A CEL-BASED MODEL FOR PAINT SYSTEMS

Terry M. Higgins and Kellogg S. Booth

Computer Graphics Laboratory, Department of Computer Science
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
Tel: (519) 888-4534, E-Mail: KSBooth%watCGL@Waterloo.CSNet

ABSTRACT

We present a comprehensive formal model for a computer paint system providing capabilites beyond those of traditional designs. The system incorporates an *alpha* channel to enable artwork to have variable opacity in a manner reminiscent of "cel painting." Operations which may be performed on these RGBA images include digital painting, airbrushing, erasing, masking, and image compositing. These are implemented as instances of the digital compositing algebra introduced by Duff and Porter. Our implementation model extends a proposal by Tanner, *et al.* It is cost-effective and is based on the concept of a virtual frame buffer containing a higher-level description of the image being painted and an associated output transformation that maps the contents into a standard RGB frame buffer used only for viewing. Ways of implementing the model to take advantage of multiprocessing capabilities in various host and frame buffer architectures are discussed and three implementations are examined.

Keywords: *brush, cel, digital compositing, mask, multiprocessor, output transformation, paint, virtual frame buffer, RGBA.*

RÉSUMÉ

Nous présentons un modèle formel décrivant un logiciel de palette de couleurs électronique ("paint system" offrant des possibilités allant au delà des modèles traditionnels. Ce système comprend un canal *alpha* qui permet au dessin d'avoir une opacité variable ressemblant à la technique d'animation appelée gouachage de cellos. Les opérations pouvant être exécutées sur ces images RGBA comprennent: dessin digital, airbrushing, effaçage, masquage et composition d'images; ces opérations sont implémentées suivant l'algèbre digital de composition d'ecrit par Duff et Porter. Notre modèle poursuit une idée de Tanner, et autres, et se révèle peu couteux. Il est basé sur le concept d'un "frame buffer" virtuel contenant une description de "haut niveau" du dessin et d'une transformation qui lui est associée. Celle-ci transforme cette description en un format RGB servant excluisvement à l'affichage sur un "frame buffer" ordinaire. Nous discuterons des façons d'implémenter ce modèle suivant les possibilités d'exécution en parallèlle de différents ordinateurs et "frame buffer" Trois implémentations seront analysées.

The first author's current address is: National Film Board of Canada, Studio A. French Animation, Box 6100, Station A, Montreal, Quebec, Canada H3C 3H5, Tel: (514) 283-9309.

INTRODUCTION

The first computer paint system was written soon after the first frame buffer came into existence. A wide variety of styles and techniques have been implemented since then for a diversity of hardware configurations. This paper will introduce a formal model of a paint system based on an artist's conceptual model similar in many respects to traditional cel animation techniques, but extended to capture new degrees of freedom available to animators through the use of digital computers.

The work reported here is part of a joint project of the Computer Graphics Laboratory and the National Film Board of Canada. In consultation with members of the NFB's French Animation Studio in Montreal, the goal is to build a production-quality paint system. The system has been designed and two prototypes have been implemented. Since the french translation for "paint program" is *palette de couleurs electronique*, the system has been dubbed *Palette*.

Palette is intended not only for painting backgrounds but also for *direct animation*. In direct animation, an image or physical model is changed incrementally and re-photographed to create each successive frame [[LAYB79]]. In order to reduce the effort required in this labour-intensive process, *Palette* is designed to combine the direct animation potential of a typical paint program with the advantage of *cel animation*: composite images whose component parts are re-usable to save duplication of effort in those parts of the image that remain constant from frame to frame. In cel animation, this is of course achieved by painting each part of the image on a separate sheet of transparent acetate called a "cel."

Palette operates upon "digital cels," that is, RGBA images [[PORT84]], that may have been painted with the program, digitized from photos or hand-drawn pictures, or produced by other computer graphics rendering techniques. In this respect, its functionality (though by no means its performance) is similar to the *Pixar Compositor* [[LEVI84]].

The notion of a *virtual frame buffer* with an associated *output transformation* is central to the implementation model. Tanner *et al.* [[TANN83a]] have described the speed and potential cost advantages of implementing RGB paint programs using a virtual frame buffer as a cache in host memory separate from the hardware frame store used for viewing the image. Frame buffer values need never be read back to the host and the frame buffer hardware has much less stringent speed and depth requirements. [[TANN83a]] tends to present the concept as a better way of implementing existing applications. The paint system described here is an implementation of that proposal which enjoys the benefits cited, but it demonstrates that another aspect is also important. Separating a "working" description of the image (in a virtual frame buffer) from the viewing of the image (in a display frame buffer) affords the opportunity of extending the model of a paint system well beyond the set of features supported by today's hardware architectures.

The following sections present details on the artist's conceptual model (a brief "user's manual" for *Palette*), the formal model of a virtual frame buffer that instantiates the artist's conceptual model, the techniques used to implement the augmented brushing styles proposed in the conceptual model, and a brief discussion of some implementation issues that arise when the formal model is mapped onto specific graphics hardware (in this case three specific multi-processor configurations that are being used as prototype implementations of *Palette*).

THE ARTIST'S CONCEPTUAL MODEL

This section provides an overview of *Palette* as seen by its intended user, an artist. The workstation layout (Figure 1) consists of a tablet with stylus as the primary input device, a colour monitor on which the image being rendered is previewed, and an alphanumeric terminal with keyboard. The monitor displays the current image and a set of menus for selecting operations. In the prototype, additional commands and parameter specifications are made through the use of the alphanumeric terminal. A more comprehensive tablet-based menu system is planned for the production system.

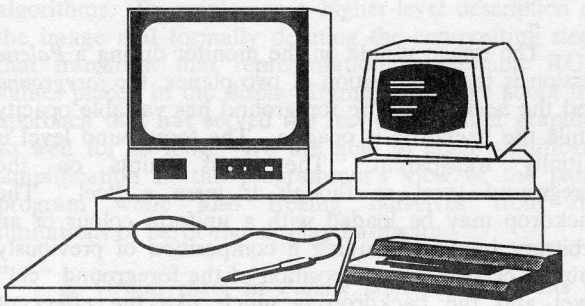

Figure 1. The Workstation Layout for *Palette*.

Central to the conceptual model of *Palette* is the notion that the work surface on which the artist draws or paints is a transparent plane called a *cel*. The term comes from the acetate layers used in animation which were at one time made of "celluloid." In conventional 2 1/2-D cel animation, each frame is created by photographing a stack of cels laid upon opaque background artwork on an animation camera stand [[MADS69]]. Because cels are transparent except for areas where they have been painted, the photographic process results in a *composite* image. Employing this cel concept in a paint program permits creation of images by composition and provides artwork with the

additional property of variable opacity. The former property facilitates not only computer animation but also a digital form of the "layout and paste-up" process which is fundamental to graphic arts. Figure 2 illustrates the artist's conceptul model of painting with *Palette*.

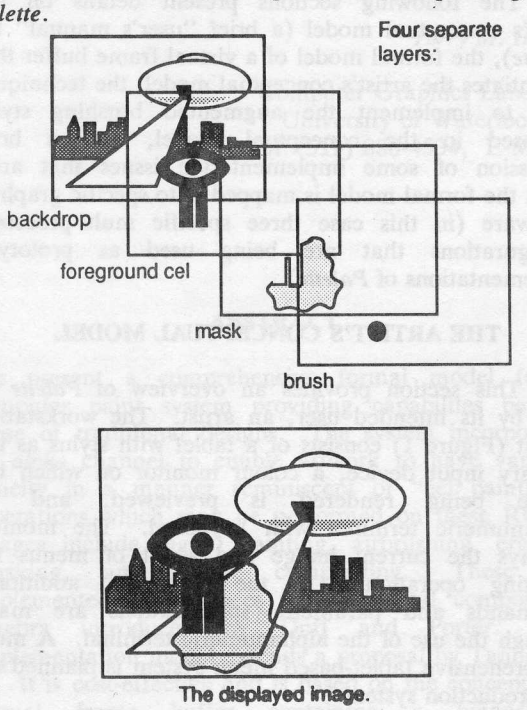

Figure 2. The Artist's Conceptual Model.

The image visible on the monitor during a *Palette* session is the composition of two planes, the *foreground* and the *backdrop*. The foreground has variable opacity while the backdrop is opaque. The foreground level is initially transparent. The artist paints on the foreground level as though it were a "cel." The backdrop may be loaded with a uniform colour or an arbitrary image (including a composition of previously painted cels). The composition of the foreground "cel" level and the backdrop is much like the effect of placing a single cel over a painted background in conventional animation – where the foreground has maximum opacity only it is visible, where the foreground has zero opacity the backdrop is fully visible, and for intermediate opacities the backdrop is partially visible through the foreground.

The prime motivation for the backdrop image is the need for *something* to be visible wherever the foreground is transparent. In addition, while the foreground cel is being painted, the backdrop can be used to hold a reference image for sequence registration (e.g. the previous pose of a cartoon character) or for context (e.g. a background matte painting).

Palette provides only full-colour (24-bit) fully antialiased brushes that have smooth edges and variable opacity determined by brush specifications under the control of the artist. Essentially, a brush is just another variable-opacity raster image, typically smaller than the image being painted. The painting operation itself is a sequence of applications of the brush to the foreground image using one of a variety of compositing formats to blend the two. Each brush has five orthogonal attributes called *shape, stroke, colour, density,* and *operation*. The artist creates his own brush by assigning attributes to each of the five properties. These attributes determine the effect of applying the brush to the foreground image.

The *shape* property refers to the two-dimensional region of pixels (not necessarily connected) affected by a single imprint of the brush. *Palette* provides a variety of standard antialiased square and circle brush shapes automatically. Alternatively, the user may paint an arbitrary shape to be used as a brush.

The *stroke* property determines the relationship between the motion and pressure applied to the stylus and application of the brush to the cel. When the stroke property has the attribute "stamp," each press of the stylus causes a single composition of the brush with the foreground image. The attribute "repeat stamp" causes a succession of brush composites at a constant rate independent of the speed of stylus motion, thus the gap between brush imprints increases with the speed of the stroke. The "continuous" attribute produces a continuous antialiased stroke without the gaps of the "repeat stamp" stroke. The attribute "straightedge" is similar to "continuous" but always produces a straight stroke between positions indicated by momentary presses of the stylus. A sequence of such strokes is terminated by pressing the stylus at (or very near) the same position twice in succession.

The *colour* property is the set of colours, one for each pixel within the brush shape, which is composited with the foreground image. The brush may be a single colour (the same at each pixel within the shape) or multiple colours.

The *density* property is a weighting function that determines the extent to which a single application of the brush alters the image within its imprint. During painting, density represents the opacity at each pixel within the brush shape. A maximum density causes the colour of the brush to entirely overwrite the cel colour, whereas a zero density leaves the cel unchanged. Intermediate values cause a blending of the brush and image colours using the density as a weighting factor. Densities can be created automatically using constant, Gaussian, or cusp functions centered at the origin of the brush. A Gaussian density function, for example, produces an effect closely resembling conventional airbrush.

The *operation* property specifies one of four modes of brushing: *paint, erase, mask*, or *mask-erase*. Paint mode uses the brush density to control the blending of the brush with the image.

Erase mode is a unique feature of *Palette* and reveals part of the power of the underlying cel model. In this mode the colour of the brush is ignored, but the density is used to decrease the opacity of the foreground image each time the brush is applied. Thus, where the brush has maximum density, the foreground image becomes absolutely transparent, exposing the backdrop beneath, whereas repeated application of an intermediate density gradually "fades" artwork so that the backdrop becomes increasingly visible through it. A zero density erase brush leaves the foreground unchanged.

Mask mode is analogous to graphic artists' conventional practice of temporarily *masking* areas of artwork by means of paper, masking tape, or *frisket* to protect them from subsequent painting or airbrushing. Again, the colour of the brush is ignored, but the density of the brush determines the permeability of the *mask* associated with each pixel in the foreground image. A maximum-density brush creates an impenetrable mask. During subsequent painting or erasing the effect of a brush will be reduced according to the degree of mask present at each pixel. Masked areas may be set globally visible or invisible by the artist. If visible, presence of a mask is signified by a specified global mask colour.

Mask-erase mode functions similarly to erase mode, but operates on the mask rather than on the foreground cel. It is used to reduce the permeability of a mask or to do away with it entirely.

Palette provides functions for clearing, loading, and saving the foreground, backdrop, and mask portions of an image, for merging the foreground and backdrop images, and for *undoing* the most recent operation applied to the image. Images are stored in a file format that permits a number of other tools in use at Waterloo to be used on images created with *Palette*. One is a general image manipulation package [[PAET85]] and another is a general compositing package [[KLAS85]] implementing the full set of operations proposed by Duff and Porter [[PORT84]].

This section has provided an overview of functions provided in the current implementation of *Palette*. A more complete description of the artist's view of the final system is contained in the functional specification [[HIGG83]].

THE VIRTUAL FRAME BUFFER

This section discusses the actual implementation of that conceptual model. The key idea which is introduced in *Palette* is the notion of a virtual frame buffer in which the foreground cel and the backdrop, as well as the mask, can be represented. On page 29 of their text [[FOLE82]], Foley and Van Dam note that the traditional design philosophy of paint programs has been that the image being painted is precisely the image being displayed on the monitor – unlike the rest of computer graphics, no other representation of the image is maintained. They term this philosophy "what you see is what you get." This approach has advantages in terms of performance but limits paint programs to functionality easily supported in available hardware. Under this philosophy, if its conceptual model is to be taken seriously, implementation of *Palette* would require a substantial investment in custom hardware. In addition to "on the fly" compositing hardware, the frame buffer would require a substantial number of bit planes in order to store the mask, the foreground cel, and the backdrop.

The model presented here breaks with the time-worn "what you see is what you get" approach to paint program design. It is based on a cost-effective approach that separates the painting function from the viewing function by introducing a *virtual frame buffer* in which all painting operations are performed and an *output transformation* that maps the data in the virtual frame buffer into a form suitable for the video-refresh circuitry in a conventional frame buffer. The virtual frame buffer need not be accessible to the video output (in particular, it need be neither dual-ported nor accessible at standard video rates in excess of 100 or even 25 nanoseconds per pixel). Instead, it can be stored in memory that is more readily accessible to stroke rendering routines (either in cpu memory of the host or workstation or in a portion of the physical frame buffer not required for video refresh). Image data in the virtual frame buffer is not subject to the dictates of the video-refresh circuitry and can be stored in whatever format is most efficient for painting algorithms. By employing a higher-level description of the image and formally defining the compositing steps that transform that representation to viewable RGB values placed in the display frame buffer, we adopt the approach that has served the rest of computer graphics so well for so long. We are able to make significant simplification in the programmer's view of the paint program while also freeing ourselves from the limitations of particular display hardware.

A virtual pixel in *Palette* consists of 64 bits of information (Figure 3), 24 bits for each of the foreground and background images, 8 bits for the foreground opacity, 7 bits for the mask, and one additional *contrast flag* that is used to implement temporary feedback images [[NEWM79]] such as position markers, grid guidelines, and bounding boxes.

Palette's output transformation defines the mapping from these 64 bits to a standard R-G-B representation of each pixel. The first step is to composite the cel and backdrop images according to Wallace's formulation [[WALL81]]. If the artist has indicated that masking is to be visible, the second step is to check whether the pixel's mask value is non-zero and, if so, to composite the global mask colour over the result of the first step using the mask density value as the opacity. To ensure that visible masking does not overly obscure artwork beneath it, the mask density is

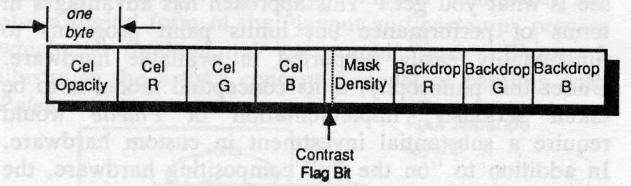

Figure 3. The Virtual Pixel Used in *Palette*.

scaled so that it its opacity never exceeds one half. If the pixel's contrast flag bit is set, the final step is to apply a contrasting function to the R-G-B value to map it to a contrasting value that will distinguish the pixel from its neighbors.

Assuming (R,G,B) is an R-G-B pixel value whose components are each in the normalized range [0,1], the contrasting function usually used in raster graphics is (1,1,1) - (R,G,B) [[NEWM79]]. This is efficiently implemented as complementation of the bit-wise representation for the pixel. This effectively complements the hue of a colour. For cases in which the pixel colour is highly unsaturated and mid-intensity, however, this function produces little apparent change. The worst case is a pixel value of (0.5,0.5,0.5). In the course of formulating our model, Tanner [[TANN83b]] suggested an alternative function to change a colour by a consistent amount regardless of its original value. The function is (R,G,B) + (0.5,0.5,0.5) modulo 1 and is efficiently implemented by complementing only the high-order bit of each component value.

Palette's cels are based upon the full-colour digital representations of cels and backgrounds for animation described by Wallace [[WALL81]]. In addition to R-G-B values, Wallace stores an opacity value which is a compact approximation of the the edge information at each pixel. Wallace's formula for associative pair-wise composition of cels reduces the total number of compositing steps in a sequence of frames by allowing cels that remain adjacent from frame to frame to be pre-merged. Duff and Porter [[PORT84]] introduce a compositing algebra in which Wallace's formula is only one of a dozen operations which go beyond what is possible with traditional cels. They call Wallace's "opacity" values "alpha" values and store their images in a different format in which each of the R-G-B components is pre-multiplied by the alpha value. Duff [[DUFF85]] has extended that model to include a notion of z-depth.

Storing the brush and foreground cel as R-G-B values pre-multiplied by their opacities as recommended by Duff and Porter would require allocating additional bits for each channel in order to avoid severe roundoff error in the course of brush compositing. Our experiments indicate that at least twelve bits would be needed for each of red, green and blue. Rather than substantially increase the storage required in the virtual frame buffer, the cel R-G-B values are retained in their unscaled form.

The reason originally advanced for storing R-G-B as pre-multiplied values is that the compositing operations can be performed more quickly in that format [[PORT84]]. *Palette* uses an alternate formulation of compositing due to Hardtke that realizes the same results with almost the same efficiency, while permitting R, G, B, and opacity to be stored as separate 8-bit values [[HARD85]].

The final issue concerning the virtual frame buffer is the frequency with which the output transformation must be applied. Conceptually, the virtual frame buffer is continually being transformed from its 64-bit internal representation to its 24-bit representation in the physical frame buffer. This is not possible because of the computational bandwidth required. The practical approach is to perform the output transformation at intervals on only those virtual pixels which have been modified. The details of this are very dependent upon the particular architecture upon which *Palette* is implemented. Examples are discussed in Section 5.

BRUSHING TECHNIQUES

The brushes used in *Palette* are relatively complicated objects. For efficiency a *brush record* is maintained that defines the five properties shape, stroke, colour, density, and operation. While stroke and operation are easily preserved as simple scalar values, the shape, colour and density information requires more elaborate data structures. Square and round brushes would be easy to handle, but rather than cater to special cases, brushes are kept in a general linked list data structure whose goal is to save storage and speed brushing by avoiding pixels that are not affected by the brush.

Painting operations within the virtual frame buffer are implemented in a straightforward manner because the foreground cel and the backdrop are stored separately and only the foreground cel or the mask is modified by brushing. To implement the four types of operations, paint, erase, mask, and mask-erase, the brush, the mask, and the cel are treated as images which are combined in various ways using Duff and Porter's compositing algebra. Using the terminology of [[PORT84]], the paint operation is simply the compositing operation "(brush *out* mask) *over* cel." The erase operation is "cel *out* (brush *out* mask)." The mask operation is "brush-density *over* mask" and erase-mask is "mask *out* brush-density." As has already been mentioned, because *Palette* does not pre-multiply the R-G-B components of its RGBA images by "A" (alpha) as in [[PORT84]], it uses a slightly different formulation of these compositing operations [[HIGG86]].

IMPLEMENTATION ISSUES

Practical issues remain to be addressed, such as how and when to update the display frame buffer to reflect modifications to the working copy in the virtual frame buffer. Such questions rely on the particular hardware chosen for the implementation. We thus conclude with a brief overview of three particular architectures on which versions of *Palette* have or will be implemented.

To date, versions of the system have been implemented on two hardware configurations and are planned for a third. The first implementation, a feasibility study, was on a Norpak VDP-1 frame buffer attached via a DMA link to a VAX 11/780 host computer running VMS. A Motorola 68000 microcomputer was attached to the VDP-1 as a dedicated user-programmable display processor. Figure 4 gives a schematic diagram of the system. This equipment is located at the National Research Council of Canada.

In this implementation the entire virtual frame buffer and the undo buffer are located in the host VAX. The VAX is responsible for tablet sampling, all operations on the foreground cel and the backdrop, and the output transformation. The 68000 is responsible for maintaining tracking and writing R-G-B values into the frame buffer. Both processors are programmed in C.

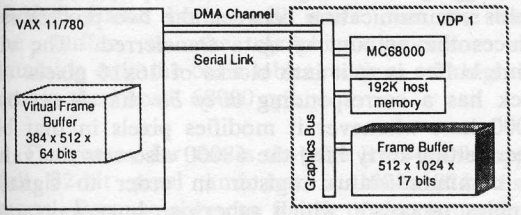

Figure 4. The VAX/VDP-1 Architecture

Because the entire virtual frame buffer is simply an array residing in host memory, many of the bottlenecks associated with traditional host-based paint systems are overcome. Of particular importance is the fact that the VDP-1 is used in a "write-only" manner; the values stored in the hardware frame buffer are never read back during painting. This is fortunate because, like many commercial frame buffers, the VDP-1 does not easily support the operation of read-modify-write on a single pixel. This is the essence of the inner loop of any paint program and it must be made to execute efficiently.

In the VAX/VDP implementation, a stroke is rendered by rubberstamping the brush at each pixel in a straight line between each pair of sampled tablet locations. Fishkin calls this the *Naive* algorithm [FISH84] and notes that the excessive overlap of the brush imprints causes each pixel in the stroke to be "visited" multiple times by the renderer. The bottleneck in this process is data transfer from the VAX to the VDP-1. In order to minimize the amount of data transferred, the VAX performs the output transformation once on pixels in the "wake" of the brush, that is, pixels that the renderer has finished "visiting." For a given brush shape, the eight possible *wake* patterns are pre-computed. Each of these define those pixel positions in one imprint of the brush shape that are unaffected by a second imprint offset from the first by one pixel position. (See Figure 5.) After each imprint of the brush shape into the virtual frame buffer in the course of rendering a stroke, the VAX performs the output transformation on the pixels in the wake pattern corresponding to the direction offset between the current imprint and the one before it. The resulting RGB values, the current imprint position, and a number identifying the wake pattern used are all transferred to the graphics processor which writes the pixel values into the frame buffer at the appropriate positions based on its *own* copy of the specified wake pattern. At the end of the stroke, the full brush shape is used rather than a wake pattern. Figure 5 shows the display buffer updates required for an example stroke. The cross-hatchings indicate the wake pattern that affects each pixel in the stroke.

A noteworthy feature of this implementation is the fact that the VDP-1 at the National Research Council has only seventeen bits per pixel. The actual image displayed on the monitor is generated using the

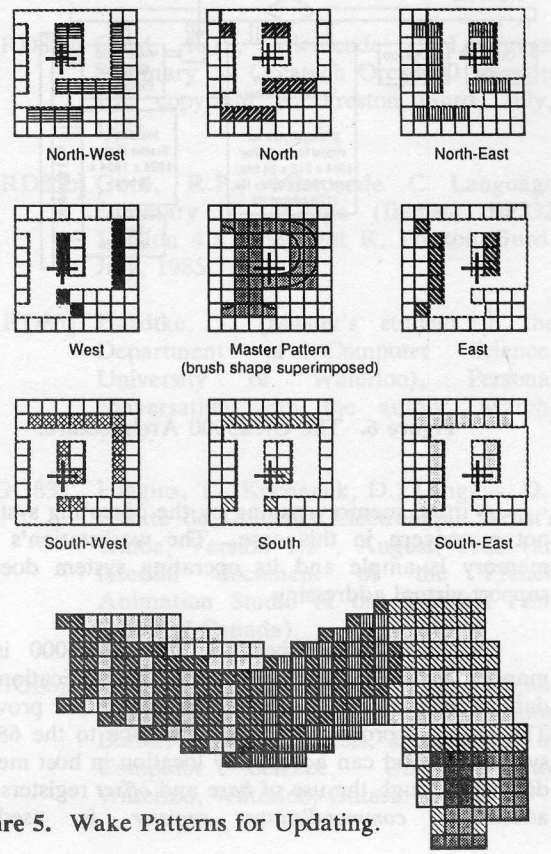

Figure 5. Wake Patterns for Updating.

high five bits of red, the high seven bits of green, and the high four bits of blue. One bit is reserved for tracking feedback. Although this is inadequate for the final image, it is sufficient for viewing the image during its creation. The full-precision version of the image stored in the virtual frame buffer is the useful product.

This first implementation proved the soundness of the artist's conceptual model, but left much to be desired in the way of performance. Reasonable response was precluded by Floating-point compositing code, time-sharing the VAX, virtual memory paging by the operating system, and a 2-millisecond overhead for each system call transferring data to the frame buffer

The second implementation is on an Orca3000 workstation comprising a MC68000 cpu running Unix, a custom bit-slice graphics processor, and a 1024-line 8-bit frame buffer equipped with colour lookup tables [[ORCA83]]. Figure 6 is a schematic diagram of the system. The workstation used for *Palette* has 4 megabytes of host memory. Its frame buffer memory is only addressable by the graphics processor. The graphics processor is programmed in C using a cross-compiler [[GURD85a]].

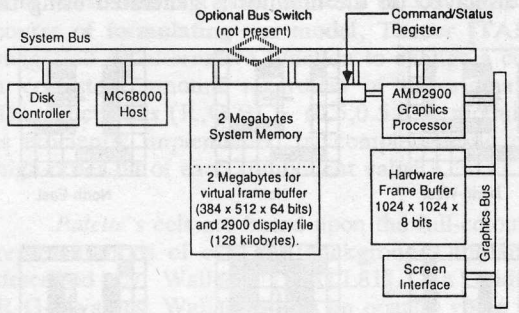

Figure 6. The Orca3000 Architecture.

Virtual memory paging by the operating system is not a concern in this case. The workstation's main memory is ample and its operating system does not support virtual addressing.

An intriguing aspect of the Orca3000 is the manner in which inter-processor communication and data memory for the graphics processor are provided. The graphics processor has an interface to the 68000's system bus and can access any location in host memory directly through the use of *base* and *offset* registers. An additional *command/status register* is used to communicate with the graphics processor without requiring it to generate contention on the 68000 system bus.

Work is more equitably allocated between the two processors in the Orca implementation because the graphics processor assumes the burden of performing the output transformation. Custom microcode written in C performs the output transformation, the tracking function, and menu-handling on the graphics processor. The 68000 host performs all other functions, including maintenance of the virtual frame buffer. Rather than using the Naive algorithm to render strokes, the 68000 uses a more efficient algorithm which visits each pixel only once. This approach uses Gupta and Sproull's antialiased line rendering algorithm [[GUPT81]] to look up appropriate values in Fishkin's "Sweep" arrays which contain pre-convolved opacities for a stroke [[FISH84]].

The "wake" patterns employed in the first implementation of *Palette* are discarded. Frame buffer updates are instead performed by means of a *paging* scheme whereby the virtual frame buffer is divided into rectangular blocks that are marked whenever they are modified. The output transformation is periodically applied to those blocks that have been marked since its last application.

Paging the image to the display frame buffer speeds communication between the two processors and reduces the amount of data transferred. The virtual frame buffer is split into blocks of 16x16 pixels. Each block has a corresponding *dirty bit* that is set by the 68000 host whenever it modifies pixels in that block. After setting dirty bits, the 68000 also sets the value of the command/status register in order to signal the graphics processor which otherwise busy-waits on the register to avoid saturating the 68000's system bus. When alerted, the graphics processor checks the dirty bits to find each block requiring the output transformation. Bits are checked in round-robin fashion to avoid looping on blocks that change frequently to the exclusion of the rest. Before starting the output transformation, the graphics processor resets the block's dirty bit to avoid race conditions with the host. While the 68000 continues painting, the graphics processor accesses the block in host memory directly, performs the output transformation, and writes the resulting pixel values into the display frame buffer.

In the Orca implementation, there is a mismatch in resolution between the 512x512x24-bit image produced by the output transformation described in Section 3 and the 1024x1024x8-bit frame store which must display it. To overcome this problem, an additional step involving *digital halftoning*, is added at the end of the output transformation to map each 24-bit RGB colour to a 2x2 array of 8-bit Orca pixels. In essence three bits of each pixel are allocated to a

halftone version of the red portion of the image, three to the green portion, and two to the blue portion. The halftone patterns are out of phase with each other to avoid *moire* effects and the values in the colourmaps are gamma-corrected for better antialiasing.

Trading spatial-resolution for intensity resolution in this manner is equivalent to an additional two bits in each of the red, green, and blue channels. Thus, the displayed image is effectively 512x512x14 bits. This scheme has worked very well. In side-by-side comparisons with full 24-bit images, differences are difficult to discern at normal viewing distances. Again, a certain amount of discrepancy is acceptable due to the distinction between previewing images during painting and the ultimate resolution required for photographing or video-recording finished artwork. These results support the contention of Tanner, *et al* that the virtual frame buffer approach would permit construction of less expensive "24-bit" painting stations employing hardware frame buffers having fewer than 24 bitplanes.

The third implementation is not yet underway, but is worth considering briefly because it complements the first two approaches. The VAX/VDP-1 implementation performs almost all of the calculations on the host with the frame buffer serving only for viewing. The Orca3000 implementation offloads all of the output transformation to the graphics processor, while maintaining the virtual frame buffer within the host. A proposed implementation for the Adage/Ikonas RDS-3000 frame buffer will move even the virtual frame buffer to the graphics processor while maintaining only the basic tablet routines and high-level control in the "host" 68000 cpu.

The reason for this is that the RDS-3000 supports a full 32-bit pixel and has a 1024x1024 display memory. Because only one fourth of that is needed for the viewing image, the other three-fourths can be used to store the entire virtual frame buffer. A 32-bit custom bit-slice (similar in many respects to Orca3000's 16-bit graphics processor) has sufficient computing power and high-bandwidth access to the display memory that we expect to be able to perform both the basic painting algorithm and the output transformation on the bit-slice without using the 68000 that is attached to the frame buffer.

Because both the 68000 and the bit-slice are programmed in C (the bit-slice has a similar cross-compiler [GURD85b]) we hope to move much of the program from the Orca3000 to the RDS-3000 with little modification.

ACKNOWLEDGEMENTS

The authors wish to thank the National Research Council of Canada for providing access to their VAX/VDP-1 system for the pilot project, to Orcatech for providing the Orca3000 workstation on which the prototype of *Palette* has been implemented, and to the National Film Board of Canada for supporting the first author during this research. Additional funding was provided by the Natural Sciences and Engineering Research Council of Canada. Special thanks are extended to Marceli Wein of NRC for his suggestions and encouragement.

REFERENCES

[DUFF85] Duff, T., "Compositing 3-D Rendered Images," Computer Graphics, Vol. 19, No. 3, July, 1985, pp. 41-44.

[FISH84] Fishkin, K.P., "Algorithms for Brush Movement in Paint Systems," Proceedings of Graphics Interface '84, Ottawa: May 28-June 1, 1984, pp. 9-16.

[FOLE82] Foley, J.D., Van Dam, A., Fundamentals of Interactive Computer Graphics, Addison Wesley, 1982.

[GILO85] Giloth, C., Veeder, J., "The Paint Problem," IEEE Computer Graphics and Applications, Vol. 5, No. 7, July, 1985, pp. 66-75.

[GUPT81] Gupta, S., Sproull, R., "Filtering Edges for Gray Scale Displays," Computer Graphics, Vol. 15, No. 3, July, 1981, pp. 1-5.

[GURD85a] Gurd, R.P., Microcode C Language Summary — Orcatech Orca3000 Version 1.8, copyright R. Preston Gurd, July, 1985.

[GURD85b] Gurd, R.P., Microcode C Language Summary — Adage (Ikonas) BPS32 Version 4.8 copyright R. Preston Gurd, July, 1985.

[HARD85] Hardtke, I. (Master's student in the Department of Computer Science, University of Waterloo), Personal conversation with the author, March, 1985.

[HIGG83] Higgins, T., Kochanek, D., Langlois, D., Palette de Couleurs Electronique Artist's Guide, Version 1.2 , August, 1983 (an internal document of the French Animation Studio of the National Film Board of Canada).

[HIGG86] Higgins, T., Painting a Cel: Digital Painting in a Virtual RGBA Frame Buffer, Master's Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, 1986.

[KLAS85] Klassen, V. (PhD student in the Department of Computer Science, University of Waterloo), Personal conversation with the author, March, 1985.

[LAYB79] Laybourne, K., The Animation Book, Crown Publishers, New York, 1979.

[LEVI84] Levinthal, A., Porter, T., "Chap – A SIMD Graphics Processor," Computer Graphics, Vol. 18, No. 3, July, 1984, pp. 77-82.

[MADS69] Madsen, R., Animated Film: Concepts, Methods, Uses, Interland, New York, 1969.

[NEWM79] Newman, W.M., Sproull, R.F., Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill, 1979.

[ORCA83] Orca3000 Hardware Reference Manual, Issue 2.0, Orcatech Incorporated, Ottawa, Ontario, December, 1983.

[PAET85] Paeth, A., The IM Toolkit – A Comprehensive Raster Manipulation Package Presented through Design and Example, Master's Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, 1985.

[PORT84] Porter, T., Duff, T., "Compositing Digital Images," Computer Graphics, Vol. 18, No. 3, July, 1984, pp. 253-259.

[TANN83a] Tanner, P., Cowan, W., Wein, M., "Colour Selection, Swath Brushes and Memory Architectures for Paint Systems," Proceedings of Graphics Interface '83, Edmonton: May 9-13, 1983, pp. 171-180.

[TANN83b] Tanner, P., Personal conversation with the author, July, 1983.

[WALL81] Wallace, B.A., "Merging and Transformation of Raster Images for Cartoon Animation," Computer Graphics, Vol. 15, No. 3, July, 1981, pp. 253-262.