it may be advisable to extinguish the cursor (fixation point) before presentation of the image and to reactivate afterward.

IMAGE MOTION

Framebuffers typically provide control over the horizontal and vertical starting point of the displayed data in the FB. These are usually called horizontal and vertical window values. By altering these values in successive frames, the display can be made to move. At the most general level, the user may provide a time list of horizontal and vertical coordinates along which the image is to travel. More often, a constant velocity is desired. Note that speed in each dimension, expressed in pixels/frame, must be a rational number. Furthermore, it is usually inadvisable to let the denominator be too large, since this will result in jerky motion. These constraints on the one-dimensional speeds impose constraints on the two-dimensional velocity. The speed in one dimension is equal to

$$\text{speed(deg/sec)} = \text{pix\_frame} * \text{frame\_sec/pix\_deg} \quad (5)$$

where **pix_frame** is the displacement in pixels/frame, **frame_sec** is the frame rate in frames/sec, and **pix_deg** is the display resolution in pixels/degree. For a typical display with 20 pixels/cm at a viewing distance of 114 cm, **pix_deg** = 40, and if **frame_sec** = 60 Hz and no more than three frames elapse between each shift (**pix_frame** ≥ 1/3), then the lowest speed that can be obtained is 0.5°/sec. This is a sampling problem, and can to some extent be ameliorated by filtering the space-time image prior to sampling. This means, of course, creating a different image for each frame of the display (see Movies).

Since motion is produced by scrolling the display window through the framebuffer memory, care must be taken that only the desired imagery comes into view during the

duration of the display. On many systems, the memory wraps around, so that if an address beyond the range of permissible pixel addresses is called for, the pixel address modulo the width (or height) of the memory is used. Thus, for example, if a periodic pattern with period that integrally divides 1,024 were scrolled within a display of that size, it would continue to move without disruption indefinitely. Alternatively, the image to be moved may be padded by an appropriate background as large as the image in each dimension. The resulting padded image will have four times the number of pixels as will the original. Then the actual offset will be the desired offset in each dimension modulo the size of the original image.

MOVIES

A sequence of frames may be displayed as a movie by successively moving the window to each frame. It is worth noting that substantial movies (by psychophysical standards) can be effected in this way. For example, if there are 4 FB planes of $1,024 \times 1,024$ pixels, and if each frame is 128×128, then a total duration of 256 frames, or about 4 sec, may be displayed. If fewer than 8 bits are allocated to each frame, then the XBAR may be used to select individual bit planes or sets of bit planes. A binary movie with dimensions of 128×128 could run for 2,048 frames, or about 34 sec.

STAGING AREAS

Consider an experiment in which images, drawn from a set of N, are viewed side by side in pairs. If any possible pairing must be available, and if the ordering (left vs. right) is important, then there are $N(N-1)$ pairs. To store this many images in the FB may not be possible. It is much simpler to store only the N possible components, and then provide a means of assembling a pair at the time it is needed.

We do this by creating a **staging area**, a reserved portion of the FB, large enough to hold the desired multicomponent display. Prior to display, a copy of each component is moved to the desired location within the staging area. This method, which is useful with a wide variety of multicomponent displays (e.g., apparent motion between various target pairs, two or more spatial alternative forced-choice experiments, simultaneous comparisons, etc.), requires two routines. The first is:

image_n = stage_fb(stg_width, stg_height)

This behaves identically to the routine **load_fb()** introduced previously, except that it does not transfer an image into the FB. The arguments specify the size of the FB region to be reserved. Thus, the staging area is simply another image, except that its contents are not defined by reading into it an image from the file system. The virtue of this technique is that the staging area can now be

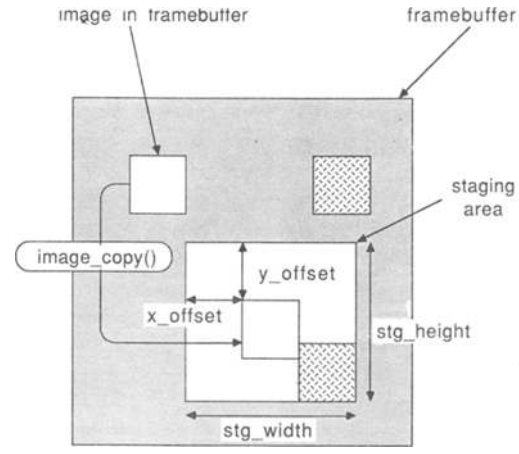

Figure 6. Use of a staging area. An image is copied from one region of the framebuffer into the staging area region. Although the illustration shows the image and the staging area in the same plane, this need not be so.

treated like any other image; for example, it can be referred to in a call to the display routines.

The second routine is:

image_copy(source_n, destination_n, x_offset, y_offset)

This routine copies an image, as specified by the image number **source_n**, to the staging area, as specified by the image number **destination_n**, but places the upper left corner of the source image at the coordinate **(x_offset, y_offset)** within the staging area. The image is clipped to the limits of the staging area. This routine executes on the high-speed graphics processor that is an optional part of the Adage RDS-3000. The use of staging areas is illustrated in Figure 6.

COLOR DISPLAYS

For many purposes, color displays can be arranged by repeating the above procedures three times, once for each of the red, green, and blue displays. The two-channel method of contrast control cannot be easily used with color displays. It is also possible to arrange more elaborate display routines, in which, for example, the three memory planes are used to accommodate imagery encoded in some linear transformation of the red, green, and blue signals, for example, as red/green, blue/yellow, and black/white signals.

DESIRABLE FEATURES

Many of the techniques described here require hardware features that not all systems have. Although many of these may not be needed, we thought it worthwhile to list those features that seem particularly useful in light of our experience.

1. **10-bit DACs**: important for digital contrast control,

particularly if the two-channel method cannot be used (e.g., when all three channels are needed for color).

2 **Window**: essential for motion and selection of images within the FB.

3. **Viewport**: essential if stored images are smaller than the display size.

4. **Multiple LUTs**: essential for real-time contrast control, unless a new LUT can be loaded during the vertical retrace interval (see Real-Time Control Contrast of above).

5. **Large FB**: useful for storage of multiple high-resolution images and for movies.

6. **Local processor**: may be essential for high-speed image copy operation. A local processor is also valuable when the host uses a non-real-time operating system such as UNIX. Then the real-time operations can be placed under the control of the local processor.

REFERENCES

ADAGE, INC.(1982). *RDS 3000 User's Guide*. Billerica, MA: Author.

CATMULL, E. (1979). A tutorial on compensation tables. *Computer Graphics*, **13**, 1-7.

THE CONRAC CORPORATION (1985). *Raster graphics handbook* (2nd Ed.). New York: Van Nostrand.

LANDY, M. S., COHEN, Y., & SPERLING, G. (1984). HIPS: A Unix-based image processing system. *Computer Vision, Graphics, & Image Processing*, **25**, 331-347.