Colour Selection, Swath Brushes and Memory Architectures for Paint Systems.

P. Tanner W. Cowan* M. Wein
Division of Electrical Engineering
*Division of Physics
National Research Council of Canada
Ottawa, Ontario, Canada

ABSTRACT

The development of raster graphics based on frame-buffer technology has introduced a new artists' tool - the computer paint system. With such a system, an artist uses such tools as "instance brushes", "simulated air brushes", and "simulated acrylic paint" to create an image, using them as he would a brush on canvas.

This paper describes developments in three aspects of paint systems: colour selection, swath brushes, and memory architecture. The colour selection technique uses the tablet in a two-dimensional mode, varying hue and saturation integrally. A separate, one-dimensional mode varies brightness. The swath brush is a brush that affects a swath of the frame buffer in an even and controlled way. This swath follows the motion of the tablet stylus. With the decreasing cost of memory, raster systems can be built with a memory architecture specifically suited for paint systems. This paper describes an architecture that greatly speeds the read-modify-write pixel operation which is used so frequently in such systems.

RESUME

Le développement de l'infographie à quadrillage utilisant la technologie de mémoire-image a donné à l'artiste un nouvel outil: le système de peinture informatisé. Un tel système met à sa disposition des outils comme des "pinceaux ponctuels", des "pinceaux vaporisateurs simulés" et de la "peinture acrylique simulée" lui permettant de créer une image, un peu comme avec un pinceau sur une toile.

Le document décrit les perfectionnements de trois aspects des systèmes de peinture: la sélection des couleurs, les coups de pinceau et l'architecture de mémoire. La technique de sélection des couleurs utilise une tablette en mode bidimensionnel, faisant varier intégralement la teinte et la saturation. Un mode unidimensionnel distinct permet de faire varier l'intensité. Le coup de pinceau est une opération qui permet d'introduire un trait uniforme et commandé dans la mémoire-image. Le trait suit le mouvement du style de la tablette. Etant donné la diminution du coût des mémoires, il est possible de fabriquer des systèmes à trame dont l'architecture de mémoire est spécialement adaptée aux systèmes de peinture. Le document décrit une architecture qui accélère grandement l'opération lecture-modification-écriture de pixels, si fréquemment utilisée dans de tels systèmes.

1.0 INTRODUCTION

Any paint system is a set of complementary tools, adding one tool often increases the power or capibilities of the others. This paper describes two new tools as well as the design of a memory architecture for a raster paint station. The first tool, described in section 2, is a technique for colour selection - one that assists the user in selecting a desired colour more easily and quickly. Colour selection is, of course, integral to the use of any paint system. Section 3 describes the second tool, a swath brush. This brush modifies the display in a more even manner than does the more common instance brush, and supports such effects as transparent brushes and brushes that combine transparency with tint paint or value paint. Paint systems make very special use of the frame buffer system on which they reside. In particular, the pixel-blending (the combining of an input colour with the current pixel colour) of many brush types taxes the resources of most frame-buffer systems. With memory costs going down, a memory architecture designed specifically to support pixel-blending in particular, and paint systems in general now becomes feasible. Such a design is described in Section 4.

1.1 Paint Systems

A paint system, given an input such as the path of a tablet stylus or a mouse, modifies pixels in a frame-buffer on or about that path. Typically, the user defines a "brush" as a specific coloured pattern.

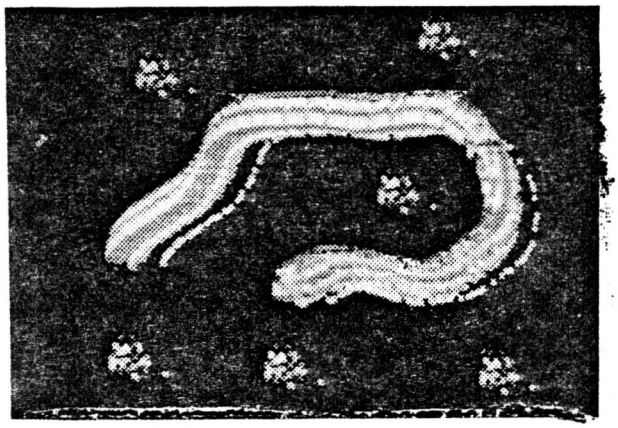

Figure 1. A brush has been "rubber stamped" on the "canvas" six times. The long streak is the same brush brushed over the surface of the tablet.

The artist may then "rubber stamp" this brush at will anywhere on his "canvas" (display screen) with the tablet pen (Figure 1). By moving the pen along the surface of the tablet, the artist can make a series of brush instances (also in Figure 1). Notice in this figure that the spacing between instances varies because the speed that the artist paints is too fast for the host computer. This problem of speed will be discussed in more detail in both sections 3 and 4.

Basic to many paint system capabilities is the concept of pixel blending - the blending of the pixel colour of a brush with the original frame buffer pixel colour. Whenever a brush is rubber-stamped into a frame-buffer, the resulting image is a function of both the current pixel values in the frame buffer and the pixel values of the brush. If a brush were to have a transparency component of 0.2, each pixel that is written during the stamping operation would have a final value of 0.2x(brush pixel value) + (1.0-0.2)x(original pixel value). The brush colours are blended with the background colours so as to make the brush appear transparent. Figure 2 gives an example of the brush in Figure 1 being stamped on different coloured backgrounds in non-transparent mode (top corners), stamped in transparent mode (each non-black quadrant), and brushed in transparent mode in the centre. Notice that the result of the brush is quite different depending on which colour it is combined with.

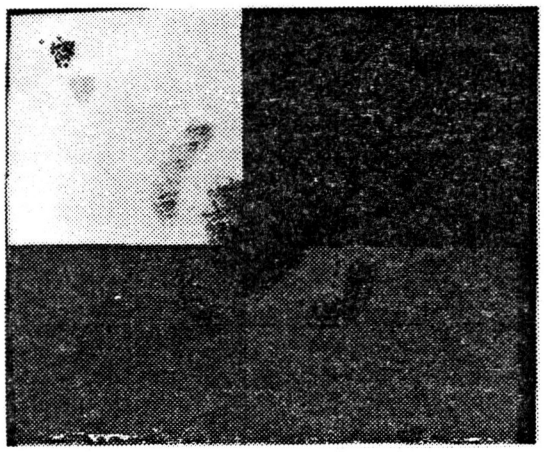

Figure 2. A transparent brush stamped and brushed onto different coloured backgrounds.

There are many other paint effects that can be obtained by taking functions of the brush colour and the current pixel colour. For example, TINT paint uses the hue and saturation of the brush while leaving the intensity of the pixel untouched. Inversely, a VALUE brush sets

the pixel to the intensity or value of the brush while not affecting the colour (hue and saturation) of the pixel. Both of these techniques can be combined with the transparency function so that a stroke of the brush will produce a subtler effect. Other brush functions blend colours from a neighbourhood around the brush in such a way as to simulate a smearing effect. Alvy Ray Smith describes a large set of painting techniques in an excellent paper on his own paint system [Smit78a].

1.2 Applications

Television studios, conventional and experimental animation studios and graphics design artists can all make use of paint systems. Initially, these systems were developed for artists to paint background images for computer animated films. The foreground animation made use of two-dimensional keyframe or three-dimensional geometric modelling techniques. Of course, richly coloured painted images contrasted with the structurally more complicated, but simply coloured animated objects. As the idea of mapping shaded images onto three-dimensional geometric shapes has been around sice 1976 [Blin76], it was a logical step to take painted images and map these onto three-dimensional geometric objects. Perhaps the most familiar examples of such mappings are the planets and moons in Jim Blinn's films made at JPL for NASA [Blin80, Blin81, Blin82]. Another application is the use of a painted image for texture control [Blin78]. This texture control can be mapped onto a three-dimensional object so as to vary the intensity of light reflecting from the object, thereby giving it an apparent texture.

Commercially, paint systems are frequently used in the advertising business. Photographs or frames of video, after they have been digitized, can be manipulated and modified in the same way as a painted image. Real and synthetic images may be combined.

A newer, but little used application of the paint program is direct animation. Each frame of an image is either painted separately, or created by modifying another frame by overpainting. This process, although time consuming, has the potential for quite stunning sequences.

2.0 COLOUR SELECTION.

In a paint program the brush changes the colour of an area of the screen. Associated with the brush is a target colour, which may be the final colour of the area, or which may be used in deriving the final colour, as with pixel blending. Somehow, the artist must choose the colour.

2.1 Colour Selection Desiderata.

A colour selection program should have as many of the following properties as possible:

1. Completeness - It should make available the full gamut of colours. This means not only fully-saturated colours, but also colours that are arbitrarily close together. If 4,000,000 colours are available, each should be individually selectable.

2. Predictablity - The artist should be able easily to predict which operations are needed to get to any new colour.

3. Transferability - The artist should easily know from the selection process how the colour will look in the image. Deficiencies in transferability occur most often because of contrast effects [Goet82], which happen when the colour has a different environment in the image than it does during the selection process.

4. Simplicity - Colour selection is a small part of a large program; its operation should be kept as simple as possible.

Most colour selection schemes fail to provide one or more of these features. To be fair, however, we should note that traditional artists media: paint, etc., also fail to provide them all.

2.2 Current Colour Selection Methods.

Two methods are currently in wide use for colour selection. The first is the palette type. Small patches of a variety of colours are visible to the artist, who uses a pick operation to select one of them. This method is predictable and simple. It has reasonable transferability, since the colour patch is surrounded by a variety of colours in the palette, as it is in the image. But, except for systems with small (<100) numbers of colours available, it is seriously incomplete. One solution is to introduce a hierarchy of

palettes, each containing a narrower range of colours separated by smaller steps. Note: if 4,000,000 colours are made available in a 32 colour palette, at least five levels of hierarchy are needed. Such solutions, to create completeness, eliminate simplicity. They also lessen transferability, since colours in the palette are now all very similar.

The second uses one or another colour order scheme. The artist sees three indicators which show the current value of each dimension of the chosen scheme (RGB, HSV, etc.); he also sees a patch of colour produced by the values. Three valuators change the values until the artist sees the colour he wants. This method is complete and simple, though less so than the palette. It is, however, unpredictable, primarily because all three controls must be altered for a general colour change. It is difficult to learn how to get around. HSV is easier than RGB [Smit78b], but, in our experience, both require considerable learning. It is deficient in transferability. The selection area has a black background, so that when the colour is moved to the image, where it is surrounded by other colours, large changes in appearance can occur [Goet82]. Brown, olive, and navy blue colours present particular difficulties.

2.3 A New Colour Selection Method.

Our new method, a variety of colour order scheme, is based on the distinction between separable and integral properties. Properties are separable if they are perceived independently, as are the redness and squareness of a red square. They are integral if they are perceived as different dimensions of a unitary percept, as are the hue and saturarion of a red square. ([Garn82] and [Garn70] give more detailed explanations of this distinction.) Clearly, the three dimensions of a colour (RGB, HSV, XYZ, etc.) are integral; we should vary colours using a peripheral device having three integral dimensions. But, to keep a paint program simple we are constrained to use an already given device for colour selection, usually a digitizing tablet, which has two integral dimensions. Fortunately, the brightness dimension (we are speaking roughly, so this is about the same as luminance, value, lightness, etc) is relatively separable. Thus, we use a one-dimensional valuator to control brightness and a two dimensional one to control the two conjugate dimensions (roughly, chrominance, hue-saturation, etc.). Each is implemented as a mode of the digitizing tablet, so that to select a colour the artist varies brightness, then chrominance, then brightness, and so on.

Using C syntax for illustrative purposes, the transformation from these coordinates to the RGB colour space of the monitor is the following. First, in brightness mode,

```
for red, green, and blue guns
   voltage[gun] *= (1.0 + delta / SCALE1);
   voltage[gun] = max(LOWERLIMIT,
                  min(UPPERLIMIT, voltage[gun]));
```

where delta is the valuator change, and SCALE1 is a scaling factor. UPPERLIMIT prevents the voltage from exceeding the range of the system; LOWERLIMIT from falling below a finite value (low enough to be below the black-clamping level of the monitor). The effect of these limits is to cause saturated colours to converge to the white (black) corner of the RGB colour cube (along maximum saturation faces) as they get brighter/dimmer.

In the chrominance mode

```
voltage[red] += deltax / SCALE2;
voltage[red] = max(0, min(UPPERLIMIT,
                          voltage[red]);
voltage[green] += deltay / SCALE2;
voltage[green] = max(0, min(UPPERLIMIT,
                          voltage[green]);
voltage[blue] = voltage_sum - voltage[red]
                            - voltage[green];
```

where deltax/deltay is the valuator change in the x/y direction, and voltage_sum is the sum of the three voltages, which is held constant in the chrominance mode. The blue voltage must also be kept between 0 and UPPERLIMIT. The method we chose translates the red and green voltages along a 45 degree line (in the red-green plane) until they reach the line where voltage_sum is a constant.

```
if (voltage[blue] < 0)
   voltage[blue] = 0;
else if (voltage[blue] > UPPERLIMIT)
   voltage[blue] = UPPERLIMIT;
voltage[red] = (voltage_sum - voltage[blue]
          + voltage[red] - voltage[green]) / 2;
voltage[green] = voltage_sum - voltage[blue]
                             - voltage[red];
```

This algorithm puts red, yellow, green, and blue colours in the four corners of the digitizing tablet. They are arranged in a hue circuit, with white in the middle, as suggested by opponent-colours theory [Hurv81]. Brightness is not held exactly constant in this adjustment mode. Gamma correction and calibration of the monitor would be needed to do so, as well as

considerable knowledge of visual system function. Our experience suggests, indeed, that constancy of brightness is unimportant. What does matter is the hue circuit, and constancy of chrominance under brightness adjustment.

Informal experiments with this algorithm - users are asked to match a variable colour to a fixed one - shows that inexperienced users are able rapidly and reliably to find the colour they want with only a few minutes learning. Users experienced with either RGB or HSV comment on the ease of use of this system.

This algorithm makes colour control complete, simple, and predictable. However, transferability can still be a problem, and some difficult colours - browns, olives, navy blues - are hard to find. Surrounding the colour selection area with a grey-white region improves these problems. Contrast makes the difficult colours easy to find, and transferability is eased, since the surrounding region is, in a rough way, an average of possible surroundings the colour will have in the image.

2.4 Future Considerations

Several issues in colour selection remain worth investigating. One is colour selection in the image. Might the artist draw an area in the image, then change its colour in situ? Such a process, since shading should remain as the colour changes, would require novel blending algorithms. It is unclear how useful such a feature would be to the artist.

A second problem area concerns colour reference schemes. Colours are referred to by monitor-related systems (RGB, HSV, etc.), by light-related systems (XYZ, CIELab, etc.), and by object-related systems (Munsell, etc.). We do not know how the object-related system which the artist tries to reproduce compares to the monitor-related system which is his medium. Further improvements in colour selection must rest on better knowledge in this area.

3.0 SWATH BRUSHES

One traditional method of computer painting is the rubber stamp approach. A "brush" is defined which is a small, arbitrarily shaped area of pixels, with each pixel having an associated colour. The user can then, by positioning the tablet tracker on the screen, cause one instance of the brush to appear on the screen with each push of the tablet pen. Moving the pen, while applying downward pressure, will cause a series of instances of the brush. Smooth area filling brushes are often implemented in this fashion - Figure 3 shows the outlines of several instances of a circle brush as it is brushed along a path. Figure 4 shows the result - an area similar to the shape of the path, filled in with some colour.

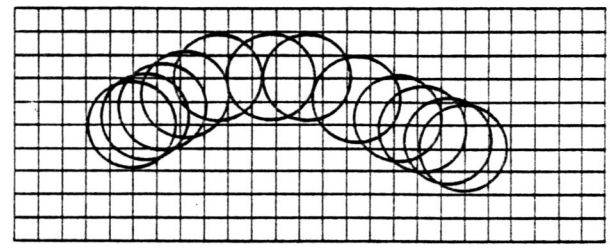

Figure 3. A series of instances of a circular brush.

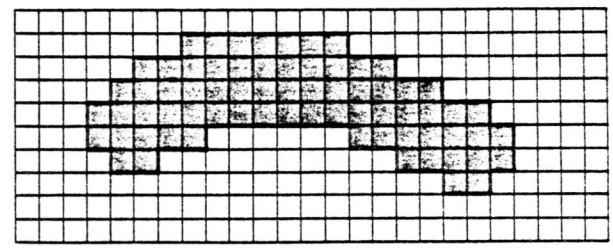

Figure 4. The area filled by the brush stroke of Figure 3.

This method becomes less than ideal when a pixel blending brush is being used. Consider the example in which the brush of Figure 3 adds 10% of its colour (colour A) to 90% of the existing colour in the pixel. In this case, every pixel centre-point surrounded by one instance of the brush will be 10% colour A. If the pixel is affected by two instances of the brush, it contains 19% colour A (not 20% due to the compounding effect). Figure 5 shows the percent of colour A in the various pixels. Notice the extreme unevenness of the result.

One method used to give the artist control to produce a smoother brush effect is to make the transparency of the brush vary with the distance from its centre. Applying a convolution function to the transparency, so that the brush is somewhat opaque in the centre, and fades to total transparency around the edge, yields the effect we see in Figure 6. This is indeed a smoother result, but improvements can still be made. The swath of colour made with a

variable intensity brush varies with the frequency of writing brush instances into the frame buffer. In Figure 3, the brush instances at the left are more frequent than those closer to the centre. This may be due to the varying speed of the artist's hand or perhaps competition from other jobs on a time sharing system. Intensity variation due to the speed of the brush may at times be acceptable, although often an artist would require a more consistent result. Effects due to the loading of the system should always be avoided.

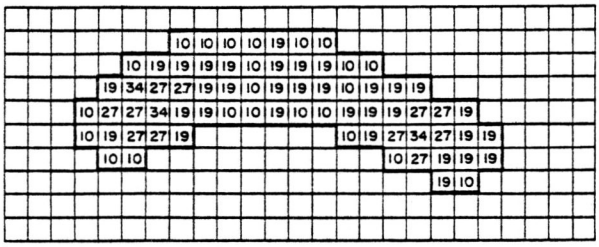

Figure 5. Percentage of brush colour blended into each pixel using a brush of constant transparency.

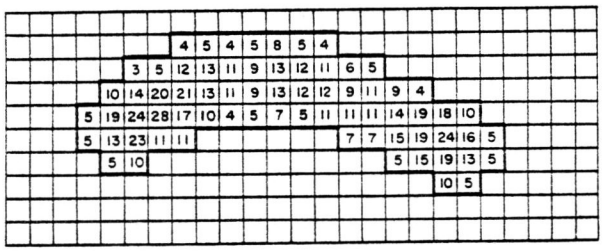

Figure 6. Percentage of brush colour blended into each pixel using a brush that is more transparent around the edges.

A brush that affects a swath of pixels in an even and consistent manner can be implemented as a series of vectors, each vector exactly one pixel away from the next in a horizontal or vertical direction (figure 7). Each vector can then be plotted with pixel blending, by the software or firmware of the frame buffer processor.

In more detail, the algorithm works as follows.

1. The first tablet value (x1, y1, status) is read (if the pen is not depressed, the value is of course ignored and step 1 is repeated).

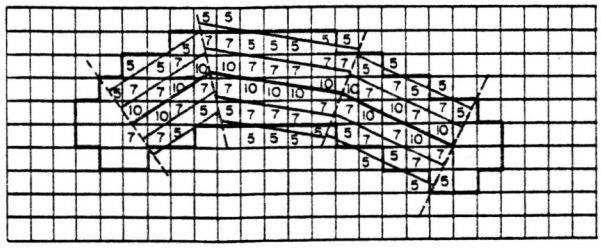

Figure 7. Swath brush example showing percentage of brush colour blended into each pixel.

2. The second tablet value (x2, y2, status) is read. A set of vector endpoints perpendicular to (x1,y1) (x2,y2), is formed (the leftmost dashed line in Figure 7).

3. The next tablet value is read. (Note that nothing appears on the screen until three values are read.) We now have two control vectors (x1,y1) (x2,y2) and (x2,y2) (x3,y3). If these vectors have a similar slope, a set of vector endpoints is found along a line roughly bisecting the two vectors (ie. the second dashed line in Figure 7). If the two vectors do not have a similar slope, they are treated as two distinct line segments (Figure 8a). Treating these vectors similarly to smooth vector pairs would result in long pointed spears being drawn along the bisecting line (see Figure 8b).

The lines parallel to the control vector must be precisely an integral number of pixels away from the control vector in either a horizontal or vertical direction, or a few odd pixels will remain between pairs of vectors, or be affected by two vectors. The routine for finding the vector endpoints must do this very precisely.

The implementation of the swath brush has been simulated on a PDP 11 running a Norpak VDP1. The authors are currently in the process of moving it to a M68000 which talks directly to the GBUS of the VDP1. We hope to get artists' opinions of the swath brush concept during the summer of 1983.

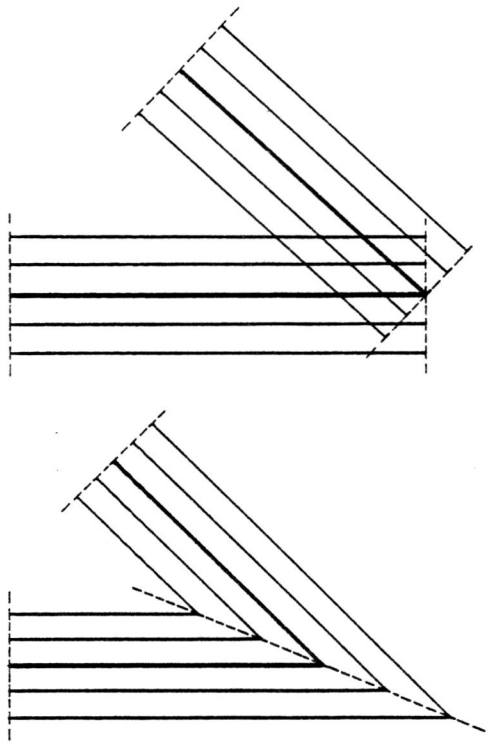

Figure 8. Sharp changes in slope cause line segments to be treated as separate strokes (top), or else long painted coloured areas result (bottom).

4.0 MEMORY ARCHITECTURE FOR PAINT SYSTEM

As described earlier in this paper, paint programs make extensive use of pixel blending - the writing of a colour into a pixel so that the resultant colour is some function of both the input colour and the colour originally in the frame buffer. With many raster frame buffers, this is a slow process. For each pixel to be modified, the value (24 or more bits) is read from the frame buffer into the host. Then the pixel blending function is applied, and the resulting value is written back into the frame buffer. This method can be unacceptably slow due to the time taken to initiate and process the data transfers between the host and the memory.

The performance of frame buffers is affected not only by the bandwidth of the data path between the host and the frame buffer, but also by the intrinsic bandwidth of the frame buffer. The intrinsic bandwidth is limited by the contention between host access and the display update. Ironically, the larger 64K by 1 bit memory circuits offer lower bandwidth than the older 16K by 1 bit memories, because fewer are used to implement a given resolution. However, the emergence of the newer 16K by 4 memory circuits will improve the intrinsic frame buffer bandwidth.

The following discussion compares the various ways frame buffers have been driven from the host computer. As well, a scheme is proposed in Section 4.5, as a way of solving some of the system problems and at the same time reducing the demand for high intrinsic bandwidth.

4.1 Frame Buffer Memory In The Address Space Of The Host.

This approach of placing the frame buffer memory in the address space of the host has been going in and out of fashion over the last 15 years, first with caligraphic displays, then with raster displays. Many early caligraphic display processors were designed such that the processor traversed the display list in the host main memory. Early frame buffers were also designed to have the pixel memory in the address space. A lack of address space on 16 bit hosts has led designers to "fold" the frame buffer space into 64KB of the memory space. Consequently at any time only vertical strips of the image were mapped into the host address space.

With the increasing complexity of operating systems, the direct mapped frame buffer has dropped from favour. However, the renewed interest in such an architecture has been stimulated by the emergence of powerful micro-processors such as the MC68000 used as dedicated hosts. In this way, one is re-implementing the dedicated host of 15 years ago but now on a micro. Such a wheel of reincarnation has been described by Myer [Myer68].

4.2 Frame Buffer On A Small Mini (PDP11/34 Or PDP11/45 Class) Using A Programmed I/O Full Duplex Channel.

This arrangement involves a frame buffer connected to the host using a bidirectional programmed interface. In such an interface the paint program has access to a set of input/output registers containing data for defining one pixel value: <R, G, B, row, col, control>. By writing to these registers, the

program has random read/write access to the frame buffer, one pixel at a time. Im most operating systems it is necessary to grant access privileges to the paint program to permit direct I/O commands from the program. Functionally, there is great similarity between this architecture and one in which the RGB pixel memories are actually in the address space of the processor (described in 4.1). The read/modify/write cycle is entirely programmed either in the paint process itself or as a separate spawned process.

4.3 A Frame Buffer On A DMA Channel

The third approach uses a conventional DMA channel between the host and the frame buffer. Data transfers between the host and the frame buffer are uni-directional at any one instant (half duplex). There is usually an overhead exceeding 2ms to set up each data transfer. Typically data blocks being transferred can be of arbitrary length and the transfer rate is up to 1 Mbyte/sec. At the very least there should be commands for transferring pixel rows. Preferably one would like to have a READ/WRITE PATTERN command as on Norpak VDP1 [Wein79], permitting the definition of an arbitrary relative set of pixels called a neighbourhood. This pixel pattern may be written, or read, by merely transmitting a reference x,y. A series of (RGB) values is then transmitted, one for each pixel in the pattern. Such a higher level command construct reduces the number of individual read requests necessitating reversal of the direction of transfer, as well it reduces the amount of positional data communicated.

4.4 Frame Buffer On A DMA Channel - Internal Memory Used As A Cache.

This approach is a variant of the third one, described in 4.3, in that the hardware is a DMA channel interface. However, the entire master RGB pixel image is kept in the host memory so that the painting process operates entirely in the internal memory. Every 50ms, a process wakes up and detects where the brush has been in the last 50-ms period and transmits a convex hull rectangle about the track (or alternately an arbitrary set of modified pixels determined in some other way) as a DMA block, from host to the frame buffer.

This technique has many advantages. Since a copy of the frame buffer is kept in host memory, the pixel blending process does not require reading pixel values back to the host. Secondly, the display frame-buffer memory has much less stringent speed and depth requirements than would be the case without the host buffer. For example one could conceivably have 3 bits for each of red, blue and green in the display memory, while retaining the full 8 bits for each colour required for pixel blending in the less expensive host buffer.

4.5 Frame Buffer On A Special Bus Interface With Host Memory

With the changing price trade-offs for memory integrated circuits, a memory can be designed that is particularly suited to the requirements of a paint station. The main requirement is, as mentioned above, a rapid read/modify/write cycle. Our system design for such a paint station is a hardware implementation of the approach just described in 4.4. The paint program runs in the host using main host memory for the RGB pixel data (as above). However, the main memory is used as a hardware "write-through" cache memory. An associative memory controller detects memory write cycles into that portion of the main memory which contains the RGB data and arranges the update of the external RGB frame buffer through a memory-to-memory transfer. To the extent that the analogy to a cache is valid, it is a perfect cache, with a hit ratio of unity. As with a cache, the access time to the external memory is longer than to the internal one. Typically, there is a latency time until a frame buffer write cycle is available, so that the associative controller mantains a queue of transfer cycles which are executed as frame buffer cycles become available.

There is an alternative way to describe the achitecture. There are two memories: the image memory (frame buffer) controlling the screen and the host memory containing the same (or perhaps more complete) pixel data. The image memory is overlaid on the host memory, such that every address in the frame buffer memory also exists in the host memory. In microcomputer architectures, such an arrangement is often referred as shadow memory, where ROM and RAM are overlaid.

The proposed system is shown in more detail in Figure 9. The cache controller monitors the traffic on the system bus and performs an address comparison and a translation from process virtual addresses to physical addresses. An address of a write cycle falling within the RGB arrays is identified and decoded into the primitive components <bank, row, column, pixel>. While the normal write cycle to host memory is

executed the corresponding write request with the decoded information is placed into the request queue. The latency time to the frame buffer depends on the detailed design of the frame buffer. If the access to the frame buffer is only during the horizontal retrace interval, that delay could be up to 50 microseconds. The existence of the cache makes it possible to simplify the frame buffer design and to limit the access to only the retrace intervals without jeopardizing performance.

The proposed architecture results in a redundancy of information: All image data that exist in the frame buffer memory are replicated in the host memory. Therefore, pixel reads are local to the host memory and so are very rapid. The read/modify/write cycle is rapid as the only access to the frame buffer is to write the result. Even this process is placed in a hardware queue, so as not to introduce host wait states.

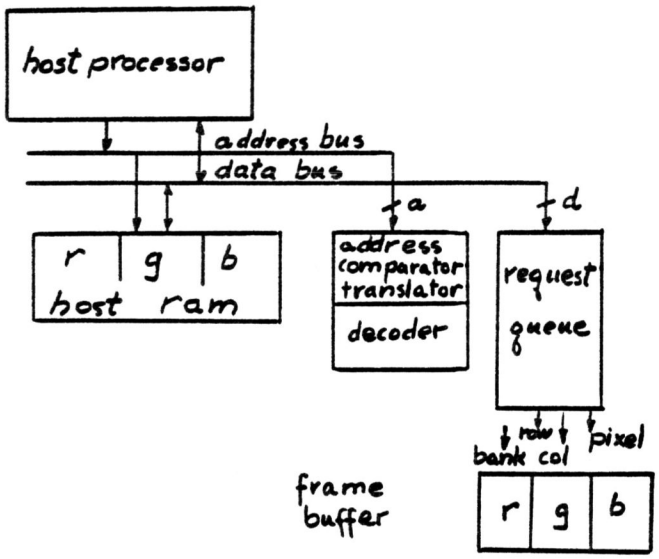

Figure 9. Frame buffer with host memory acting as a cache.

The technique described here allows the memory in the frame buffer to be limited to perhaps 9 or 12 bits per pixel, while the full description of the image, with perhaps 24, 32 or even 48 bits per pixel is available in the host memory at a considerably lower cost. Additional bits in pixel depth are useful for various paint techniques.

5.0 SUMMARY

This paper describes three mechanisms to improve on currently existing computer paint stations. Two are techniques, one being a better tool for colour selection, the other for applying smooth subtle effects to the canvas. The third topic is a design for tailoring a controller/frame-buffer memory system to the requirements and constraints of a paint station.

The paper is very much a working paper in that it points the way to more work to be done. Although the two tools have been implemented, they have not yet been tried out by artists within the frame-work of a full paint station. We have yet to see how these tools will complement other paint system tools once they are in the "hands" of the artists. The memory architecture design is just that, a design. We look forward to its being implemented.

Paint stations present the designer with a fascinating test bed to try out new ideas and techniques. The authors expect to see considerable development over the next few years.

References

1. [Blin76] Blinn J.F., Newell M.E. Texture and reflection in computer generated images. CACM 19(10), Oct. 1976.

2. [Blin78] Blinn J.F. Simulation of wrinkled surfaces. Computer Graphics 12(3), Aug. 1978.

3. [Blin80] Blinn J.F. et al. Voyager 2. JPL Film, available on Siggraph Video Review 1, May 1980.

4. [Blin81] Blinn J.F. et al. Saturn. JPL Film, available on Siggraph Video Review 2, Aug. 1981.

5. [Blin82] Blinn J.F. et al. Mimas/Voyager2. JPL Film, available on Siggraph Video Review 6, Oct. 1982.

6. [Garn70] Garner W.R., Felfoldy G.L. Integrality of stimulus dimensions in various types of information processing. Cognitive Psychology 1, 225-241, 1970.

7. [Garn82] Garner W.R. The analysis of unanalyzed perceptions. in Perceptual

Organization, ed. Kubovy & Pomerantz (L Erlbaum Associates), 1982.

8. [Goet82] Goetz S.M., Beatty J.C., Rasquinha D.J. Colour principles and experience for computer graphics. Proc. GI'82, 313-322.

9. [Hurv82] Hurvich L.M. Color Vision published by Sinauer: Sunderland, Massachusetts, 1982.

10. [Myer68] Myer T.H., Sutherland I.E. On the design of display processors. CACM 11(6), 410-414, June 1968.

11. [Smit78a] Smith A.R. Paint. NYIT Technical Memo 7, July 1978. Reprinted in Two-dimensional Computer Animation Tutorial, Siggraph '82, pg 44-68, July 1982. Also reprinted in Tutorial: Computer Graphics (Beatty J.C., Booth K.S. eds) IEEE EH0194-1, PG 501-515, 1982.

12. [Smit78b] Smith A.R. Color gamut transform pairs. Computer Graphics 12(3), Aug. 1978.

13. [Wein79] Wein M., Burtnyk N, Davis W.A., Norton J. A raster display system for computer graphics and image processing. 6th Man-Computer Communications Conference, May 1979.