Techniques for Frame Buffer Animation

by

Stephen A. MacKay

An essay
presented to the University of Waterloo
in partial fulfillment of the
requirements for the degree of
Master of Mathematics
in Computer Science

Waterloo, Ontario, 1982

Abstract

Frame buffer animation creates the illusion of limited real-time animation with static pictures stored in the image memory of a digital frame buffer. Hardware features in the output video chain such as the video controller, the crossbar switch, and the colour tables allow techniques like zoom-pan-scroll animation, crossbar animation, three forms of colour table animation, and various combinations. These techniques, the hardware required to implement them, their application to low-cost systems such as Telidon, and future hardware and software enhancements are discussed.

Introduction

Animation is a technique for creating the illusion of motion with a series of static "images". The motivation for using animation is suggested by a secondary definition for "animation": imparting interest or zest and the enlivening of images [8]. Throughout this essay we will think of animation not as an alternative to live action filming, but as a means for improving an otherwise static presentation.

Traditional forms of animation such as cel animation and three dimensional model animation are highly labour intensive tasks and thus very expensive. The use of computers to reduce the effort in the creation of in-between frames, in inking cels, and in the manipulation and simulation of three dimensional models is the subject of much current research. Various surveys, [16], [9] and [5], provide a more complete description of traditional and computer animation. We will restrict attention to the specific goal of using digital frame buffers to achieve real-time playback of an animation sequence.

Ron Baecker has already championed the proposition that dynamic graphics is no longer the sole domain of calligraphic systems, but he observed that "current frame buffers are unsuited to dynamic imagery" because of the requirement, inherent in frame buffer architecture, that changes to the image require explicit modification of pixels in memory [3, p. 54]. The amount of computation required for major changes, coupled with the need to synchronize those changes with video generation, have precluded full real-time animation.

The term *frame buffer animation* refers to a class of animation techniques that make use of digital frame buffers to enhance static images through the illusion of real-time motion. The trick is that the image stored in the memory of the frame buffer remains entirely unchanged once created. Dynamics are added by modifying the pattern in which pixels are read from memory and the way in which their bits are interpreted.

Frame buffer animation provides some degree of interactive control over the playback of an animated sequence because an intermediate medium, such as film or video tape, is not necessary. As well, frame buffer animation is cheaper than performing "real" animation, not because the images are less expensive to create (they are not), but because time consuming post-processing is not required.

Drawbacks to this form of animation include the high cost of initial image creation, the possibility of reduced image quality and the small number of frames available. Moreover, as hardware costs continue to drop, many of the features described in the following chapters will become commonplace in graphics workstations and terminals. As a consequence frame buffer animation will become a valuable tool in the field of "low-cost" graphics.

Development Environment

The techniques described in this essay were developed using a PDP 11/45 mini-computer running the Unix operating system, and an Ikonas RDS-3000 frame buffer. The basic principles we discuss can be applied to other frame buffer configurations with similar features. Because specific examples in later sections will be drawn from our particular environment we will examine it in some detail before surveying our animation techniques. Much of this section is based on information collected from the documentation provided by Ikonas [13, 14, 15].

The Frame Buffer

The frame buffer system in use in the Computer Graphics Laboratory of the University of Waterloo is an Ikonas RDS-3000 consisting of a 512x512x32 bit image memory with a video controller, a crossbar switch, colour lookup tables, direct memory access interface to the host computer and a bit-slice microprocessor with scratch pad memory for communication with the hosts. A memory map and addressing layout is shown in Figure 1. These components and their functions will be discussed in detail in this section.

Image Memory

The image or pixel memory of the Ikonas is configured as an array of 512x512 pixels each thirty-two bits deep. This dynamic RAM can be alternatively configured as 1024x1024x8 or as 256K of 32-bit words under software control. Access to memory is through a 12 Mbyte/sec video input port, a 20 Mbyte/sec video output port, or a system port connected to the Ikonas bus having 200 nsec access time and 400 nsec cycle time. Pixels are read from image memory through the video output port and passed in scan line order through the video chain to the colour monitor for display. Input to the image memory is controlled by a write mask that allows any subset of the thirty-two bit-planes to be selected for modification during a write cycle.

The Video Chain

The video chain is a set of hardware modules that process the data from image memory at

37700 1777	reserved for multiple Multi-Peripheral Controller systems	
34100 0000		
34077 1777	upper 16 bits unused	MPC space
34000 0000		
33700 1777	reserved	
30300 0000		
30200 1777	crossbar switch	
30200 0000		
30100 1777	video input	
30100 0000		
30000 1777	video control module	
30000 0000		
27777 1777	reserved	
20700 0000		
20677 1777	character generator	
20600 0000		
20477 1777	matrix multiplier	
20400 0000		
20377 1777	colour tables	
20300 0000		
20277 1777	scratch pad for microprocessor and matrix multiplier	
20200 0000		
20177 1777	reserved	
20100 0000		
20077 1777	microcode	
20000 0000		
17777 1777	image memory	
00000 0000		

```
yyyyy xxxx  - 24 bit octal address
                yyyyy - upper 14 bits
                 xxxx - lower 10 bits
```

Figure 1   Ikonas Memory Map

video rates. It consists of three primary parts: the *frame buffer controller* (video control module) determines viewing, timing and cursor parameters; the *video crossbar switch* allows for the rearrangement of bits within pixels; and the *colour lookup tables* translate 24-bit indices into 10-bit red, green and blue (RGB) intensities. In addition to their more traditional uses, each module can be used to effect a different type of animation.

Video control module. The frame buffer controller contains a set of read-only registers that determine the format and timing in which pixel data is read from image memory. A schematic representation of the video control module is given in Figure 2. The display characteristics can be altered simply by changing the value in the appropriate register.

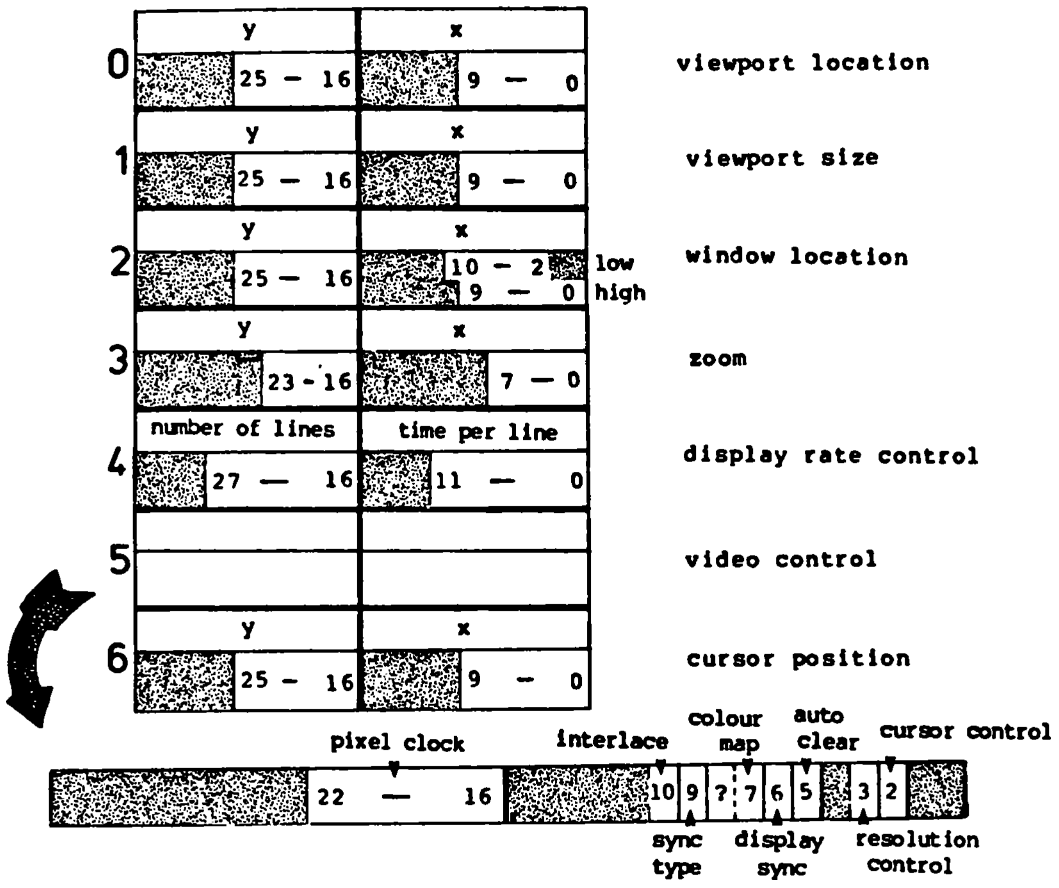

Figure 2 Video Control Registers

Register 0 *viewport location* - specifies the (x,y) coordinates of a position on the surface of the display monitor that will correspond to the upper left-hand corner of the image. Note that the low order two bits of both the x and y values are ignored so that the location is always given in four pixel increments.

Register 1	*viewport size* - specifies the size (number of pixels) to be displayed both horizontally and vertically. Oversize viewports wrap around. Again the two low order bits are ignored.
Register 2	*window location* - gives the (x,y) address of the first pixel value to be displayed in the upper left-hand corner of the viewport. The low order two bits of the x value are ignored, but the x value is shifted right two positions before use so single pixel panning is possible.
Register 3	*zoom register* - provides a replication count that specifies how many times each memory pixel is to be displayed, effectively providing independent zoom factors for both the horizontal and vertical directions.
Register 4	*display rate control* - the low order bits determine the horizontal scan frequency, measured as the number of clock cycles per line. The high order bits determine the vertical scan frequency, given as one less than the number of lines to be displayed (including vertical drive pulse and blanking interval). If the display is to be interlaced, this value must be even, which of course specifies an odd number of scan lines.
Register 5	*video control*

Bit 2	*cursor control* - hardware cursor on if set and off if clear.
Bit 3	*resolution control* - specifies low resolution (512x512) if 0 and high resolution (1024x1024) if 1.
Bit 5	*auto-clear control* - denotes normal operation if 0 and if 1 clears image memory after displaying each frame.
Bit 6	*display sync select* - uses external sync if 1 and internal sync generated automatically by the frame buffer controller if 0.
Bit 7	*colour map page* - determines which colour map will be used. (Bit 8 may be used as well, depending on which documentation is to be

believed [14].)

Bit 9 *sync timing* - if 0 then RS-170 (NTSC) sync is to be used and if 1 RS-343 (for high resolution monitors)

Bit 10 *sync timing* - interlaced if 0 and repeat field if 1.

Bits 16-22 *pixel clock* - nanoseconds per single pixel for high resolution and nsec per half pixel for low resolution. The aspect ratio of the display can be changed using this value. (It may be nsec per single pixel for low resolution and nsec per two pixels for high resolution again depending on which documentation is to be believed [14].)

Register 6 *cursor location* - specifies the (x,y) coordinates of the cursor position in the image memory.

Register 7 *unused* - formerly specified the colour of the hardware cross-hair cursor but is not used with the programable hardware cursor.

Video crossbar switch. The Ikonas system has a full crossbar switch that will route any of the thirty-two bits from pixel memory to any of the twenty-four input lines of the lookup tables. In addition to re-routing the inputs, the crossbar switch is capable of ignoring input and passing a zero bit through to any output. A schematic representation of the registers is shown in Figure 3, including those for colour map selection that also pass through the crossbar switch. The crossbar switch is set by loading each register, one for each output bit, with a number between 0 and 31, denoting which input bit it will pass, or a 63 to indicate ignored input.

With the crossbar switch in the video chain all the thirty-two bit-planes can be routed to any of the lookup table inputs. One common configuration is to use the crossbar switch to successively choose eight bit-planes, allowing a double buffered display. Figure 4 illustrates this double buffering for a z-buffer algorithm in which the high-order sixteen bit-planes are dedicated to storing depth information and the low-order bit-planes constitute two intensity buffers.

Colour lookup table. A colour map consists of a set of high speed registers that perform

	Input bit selected	
Pixel data 0	0	Red 0
Pixel data 1	1	Red 1
Pixel data 2	2	Red 2
Pixel data 3	3	Red 3
Pixel data 4	4	Red 4
Pixel data 5	5	Red 5
Pixel data 6	6	Red 6
Pixel data 7	7	Red 7
Pixel data 8	8	Green 0
Pixel data 9	9	Green 1
Pixel data 10	10	Green 2
Pixel data 11	11	Green 3
Pixel data 12	12	Green 4
Pixel data 13	13	Green 5
Pixel data 14	14	Green 6
Pixel data 15	15	Green 7
Pixel data 16	16	Blue 0
Pixel data 17	17	Blue 1
Pixel data 18	18	Blue 2
Pixel data 19	19	Blue 3
Pixel data 20	20	Blue 4
Pixel data 21	21	Blue 5
Pixel data 22	22	Blue 6
Pixel data 23	23	Blue 7
Pixel data 24	63	Overlay 0
Pixel data 25	63	Overlay 1
Pixel data 26	63	Overlay 2
Pixel data 27	63	Overlay 3
Pixel data 28	63	Overlay 4
Pixel data 29	63	Overlay 5
Pixel data 30	63	Overlay 6
Pixel data 31	63	Overlay 7
In-cursor bit 32	32	Cursor
Colour map page 33	33	Colour map page
(2nd colour map page)* 34	X	-nothing-

* this bit may or may not be present

Figure 3 Crossbar Switch Registers (Usual Setting)

a mapping from pixel values stored in image memory to the digital colour levels sent to the D/A converters. Originally, lookup tables were added to frame buffers for gamma correction (the lookup table values are modified to compensate for non-linearities in colour monitors, film and human perception) and to provide pseudo and false colour for image processing.

Sloan and Brown [26] provide a very good survey of the innovative uses to which colour tables subsequently have been put. They suggest that the colour table could be used to overlay simpled coloured images on detailed black and white backgrounds to aid viewing; for multi-buffering several images to achieve real animation; for colour system transformations; or to

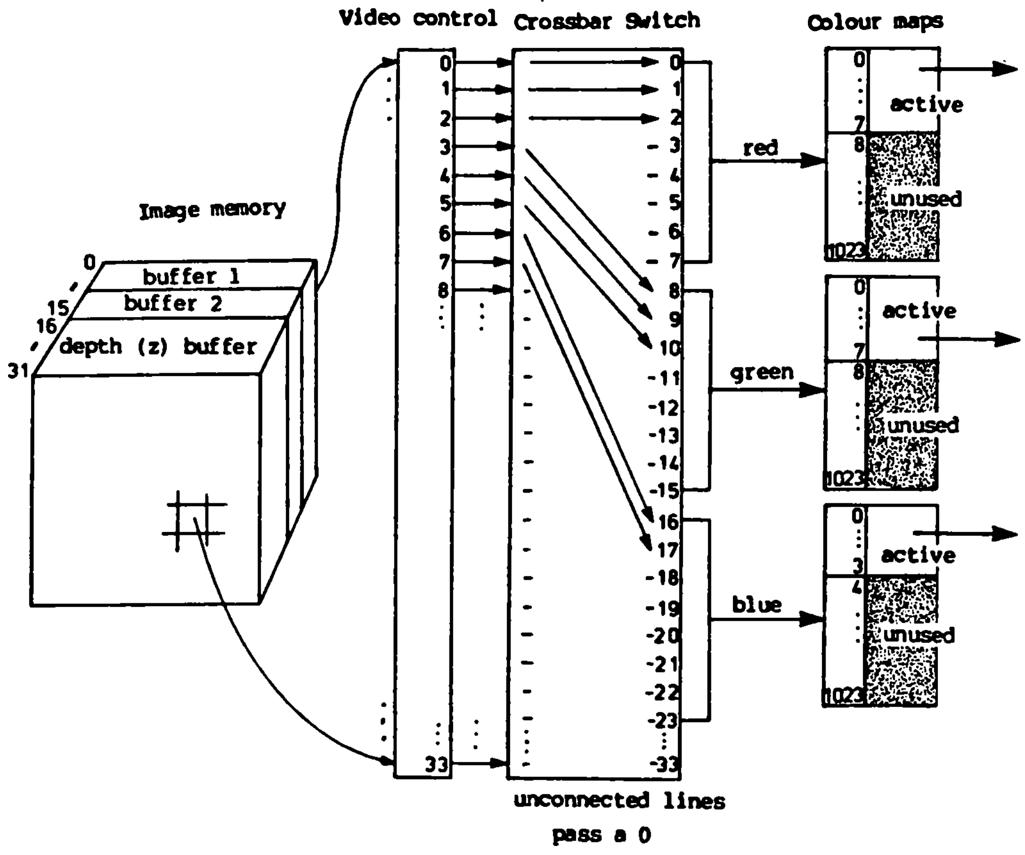

Figure 4 Double buffer/Z-buffer Configuration

increase the dynamics of a series of images that would normally take a prohibitively large amount of computation to generate, such as storing surface normals in pixel memory and loading the map with precomputed shading models [26]. Further discussion of this last use is provided by Bass [2].

The pixel value comes out of the crossbar switch as three 8-bit colour numbers, one each for red, green and blue. These numbers then act as indices into the red, green and blue colour maps from which the 10-bit digital intensities are read. This is shown schematically in Figure 5. Random access to the lookup table registers requires that they be extremely fast. In a 1000-line (30 Hz) display the colour tables are accessed once every thirty nanoseconds. For this reason colour tables are often achieved using ECL memory.

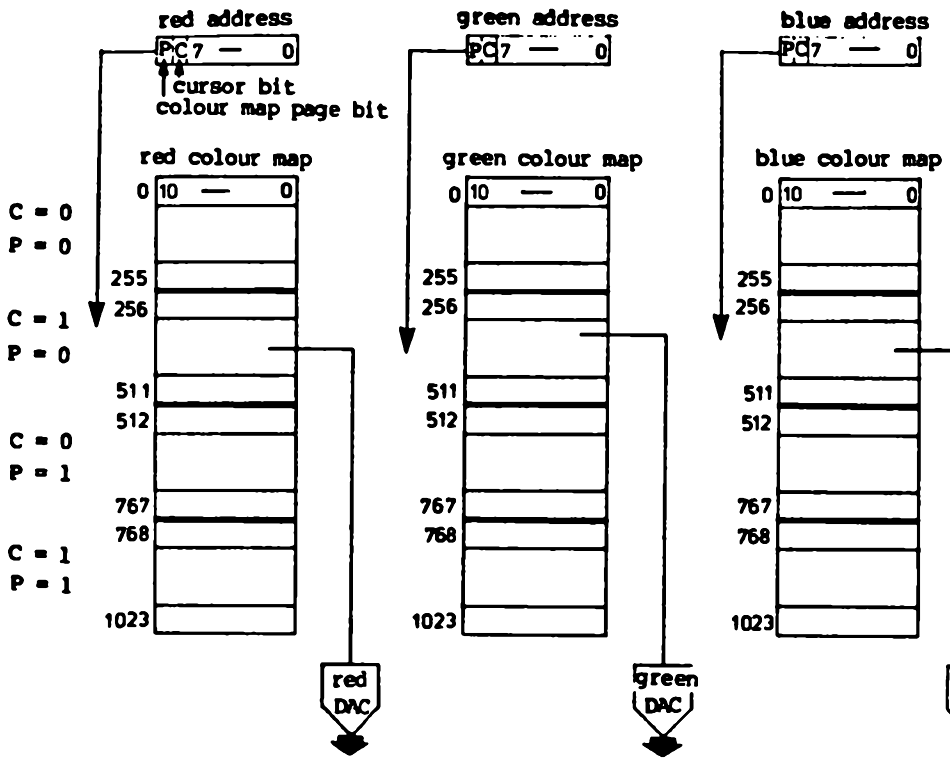

Figure 5 Ikonas Colour Maps

Input Hardware

For frame buffer animation it is necessary that all images of the sequence be stored in pixel memory at the same time. Because each image is created separately, some means must be provided to scan convert the data into specific parts of memory. The Ikonas has eight *write mask registers* (the z masks) that control which bits within a pixel change during a write to the frame buffer. This substantially cuts down on the overhead of producing pictures for frame buffer animation. Normally either the host or the frame buffer's microprocessor would have to perform a read-modify-write operation to insert only those bit-planes involved with a particular subimage. Often this is quite expensive when performed on a host, and even on special purpose bit-slice microprocessors it can still be significantly slower than simple writes. Setting the appropriate write mask to select only those bits being written allows the host or microprocessor to proceed as if the other bit-planes were not present. The reasons for wanting to do this will become clearer when crossbar and colour table animation are discussed.

The Ikonas frame buffer provides eight write masks with the intention that each host attached to the system (including the bit-slice microprocessor) will use a different mask. The mask is therefore not a critical resource and each processor can assume that its own write mask will be untouched by the others. This is not necessarily true because bits 12-14 of the Ikonas command register determine a host identification, which is essentially a write mask select. Thus under software control any host can access memory through or change any write mask; it is up to the programmers to establish mutual exclusion. The usual solution is to adopt the convention of permanently allocating a write mask to each processor. The auto-clear feature always uses write mask 0, so that mask should be used with caution.

Graphics Microprocessor

The microprocessor on the Ikonas 3000 system consists of a 32-bit custom bit-slice ALU with up to 64K of 64-bit microcode. The current configuration in the Computer Graphics Lab has only 4K of store with 8K of 32-bit scratch pad memory available. The scratch pad can be used by the microprocessor alone or shared with the host for communication. The power of the microprocessor derives from the fact that it has full access to the Ikonas bus, allowing it to manipulate registers in the video chain at very high speeds without relying on the timeshared host. The microprocessor is especially useful for frame buffer animation applications. The inner loops of the animation software that frequently access the Ikonas registers, when coded in the microprocessor, provide the exact timing necessary for visual continuity. Code can be written for the microprocessor using Microcode C, developed in the Computer Graphics Laboratory by Preston Gurd [12].

Other Hardware

Hardware attachments, other than those in use in the Computer Graphics Laboratory, are available for the Ikonas frame buffer. The *video input digitizer* allows real-time input of video images into memory from a camera or videotape player. The *matrix multiplier* is used for performing fast two and three dimensional transformations of items to be displayed. The *character generator* provides fast display of text in selectable sizes, locations and colours. Finally, the *multi-peripheral controller* is a 68000 based 16-bit microprocessor that includes up to 512K bytes of memory and a hard disk unit. It is scheduled to run Unix, allowing the Ikonas to be an extremely powerful stand alone graphics workstation.

The advantage of frame buffer animation lies in the fact that absolutely no changes are made to the contents of frame buffer memory during the animation cycle. Instead, images are created and loaded into memory once, beforehand, and after that every pixel value remains completely unchanged throughout the animation. It is the way in which the video chain interprets the data, the order in which it is read, the permutations it undergoes, and the transformations it makes that provide the illusion of motion.

Each of the colour maps, the crossbar switch, and viewport, window and replication (zoom) registers of the video control module are useful for effecting a different type of animation. These can be used singly or in combination.

Colour Table Animation

Dick Shoup is generally credited with the insight for using a colour lookup table to provide a limited form of frame buffer animation [23]. He makes the following observation in the introductory section of his tutorial paper. It applies not only to colour table techniques, but to all the frame buffer animation techniques we will examine.

> Fortunately, even motion which is graphically very simple can produce a vastly more effective visual communication than a still image. In short, *a little animation goes a long way.* If we are willing to forego full Disney-style animation and strongly limit the content and dynamics of our images, then much simpler, highly interactive solutions can be found [23, p. 9].

Shoup's taxonomy of colour table animation includes *colour cycling, alternate colour animation* and *bit-plane extraction.* (Shoup does not use the term bit-plane extraction, but includes it as an extension of alternate colour animation; we consider it separately because it is an important concept worth singling out.) Each technique treats values stored within pixels differently, the interpretation depending on the particular pattern of intensities stored within the lookup tables.

Colour cycling. Probably the simplest effect is achieved by rotating the entries in the lookup table, giving the illusion of motion as colours appear to "flow" across the screen. Because the entire image remains visible at all times, the type of sequence that can be represented must be kept simple. Flowing liquids such as the waterfall in Figure 6, moving arrows, or abstract designs are typical examples. Variations that cycle only some of the colours, or that independently cycle various subsets of the entries, often at different rates and in opposite directions, are also possible. Due to the cyclic nature of this technique, the motion can easily be made to last as long as desired.

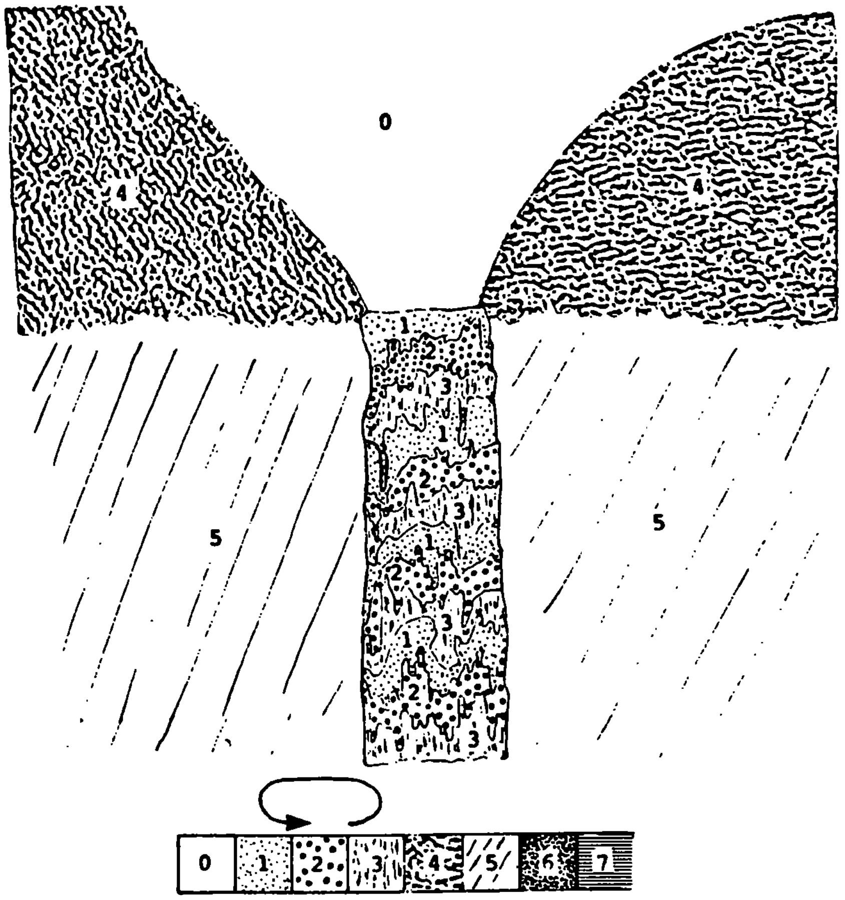

Figure 6 Colour Cycling

Alternate colour. A more powerful method, known as alternate colour animation, paints more than one image into the frame buffer memory using different pixel values for each one. Setting the lookup table entries to correspond to the background colour for all but one set of values allows a single image to be viewed. Each image in a sequence is made to appear in turn by changing the subset of visible colours. Figure 7 shows an example where the cannon and the castle are visible throughout the sequence and various cannonballs appear in turn to provide the illusion of motion.

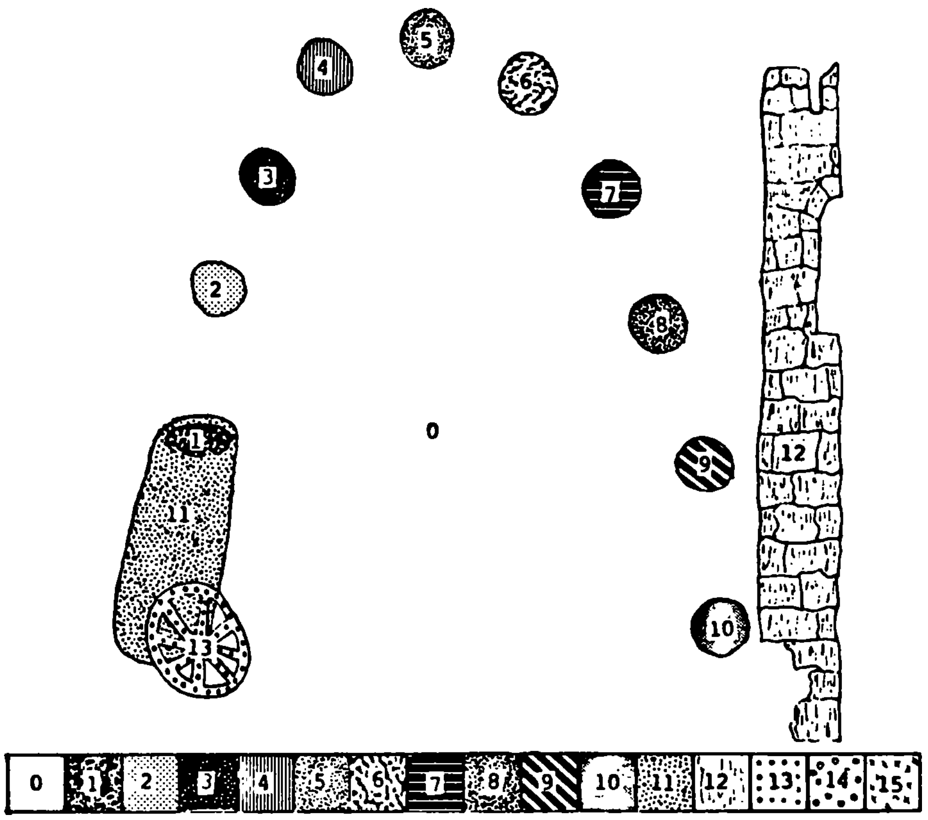

Figure 7 Alternate Colour Animation

Another use of alternate colour techniques, employed by Baldev Singh in his Benesh dance notation editor, is for the rapid display of menus to achieve greater interactivity [25]. Each collection of menu items is given a unique colour number and valid selections at any given time are assigned visible colours and menu items that may not be chosen are assigned background values. Here the object is not to provide rapid spatial movement but rather rapid disclosure of information.

There are some problems associated with alternate colour animation. Because each image has a unique pixel value associated with it, each frame in the animation sequence must be disjoint; there can be no overlap among the images. The background must be a solid colour behind each position of the moving object, further limiting the amount of detail that can be represented.

Bit-plane extraction. This extension to alternate colour animation helps alleviate some of the problems. Bit-plane extraction assigns to each overlap region a unique colour. A simple way to achieve this is to partition the frame buffer into bit-planes (a pixel in memory is now being thought of as a string of bits, one in each plane) and assigning to each image a separate set of bit-planes. Loading the lookup tables to "ignore" all but the bit-planes corresponding to a particular image allows images to be displayed separately. Figure 8 shows how the colour map would be set in a 3-bit per pixel frame buffer to access each plane. More than one image can be selected for simultaneous display. This technique can be used to mix images, overlay one image with another using a priority scheme, or even to add "transparency" by allowing one image to show "through" another. Later we will see that a crossbar switch provides an even more powerful tool for implementing some of these bit-plane extraction techniques.

Zoom-Pan-Scroll Animation

The simplest zoom-pan-scroll animation exists when the pixel memory is logically divided into different regions, each containing a separate, lower-resolution picture. Figure 9 illustrates the case in which four 256x256 images are stored within a 512x512 frame buffer. The zoom, viewport and window registers in the video control module are set so that only the first subimage is visible, filling the entire display screen. Each successive frame in the sequence can be shown by setting the x window location register ("pan") and the y window location register ("scroll") to the upper left-hand pixel of each subimage.

One advantage of this approach over lookup table animation is that each pixel maintains a full 32-bit colour value, in the case of the Ikonas, instead of sacrificing palette size for enhanced dynamics. This is clearly a tradeoff between spatial resolution and intensity resolution.

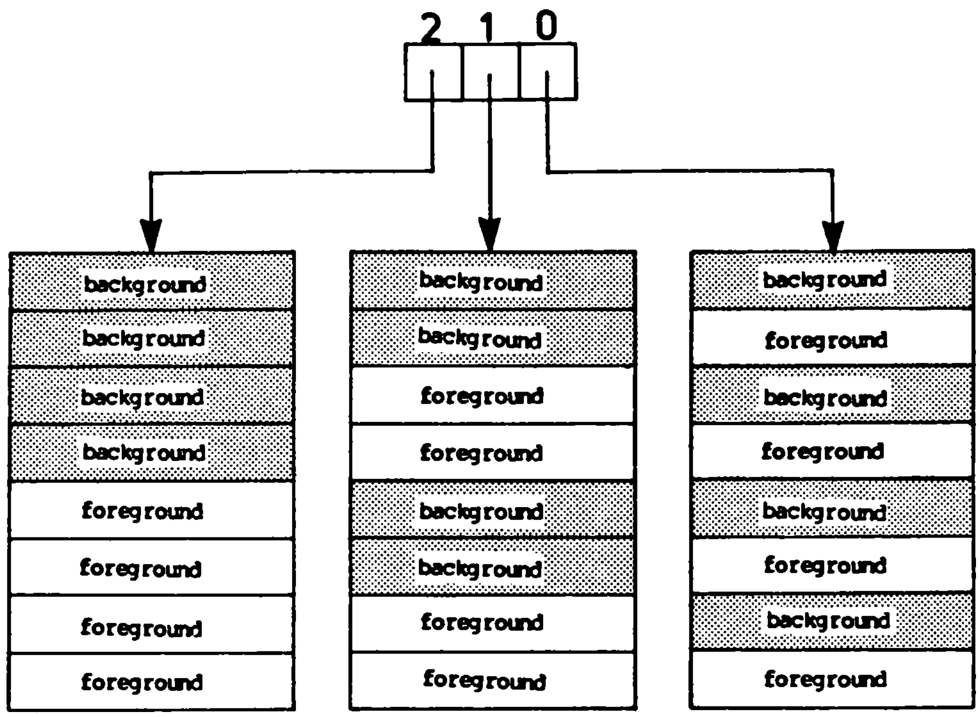

Figure 8 Colour Map Configurations for Bit-plane Extraction

The loss of spatial resolution can have the unfortunate property of accentuating the imperfections and reducing small details of an image to almost invisible proportions. If the gain in intensity resolution is exploited by techniques such as antialiasing these effects can be decreased. For the particular example of 256x256 subimages it is interesting to note that for a large class of pictures the apparent resolution of a 256x256 image in which multiple intensity levels are used for antialiasing will be superior to that of a 512x512 rendering of the same scene with no antialiasing.

A test case for this experiment was constructed consisting of two representations of a wire frame sphere, one at 512x512 resolution without antialiasing the other at 256x256 with antialiasing. To achieve a fair comparison, the 512x512 version of the picture was rendered with double thickness lines and the 256x256 version used an antialiasing routine that simulated a zoom so that both could be displayed on the monitor simultaneously. No exact measurements were made, but we observed that the lines of the 256x256 version appeared smoother than those of image ren-

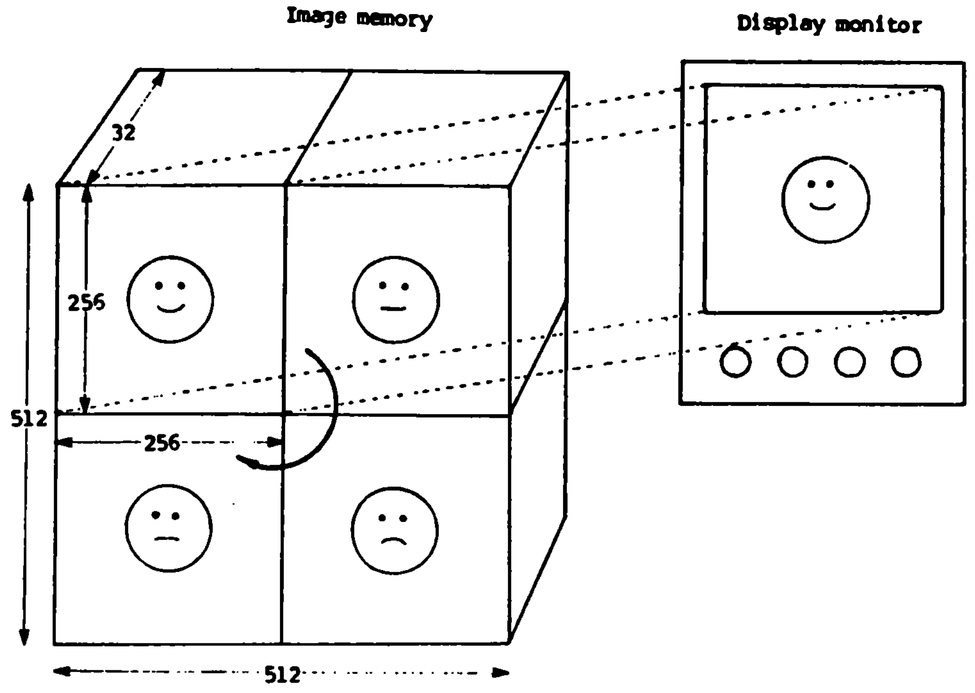

Figure 9 Zoom-pan-scroll Animation

dered at 512x512 resolution. The results cannot be convincingly reproduced here, but must be viewed on a video monitor for accurate comparison. This observation does not appear to generalize below 256x256 where the eye simply does not trade intensity for spatial resolution.

The zoom-pan-scroll technique can be extended to more subimages, at the expense of even further reduction in spatial resolution, as illustrated in Figure 10 in which sixteen different views of an Imperial Snow Walker (courtesy of Terry Higgins) are shown. Animating the images produces a fairly realistic effect, somewhat diminished by the low quality of the 128x128 subimages.

Crossbar Animation

Just as zoom-pan-scroll animation allows the partitioning of the frame buffer spatially into smaller units, the crossbar switch provides a means of subdividing the intensity resolution of each pixel so that again each image is stored in its own smaller logical frame buffer. In the simplest case, each bit in every pixel is used for a separate image, giving thirty-two images at 512x512x1 resolution. Animation is accomplished by setting the crossbar switch to only pass the "visible"

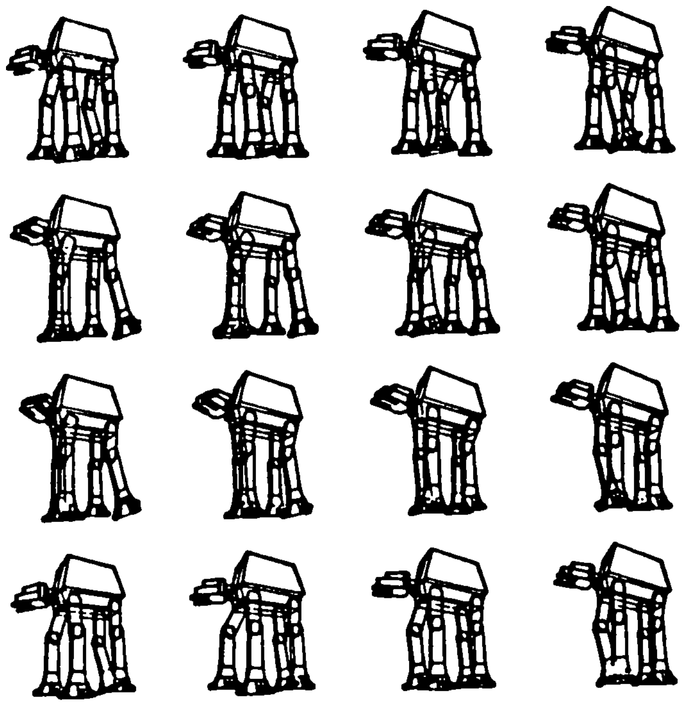

Figure 10 Zoom-pan-scroll Data

bit of each pixel to the colour table, blocking the rest. The full price must be paid for the reduction in intensity; each visible image has one bit resolution so only two colours will be selectable at any given time.

With a 32-bit frame buffer it is possible to store two 16-bit images, four 8-bit images, eight 4-bit images, sixteen 2-bit images, or thirty-two 1-bit images. Other partitionings are also available, such as ten 3-bit images, with the extra bit-planes discarded or used to hold a permanent background description. Figure 11 illustrates the crossbar switch settings for displaying a 4-bit image using a sixteen entry colour table.

Although each image is displayed at full frame buffer resolution, image quality can still be improved with antialiasing techniques. While our simple antialiasing is not appropriate for 1-bit intensity and provides only marginal improvement for the 2-bit case, there are clear improvements in the apparent resolution of the picture using three or more bits. Experimentation has shown the results of 3-bit antialiasing are almost as good as four or more bits. It is therefore a tradeoff whether to sacrifice a little picture quality to have more images in the animation

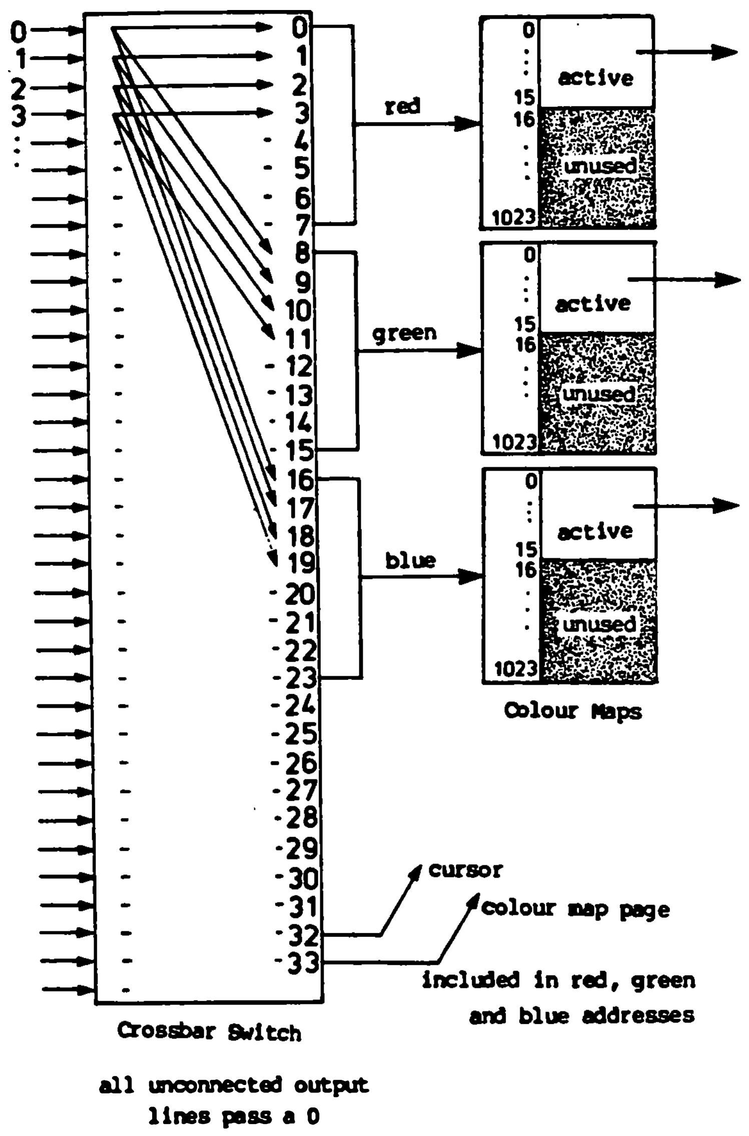

Figure 11 Crossbar Switch Settings to Display Bit-planes 0-3

sequence.

Crossbar animation is similar to the bit-plane extraction described with colour table animation. The crossbar switch has the advantage that more planes can be accessed more efficiently. With a standard 256 entry colour table, only eight bit-planes can be extracted using colour table techniques. Increasing the length of the colour map to allow thirty-two planes to be accessed would be prohibitively expensive. A colour table allowing control over eighteen bit-planes would require an address word as large as that for the entire image memory. Although random access is required at video rates to both image memory and the colour tables, several pixels can be read at once from image memory (the Ikonas reads 16) because the data is read in strict scan line order. Normally this is not the case for the colour tables because of the inherent random access characteristic of a lookup table.

Combining the Techniques

To extract maximum usage from the frame buffer hardware and to increase the running time of the animation sequences it is possible to exploit the orthogonality of the techniques discussed so far. The crossbar switch can be used independently of the frame buffer control registers and of the colour table (except that the number of colours available may be reduced) and the colour table in turn is also independent of the controller.

One particularly effective technique is *frame buffer partitioning,* which results from a combination of crossbar and zoom-pan-scroll animation. The result is a three dimensional matrix of smaller frame buffer images that can be animated analogously to the two dimensional case. Typical image resolutions range from 256x256x8 (sixteen good quality images) to 128x128x1 (512 poor quality images), the latter giving about seventeen seconds of real-time animation. Values such as 256x256x3 (forty images) or 170x170x4 (seventy-two images) are more practical. Although different partitionings will achieve the same quantity of images, 128x128x16 and 256x256x4 for example, it is often best to select the option requiring the least amount of zoom. Assuming a minimum number of bits for antialiasing and a zoomed resolution of less than 256x256, the loss of resolution due to zooming is more apparent to the eye than that of intensity reduction.

With any of zoom-pan-scroll animation, crossbar animation or frame buffer partitioning, enhanced dynamics can be provided with the addition of colour table techniques. Within each partitioned mini-frame buffer it is possible to perform any of the forms of colour table animation described in the earlier section. There is of course a limitation if the intensity resolution has been reduced and the available colours have been diminished, but assuming even three bits of depth reasonable effects still can be created with only eight active entries in the table.

Pure colour table animation can itself be enhanced by such tricks as performing colour cycling in one part of the map and alternate colour animation in another, or through the addition of extra selectable colour maps to lengthen the rotation for colour cycling. The number of ways to combine the basic frame buffer animation techniques for usable results is only limited by the imagination of the animator and the restrictions inherent in the chosen animation sequence.

Telidon: An Animation Application

The primary application of the animation techniques described so far is in the area of low-cost graphics. In particular, the rapidly growing field of videotex could well be a major beneficiary. Future videotex systems such as Telidon must incorporate at least some of these features if they are to become competitors in this potentially lucrative market. The steady drop in hardware costs will soon make many of the techniques cost-effective, especially in the projected mass markets for videotex terminals. Full 32-bit frame buffers will probably remain beyond the price range of the average consumer for some time to come, but modest extensions to include colour lookup tables and zoom-pan-scroll registers are easily within today's technology.

Animation currently performed on most prototype Telidon systems is extremely primitive and as a consequence is perceived as simplistic. Typically, animation is accomplished by drawing an image, redrawing it in the background colour and drawing the next image in the sequence at a new position. If the scene is simple enough, the effect (and restrictions) are similar to alternate colour animation. If a moving object is complicated or the transmission speed is slow, the animation will no longer be smooth because of the large amounts of data that are being transmitted from the host to redraw each image.

The addition of a colour lookup table, as proposed in both the AT&T Presentation Level Protocol [1] and the latest Telidon protocol revision [21], will certainly increase the flexibility of the system, providing a more reasonable selection of colours as well as the ability to antialias and animate images.

Unfortunately, the designers seem not to have grasped the importance of this advance. They discourage its use in all but the most advanced cases (like colour table animation) and provide instead a separate means for specifying colours directly because there exist terminals that do not have hardware colour maps [21, p. 109]. User confusion could be reduced if instead the default colour table [21, p. E3] were assumed for the more primitive terminals, so that colour table usage would be encouraged.

Another problem with present Telidon animation applications is that of transmission speeds. Although colour table techniques reduce the transmission bandwidth bottleneck, few home users (after the novelty wears off) will be prepared to sit in front of a terminal watching a flashy special effect knowing they are paying the cost of each flash. A much better arrangement would take advantage of the fact that each Telidon terminal has a microprocessor. A small program could be downloaded to the terminal to handle animation or similar effects. This can be accomplished in a number of ways.

One solution, for often used programs, would be to define a new primitive that would start the preloaded program with the transmission of a single command. Another is the telesoftware solution that has been considered for Telidon [21, p. 114] and is already being used in the Austrian Mupid system [19]. The two telesoftware approaches differ slightly. Telidon suggests that processor independent code be downloaded for interpretation by the terminal's microprocessor, while the Mupid approach is to determine the processor type of the terminal and download code specific to that machine. At the user level these differences are minor. Telesoftware is also useful for games and other home computer applications. These two solutions are both possible in one system with often used code cached and less frequently used code downloaded when needed.

Although the addition of colour tables to Telidon is undoubtedly more important than the inclusion of some of the other advanced features of the video chain, those contributions should not be overlooked. The modules in an enhanced video chain would have other uses besides frame buffer animation techniques. For example, the addition of a zoom feature would provide the control over fine detail necessary to implement an effective information provider system. Other hardware enhancements would be useful for providing the interactive feedback necessary (such as instantaneous response for menu driven systems) if Telidon is to become a truly two-way system.

Further Enhancements

More powerful hardware and further enhancements in software will expand the uses and increase the simplicity of the techniques already discussed and may lead to the development of better techniques. Hardware improvements would include a crossbar switch on the input side of image memory, a more flexible crossbar switch, or enhanced video control registers. Software developments should first be concentrated on new ways to create sequences for animation. This section provides guidance to those interested in expanding on the ideas presented here.

Hardware

An additional feature that a future hardware designer may want to consider is an *input crossbar switch,* ideally one for each processor, as with the write mask. Each crossbar switch would sit between its processor and the pixel memory so that writes as well as reads could be re-routed. This would simplify the software involved in loading an image into memory. An example of a use for this feature is a z-buffer algorithm. Standard double buffering requires that the processor alternately write into separate 8-plane banks of pixel memory. With the input crossbar switch, the double buffering becomes almost transparent to the scan conversion routine. Because the current buffer always appears to be in the same bit positions the bits of data do not have to shifted before writing. A similar advantage is obtained for bit-plane extraction animation.

Another hardware enhancement that should be considered is a more flexible crossbar switch. It should be possible to select "constant" outputs of either 0's or 1's for each output bit, rather than only the single constant value presently provided. The ability to select bits from the current (x,y) address (at least from the low-order bits, convenient for generating textured patterns) would be an asset.

Some of these features are special cases of a more general bus structure that allows many operations to be performed on pixel values as they are written into memory. Crow describes a particular implementation [10].

Further enhancements should be made to the frame buffer controller. Modifying the window register while viewing an image that has been zoomed results in each pixel always being replicated the number of times specified by the zoom factor. Thus the zoomed image makes noticeably discrete jumps during pans and scrolls. This can be overcome by adding two new zoom registers that contain horizontal and vertical replication counts used for only the first pixel on each scan line and the first scan line in the window. The replication count can be limited to at most 256 (represented as 255 within the register) as for the current zoom values, so all four counts can be stored within the current 32-bit Register 3.

Finally, it would be particularly convenient to have independent control over each bit-plane. One of the few frame buffers implementing this idea completely is the Norpak VDP-1. It has separate zoom, pan, and scroll registers for each bit-plane. This allows portions of the picture to be moved relative to each other providing an animation capability not available with any of the previously mentioned techniques. Unfortunately the later VDP-2 controls four bit-planes with a single register (another concession to memory organization) except with optional high speed memory. This is useful, but would be more useful if a crossbar switch were also present. The Ramtek 9400 frame buffer has a similar feature, again operating only on multiple bit-planes.

Software

The chief drawback to the animation techniques as described lies not with the techniques themselves, but in their implementation. Coding them as they are stated here will leave a collection of programs that will produce effective results only for specially created examples. The problem is that the animator is forced to design animation sequences that fit the techniques he wants to use rather than creating a set of pictures to be animated and having the animation system select an appropriate technique.

We must find a reasonable way to create the sequences for animation. It is not necessary to completely divorce the animation technique from the sequence, as some images are undoubtedly more suited to one technique than another, but what must be done is to create a simple way of allowing a naive user to take a set of frames that he has created in some way and put them together within the frame buffer to form an animated sequence.

One method would be the addition of animation primitives to a three-dimensional graphics package or preferably to a scene description language interfaced to a graphics package. For example, the viewport specifications could allow positioning in the x, y and z directions and with simple transformations, each three-dimensional viewport could contain a slightly different view ready for animation.

Another possible solution is an enhanced paint program with some special features built in to support animation. The "Superpaint" system described by Shoup [24] has already implemented such ideas for colour table animation sequences created with a modified paint program.

Probably a more interesting example would be an implementation of a paint program that would allow general frame buffer partitioning. It would not be necessary to design a completely new system to handle the additional features. Instead an existing system, such as the multiprocess Paint program developed in the Computer Graphics Laboratory [4, 22] could be expanded. The remainder of this section will present some suggestions for additions that could be made to Paint to accommodate frame buffer partitioning.

A facility would have to be added to specify the size of the sub-buffer needed by each image. The specification would have to include all three dimensions, height, width and depth. From this, Paint would compute the maximum number of frames possible with those dimensions, where each frame is stored both during creation and for the completed sequence, the maximum number of frames that can be stored for interactive viewing during a Paint session, and a default colour map based on the depth of each pixel in the partitioned buffer.

The animator would also be required to specify the frame number, within the sequence, of each image as it is designed so that its proper location within the frame buffer could be computed and some reasonable name could be generated if it were necessary for Paint to store it on disk to make room for newer frames in image memory. Because the current implementation of Paint uses sixteen bit-planes of working storage in image memory, the entire animation sequence could not be stored in the frame buffer. It would be reasonable to store the most recent images, as many as possible, in image memory so that the animator could interactively check his work. If he wanted to examine the entire sequence he would have to be prepared to wait a little longer while all frames in the sequence are brought in from secondary storage.

The animation itself could be handled with the addition of an animator process. Using a set of menus the details of the animation such as timing, start and stop frames, and the number of repetitions in a cycle could be conveniently controlled. A preview facility would also be useful for stepping through the sequence to check continuity.

Finally, to ensure frame to frame coherence, it is necessary to be able to copy images between bit-planes and to different sectors of the same set of planes. In many cases, smooth blending of each image into the next is easier if we start with a copy of the previous frame and make slight changes to it.

One serious problem that will not be dealt with here is that of antialiasing painted scenes. As discussed in the animation section, antialiasing is crucial in an environment where zoomed images will viewed. The problem is more difficult in an environment such as the enhanced paint program discussed here because the user is given a wide choice of colours but is only permitted to work with a palette of limited size. If a partitioned buffer of 256x256x4 is selected there will be sixteen different colour values available. Assuming a worst case in which every colour must be able to blend into every other colour, only three unique colours (using twelve table entries) are really available for use. Levoy [18] discusses some simplifying assumptions that can be made to expand the usable palette.

Acknowledgements

A preliminary version of this essay was presented at Graphics Interface '82 and appears in the Proceedings.

I would like to thank my advisor Kelly Booth for his helpful suggestions that formed a basis for many of the techniques described here and for his gentle prodding when my procrastination became serious. Thanks go to Darlene Plebon for her patience and effort in helping to sort out the details of Telidon, the Ikonas system, Paint and the other little problems that always seemed to appear. I am also grateful to Rick Beach for his insights into paint programs and to Dave Martindale for his patience in explaining hardware to a software person.

I would also like to credit Paul Breslin, Darlene Plebon, Terry Higgins and all the anonymous authors whose software and three dimensional scene data were sources of test examples for my work. Finally, I must thank Doris Kochanek, Carol Hayes, Sylvia Lea, Julie Pakalns, Baldev Singh and the many other friends and colleagues who bombarded me with helpful suggestions that allowed me to get through it all.

References

[1] American Telephone and Telegraph Company. *Presentation Level Protocol: Videotex Standard.* May 1979.

[2] Daniel H. Bass. Using the video lookup table for reflectivity calculations: Specific techniques and graphics results. *Computer Graphics and Image Processing* 17:3, pages 249-261, November 1981.

[3] R. M. Baecker. Digital video display systems and dynamic graphics. *Computer Graphics* 13:2 (Siggraph '79 Conference), pages 48-56, August 1979.

[4] Richard J. Beach, John C. Beatty, Kellogg S. Booth, Darlene A. Plebon, and Eugene L. Fiume. The message is the medium: Multiprocess structuring of an interactive paint program. *Computer Graphics* 16:3 (Siggraph '81 Conference), pages 277-287, July 1982.

[5] K. S. Booth, D. H. U. Kochanek, and M. Wein. Computer animation in the 80's. *IEEE Spectrum,* accepted for publication.

[6] Kellogg S. Booth, and Stephen A. MacKay. Techniques for frame buffer animation. *Proceedings: Graphics Interface '82,* The Canadian Man-Computer Communications Society and the National Computer Graphics Association of Canada, pages 213-220, May 1982.

[7] H. G. Bown, C. D. O'Brien, W. Sawchuk, and J. R. Storey. *A General Description of Telidon: A Canadian Proposal for Videotex Systems.* Communications Research Centre Technical Note No. 699-E, Canadian Department of Communications, December 1978.

[8] *Canadian Dictionary of the English Language.* Houghton Mifflin Canada, 1980. Animate.

[9] E. Catmull. The problems of computer-assisted animation. *Computer Graphics* 12:3 (Siggraph '78 Conference), pages 348-353, August 1978.

[10] F. C. Crow and M. W. Howard. A frame buffer system with enhanced functionality. *Computer Graphics* 15:2 (Siggraph '81 Conference), pages 63-69, August 1981.

[11] J. D. Foley and A. van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982.

[12] Preston Gurd. A Microcode C compiler for a bit-sliced microprocessor. Master's Thesis, University of Waterloo, in preparation.

[13] Ikonas Graphics Systems. *Ikonas Programming Reference Manual*. 1980.

[14] Ikonas Graphics Systems. *Ikonas User's Guide*. 1980.

[15] Ikonas Graphics Systems. Product specifications. May 1982.

[16] Doris Kochanek. A computer system for smooth keyframe animation. Master's Thesis, University of Waterloo, 1982.

[17] Sylvia C. Lea. Fecit: A structured language for describing computer generated scenes. Master's Thesis, University of Waterloo, in preparation.

[18] Marc Levoy. Frame buffer configurations for Paint programs. *Tutorial: Two Dimensional Computer Animation* (Siggraph '81 Conference), pages 18-29, August 1981.

[19] H. Maurer, Technical University - Graz. Mupid - An Austrian contribution to videotex. University of Waterloo, Department of Computer Science Colloquium, 7 July 1982.

[20] W. M. Newman and R. F. Sproull. *Fundamentals of Interactive Graphics*. McGraw-Hill, 1979.

[21] C. D. O'Brien, H. G. Bown, J. C. Smirle, Y. F. Lum, and J. Z. Kuhulka. *Telidon: Videotex Presentation Level Protocol: Augmented Picture Description Instructions*. Communications Research Centre Technical Note No. 709-E, Canadian Department of Communications, February 1982.

[22] Darlene Plebon. Interactive picture creation systems. Master's Thesis, University of Waterloo, 1982.

[23] Richard G. Shoup. Colour table animation. *Computer Graphics* 13:2 (Siggraph '79 Conference), pages 8-13, August 1979.

[24] Richard G. Shoup. "Superpaint"... The digital animator. *Datamation,* May 1979.

[25] Baldev Singh. A graphical editor for Benesh Movement Notation. Master's Thesis, University of Waterloo, 1982.

[26] K. R. Sloan Jr. and C. M. Brown. Colour map techniques. *Computer Graphics and Image Processing* 10:4, pages 297-317, August 1979.