Interactive Picture Creation Systems

Darlene A. Plebon[1]
Kellogg S. Booth[2]

Abstract

Paint is an interactive program whose design is based on multiple processes and message passing. It allows the user to create technical and artistic pictures on a raster display. We describe its design, implementation, and user interface. A second design for a prototype picture creation system that can be used to create videotex pages is presented. A goal of this design, evolved from Paint, is to present a natural user interface that is easy to learn by novices. A partial implementation of the system is discussed.

This report was originally submitted as a master's thesis to the Mathematics Faculty of the University of Waterloo by the first author. The research reported here was supported in part by the Natural Sciences and Engineering Research Council of Canada, under grants A4307, G0650, G0651, E4291, and a Postgraduate Scholarship, by the University of Waterloo and by the Computer Graphics Laboratory.

[1] Present address: Computer Graphics Laboratory, University of Waterloo, Waterloo, Ontario, N2L 3G1, (519) 886-1351.
[2] Present address: Computer Graphics Laboratory, University of Waterloo, Waterloo, Ontario, N2L 3G1, (519) 886-1351.

Acknowledgements

We would like to thank all those people who provided many helpful comments and other assistance during the development of Paint, especially Rick Beach, Paul Breslin, Doug Dyment, Eugene Fiume, Morven Gentleman, Preston Gurd, Steve MacKay and Baldev Singh. Rick Beach is a co-designer of the current implementation of Paint. Paul Breslin implemented the tablet secretary in the MLCP. Preston Gurd wrote the Microcode C compiler. Eugene Fiume designed and implemented the first version of Paint.

Contents

List of Figures

1. Introduction

A picture creation system is a computer program that facilitates the creation of illustrations that may be included in documents, used to generate slides, or viewed on graphics terminals through an information delivery system such as Telidon. Picture creation systems can be interactive or language driven. This thesis is concerned with interactive picture creation systems, and in particular with information provider systems for producing videotex pages.

Chapter Two discusses the design, implementation, and user interface of Paint, an interactive painting program used both for aesthetic drawing and for technical illustrations. Paint has proven to be an effective tool for slide creation and has served as a testbed for further research on user interface design and information provider systems.

Chapter Three surveys the state of the art in information provider systems. Currently available information provider systems for videotex are discussed. Other interactive illustration systems are also examined because there is much that can be learned by studying them. Language driven picture creation systems are briefly described.

Chapter Four discusses a design and partial implementation for a prototype picture creation system capable of producing pictures in a form suitable for display on a videotex terminal. The goals of this design are to produce a system having the following attributes.

- compatibility with both current and projected Telidon technology
- a natural, easy-to-learn user interface
- facilities for manipulating and formatting text as well as graphics
- full utilization of available hardware and software technology

Chapter Five discusses possible system configurations for such an information provider system based upon the experience gained in the work reported in Chapters Two, Three and Four.

The picture creation system has been prototyped on a powerful, sophisticated, and relatively expensive collection of hardware. The next step is to build a second stand-alone prototype system using low to medium cost hardware. Still further in the future we hope to develop an information provider system viable for commercial use, based on lessons learned with the first two systems.

2. Paint

2.1 Introduction

In this chapter, we describe a painting system, hereafter referred to as Paint, developed in the Computer Graphics Laboratory at the University of Waterloo [7].

Paint has served as a testbed for our research on information provider workstations. We have concentrated on designing the user interface for picture creation in Paint, without concerning ourselves with complicated data structures for picture definition and manipulation. The use of multiple processes and anthropomorphic programming are important concepts in the design of Paint. Much of this work is directly applicable to information provider workstations. Because the implementation is complete we have been able to test our ideas on real users.

A painting program is an interactive system that allows a user (the artist) to create images in a relatively unstructured manner through a simulation of painting as it is done in the real world. The display for a frame buffer system serves as the canvas. A simulated brush is positioned using a pointing device, such as a stylus and tablet or a mouse. The image of the brush can be any two dimensional figure. Different techniques for applying the brush image to a picture in the frame buffer yield various effects. One such painting technique involves copying brush images into the frame buffer.

A number of computer painting systems have been reported in the literature [41,61,62,65]. General features of these programs are surveyed by Alvy Ray Smith [65]. Painting techniques include rubber-stamping copies of the brush, inking the brush along a path indicated by the pointing device, and simulating an air brush. Other functions typically provided in paint programs include the definition of complex brushes, the filling and clearing of areas, saving and restoring of pictures, text annotation, and constrained drawing [65].

Paint is similar to other painting programs in its basic function, providing a useful range of services for both aesthetic drawing and technical illustrations.

2.2 The Environment

Paint is implemented using the programming language Zed [43] on a Honeywell Level 6 minicomputer running the Thoth operating system [20,21]. The Paint workstation consists of an Ikonas RDS-3000 raster graphics system [35,36] for output, a Summagraphics Bit Pad One graphics tablet [70] with either a stylus or four button puck for input, and an auxiliary 80 column by 24 line alphanumeric terminal with a keyboard.

The frame buffer is used at 512 by 512 resolution with 32-bit pixels. The Ikonas system includes four colour lookup tables, a crossbar switch for selectively routing bit planes to the colour map, a DMA interface to the host, and a bit-slice microprocessor. Other important features include multiple write masks on the frame buffer memory, overlay bit planes, and zoom/pan/scroll capabilities [35,36]. Code for the Ikonas bit-slice microprocessor is written in Microcode C, a limited subset of the Unix programming language C for which a compiler has been developed at the University of Waterloo by Preston Gurd [32].

2.3 An Overview of Paint

In this section, we present an overview of the functions and the user interface provided by Paint. We describe the system from the user's point of view here, leaving the implementation details for another section in this chapter.

2.3.1 The Screen Layout

Figure 1 shows the screen layout for Paint. The frame buffer display in Paint is divided into three areas.

- a canvas area for painting, which occupies most of the screen
- a menu area on the right where the user selects options or actions
- a spectrum along the top for colour selection

The tablet is used for painting, for selecting menu items, and for choosing the brush size and colour. An iconic tracker on the screen reflects the position of the puck or stylus on the tablet. Different tracker icons are displayed in each of the three areas on the screen to provide feedback to the user. Figure 2 shows the various tracker icons.

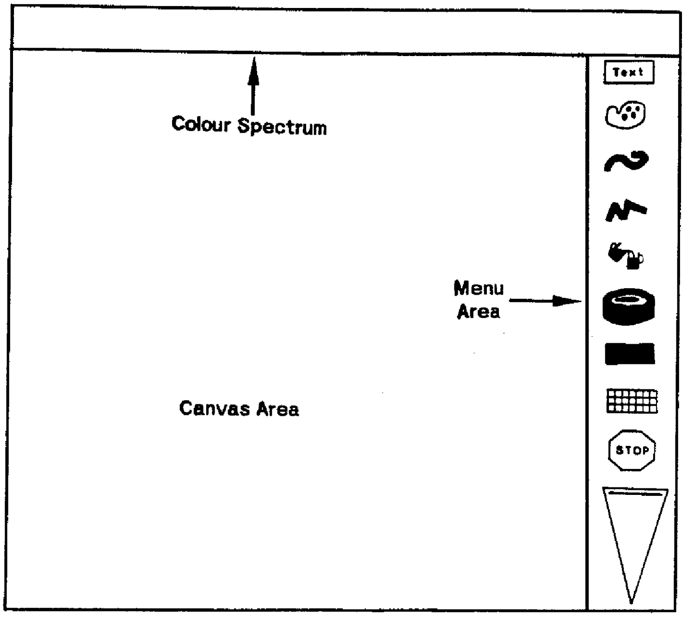
2.3.2 Menu Selection

The menu is split into two areas, the top portion containing iconic menu items, and the bottom portion containing an inverted triangle from which the user selects the brush size (see Figure 1).

A menu item is selected by moving the puck or stylus to position the tracker over the desired item and then depressing and lifting the stylus or pressing and releasing the yellow puck button. An audible beep confirms the selection and an arrow points to the item selected. Usually the tracker icon changes to indicate the new mode of operation. Other response by the program depends on the item selected.

In general, depressing and releasing the yellow button on the four button puck is equivalent to depressing and lifting the stylus. Other buttons on the puck presently have no counterpart on the stylus, but this could be implemented if one is clever. From this point on we will refer only to the puck in our discussion.

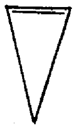
Figure 1. Screen layout for Paint.



2.3.3 Free Hand Brushing

Painting modes are selected from the menu. In freehand drawing mode, holding the yellow button down and moving the tracker on the canvas leaves a trail of brush images, providing a freeform sketching mode.

2.3.4 Brush Selection



In Paint, brushes are square in shape. Arbitrary two dimensional shapes are not available for brushes because we have not yet implemented an interface permitting the user to define his own brush shapes. Also, painting with an arbitrary shape would be somewhat slower because of the increased complexity of the overlap calculations, which enable us to minimize the number of pixels written. The user can at any time change the size of the brush by selecting a new size from the brush triangle. The colour of the brush is selected from the spectrum. In both cases, selection is accomplished by positioning the tracker and depressing and releasing the yellow button on the puck.

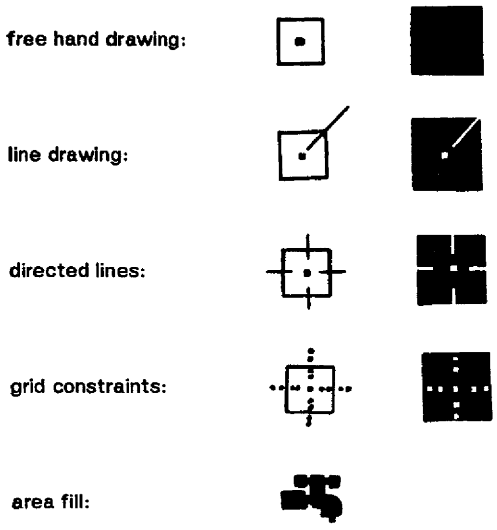


Figure 2(a). Drawing trackers.

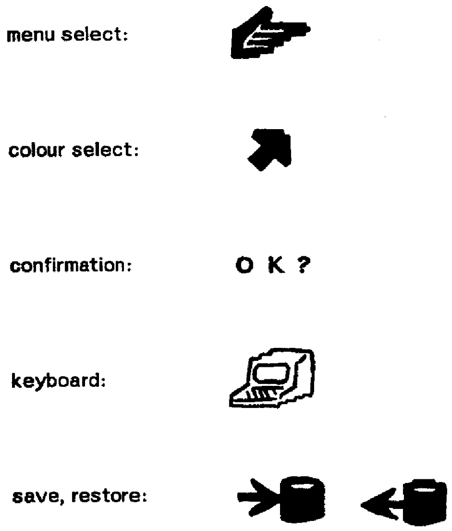
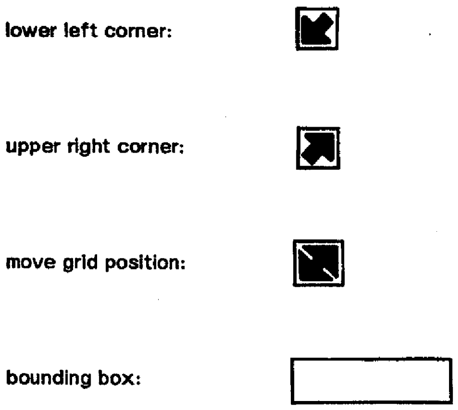


Figure 2(b). Menu trackers.

Figure 2(c). Grid trackers.



Several types of user feedback are provided regarding the size and colour of the brush. First, the tracker icon always reflects the current brush size, which greatly aids the user in painting (see Figure 2(a)). Second, a copy of the brush in its current colour is always displayed directly below the brush triangle on the menu (see Figure 1). Finally, a line is drawn across the brush triangle at the last selected size, indicating how to choose larger and smaller brushes (see Figure 1).

2.3.5 Line Segments

A second basic drawing mode provides for straight line segments in a connect-the-dots fashion. Each time the yellow button is depressed, a line is drawn using the current brush, connecting the new position with the old. This is more of a drafting action, but still somewhat freeform in that there are no constraints on the directions or endpoints of the lines. The tracker icon changes to indicate straight line rather than freeform drawing by showing a line connected to the icon (see Figure 2(a)). A break in the sequence of line segments occurs when the line segments menu item is reselected, or when the brush size or colour is changed.

2.3.6 Area Fill

Bounded regions can be coloured using the area fill menu item. In fill mode the tracker is a water faucet, indicating that colour is going to be poured onto the screen. Depressing the yellow button selects the seed point for the fill algorithm, an adaptation of Alvy Ray Smith's Basic Fill [66], which searches the bounded region for pixels of the same colour as the seed point, changing them to the new colour. A feature of our implementation is that a fill operation can be aborted simply by hitting any button on the puck. This allows the user to limit the damage caused by an incorrect seed point selection or by a leak, which will occur if the fill region is not bounded.

2.3.7 Colour Palette

The colour palette menu item provides an alternate way for selecting colour, permitting a shade already used on the canvas to be matched exactly. When the colour palette item is selected, the tracker changes to an arrow informing the user to select a colour from anywhere on the screen by pointing at it and depressing the yellow button. Paint then returns to the mode in use prior to the colour selection.

2.3.8 Clear Screen

The entire screen can be cleared to the currently selected colour using the clear screen menu item. Since this is a very destructive action, a confirmation must be made when the okay icon appears as the tracker. If confirmation by depressing the yellow button is not given within a few seconds, or if the tracker is moved out of the menu area, the okay icon disappears and menu selection resumes.

2.3.9 File Save and Restore

The disk menu item provides a means for saving and restoring pictures. When this menu item is selected, a picture of a CRT with a keyboard is displayed, directing the user's attention to the auxiliary terminal. The user is queried on the auxiliary terminal for details. He responds to a series of questions by typing on the keyboard. This is one of the few times the keyboard is used in Paint.

The user is first queried for an operation: save, restore, inventory, or cancel. For save and restore, the user is queried for the name of a picture file. He is then informed whether the file exists and is asked to confirm the action. For an inventory operation, he is presented with a list of the current picture files and queried for another operation.

As a file is being saved or restored, a disk and arrow icon (see Figure 2(b)) travels down the menu indicating the progress of the save or restore.

2.3.10 Text

Text

The text option uses a menu on the auxiliary screen to control the text composition parameters, such as typefont, size, and horizontal and vertical justification of text. Figure 3 shows the layout of the text menu on the auxiliary screen. Arrows on the screen indicate the options currently in effect. The tablet controls the auxiliary screen tracker for menu selection on the auxiliary terminal. To avoid confusion while the user is working on the auxiliary screen, there is no tracker visible on the main display.

Figure 3. Text menu layout.

```
----------------------------------+
  TEXT JUSTIFICATION MODES        |     TEXT STRING TO ANNOTATE
----------------------------------+     +------------------------
            |                     |     |Hello
            V                     |     +------------------------
    @-------@-------@    TOP      |
    |TTTTTTT|       |             |     TYPEFACE SELECTIONS
    |   T   |       |             |
    |   T   | y   y |             |     Bedford_12
 -->@---T---@-y---y-@    CENTRE   |     Bedford_16
    |   T   | y   y |             |     Bedford_32
    @---T---@-yyyyy-@    BASELINE |     Helvetica_12
    |       |     y |             |     Helvetica_16
    @-------@--yyyy-@    BOTTOM   |  -->Helvetica_24
                                  |     Helvetica_32
   LEFT   CENTRE   RIGHT          |     Bask_Bold_12
                                  |     Bask_Bold_16
----------------------------------+     Bask_Bold_24
    +--------------V--------------+     Bask_Bold_32
    |  GREY LEVELS: 5, 4, 3, 2, 1 |
    +-----------------------------+
    +---------------+
    |    RETURN     |
    +---------------+
```

The user may type text or change the composition parameters at any time in the text mode. When the user completes a text string by typing a carriage return, the focus of attention returns to the frame buffer display where the tracker represents the size of the area in which the text will be displayed. This is positioned with the tablet, like everything else. Depressing the yellow button writes the text into the picture. This action can be repeated as many times as desired.

With grid constraints on (described below in more detail) the tracker moves in discrete jumps determined by the grid spacing and text is placed at the nearest grid point. The justification parameters come into effect in this mode permitting the user to left, center, or right justify the text string horizontally on the grid, and to top, center, baseline, or bottom justify the text string vertically on the grid. This allows the user to easily create lists or otherwise format pieces of text.

2.3.11 Quitting

The stop menu item terminates Paint, if it is subsequently confirmed by depressing the yellow button when the okay icon appears.

2.3.12 Constrained Drawing

The constrained drawing functions allow one to introduce some regularity into a drawing. Horizontal and vertical lines, regular shapes such as triangles, rectangles, or hexagons, and lines of specific lengths are easily created. These capabilities are essential for producing technical illustrations.

Constraint parameters are controlled from a menu displayed on the auxiliary terminal. Figure 4 shows the constraints menu, which contains items for:

- defining the type of constraints to be applied: directional, grid (and directional), or none
- defining the directions to be constrained: horizontal and vertical; horizontal, vertical, and diagonal; or horizontal, vertical, 30 degree, and 60 degree
- defining whether or not the grid is visible
- changing the grid resolution and position
- selecting a default grid: 8x8 or 16x16
- returning to the main display

Figure 4. Constraints menu layout.

```
+--------------------+      +--------------------+
|     DIRECTIONS     |      |    CONSTRAINTS     |
+--------------------+      +--------------------+
|      Hor/Ver       |      |        None        |
|   Hor/Ver/Diag     |      |    Directional     |
|   Hor/Ver/30/60    |      |        Grid        |
+--------------------+      +--------------------+

+--------------------+      +--------------------+
|     GRID DEFN      |      |    GRID VISIBLE    |
+--------------------+      +--------------------+
|       8 x 8        |      |         No         |
|      16 x 16       |      |        Yes         |
|       Scale        |      +--------------------+
|     Translate      |
+--------------------+      +--------------------+
                            |       RETURN       |
                            +--------------------+
```

The grid constraints facility can be thought of as electronic graph paper, analogous to the user drawing on tracing paper placed over graph paper, but with the advantage that no skill is needed to follow the lines. The system does it all.

Brushing in free hand mode with horizontal and vertical grid constraints creates straight horizontal and vertical lines whose endpoints lie on the grid. Even though the puck may wander, Paint continues to draw in a straight line until a drastic change in direction is detected. Diagonal lines are also permitted at the user's option.

To constrain only the direction of the strokes and not their endpoints, one uses directional constraints. When the user brushes in free hand mode with directional constraints on, once the yellow button is depressed and the puck has been moved enough to determine the drawing direction, Paint continues to draw in that direction as long as the yellow button remains depressed. The user must lift his finger from the button before he can choose a new direction.

Directional and grid constraints work in a similar manner during the straight line segment (connect-the-dots) mode.

Because it is often desirable to bounce in and out of the constrained modes during the creation of a picture, we provide a means for quickly switching among the three drawing modes (grid constrained, direction constrained, and unconstrained) without accessing the constraints menu on the auxiliary screen. The user cycles through the three modes by repeatedly pushing the blue button on the puck. The tracker icon indicates to the user which mode is in effect. This feature is not available if a stylus is used instead of the puck.

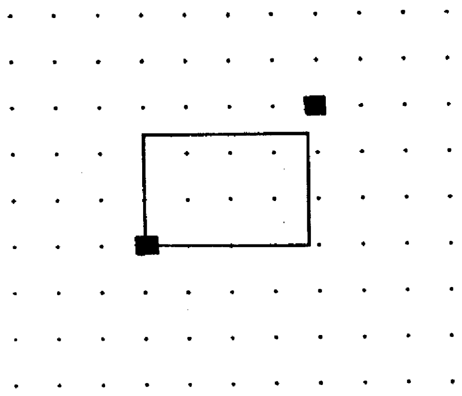
Only lattice points are displayed for the grid when it is visible. These are overlaid on the canvas area and are not part of the picture.

To change the grid resolution, the user selects the scale item on the constraints menu. Scaling of the grid is most easily accomplished when the grid is visible. The user is permitted to interactively modify the grid on the main display in the following manner. The user selects two grid points. A brush image is overlaid on the picture at each of the chosen points, providing user feedback. If the two points define a vertical or horizontal line, the grid is only scaled in the direction of the line. Otherwise, it is scaled in both the horizontal and vertical directions, although it need not be scaled the same along both axes. After the second point is selected, the tracker becomes a rubber band box with one corner anchored at the first point and the diagonally opposite corner attached to the puck (see Figure 5). When the yellow button is hit again, the grid is rescaled so that there are the same number of grid divisions in the new box defined by the first point and the newly selected point as there were in the original box defined by the first and second points. The second point is erased and a brush is overlaid on the picture at the new point. The user may continue the scaling operation by selecting either of the two displayed points to be the moving corner of the rubber band box. Alternatively, he can translate the grid by a selection near the middle of the box defined by the two points. In this case, the tracker icon becomes a rectangle whose corners will be grid lattice points when the user again hits the yellow button to cause translation.

The translate item on the constraints menu allows the user to reposition the grid without changing its resolution. When selected, the user is presented with a rectangular tracker icon on the frame buffer screen. When the yellow button is depressed, the grid is translated so that the corners of the tracker are grid lattice points.

The grid operations described in the preceding paragraphs are rather complicated. We make use of many different tracker icons to provide cues to the user as to what is being performed (see Figure 2(c)). Tracker icons in the constrained modes are the free hand and connect-the-dots icons modified to indicate the directions being constrained and the type of the constraints (see Figure 2(a)). At any time the user can exit a complicated action, such as grid

Figure 5. A grid scaling operation.



scaling or translation, simply by selecting another item from the main menu. Hopefully the user will never be stuck in an unknown mode because there is always an easy escape route.

2.3.13 Detailed Work

Two of the buttons on the puck provide a zoom capability, which facilitates detailed work. Depressing the white button zooms on the area where the tracker is positioned, using hardware pixel replication. Depressing the white button again increases the zoom factor. The green button undoes all zoom, restoring the original canvas. Hitting the green button a second time returns to the current zoom level in a single action, thus allowing the user to toggle in and out for menu selection.

The hardware zoom capability does not increase the resolution of the picture. However, it does give the user a magnified view of the pixel structure, which is very useful in making single pixel corrections. Because we use a software tracker in Paint rather than the hardware cursor on the Ikonas frame buffer system, the tracker is also magnified when the picture is zoomed. Thus, the tracker always indicates what pixels will be modified if a brush image is dropped at the current position.

This hardware zoom feature has the unfortunate problem of often zooming the menu off the screen. We have not found this to be too troublesome because the green button permits one to zoom out, do the selection, and zoom back to the same spot at the cost of two extra button pushes. We do not loose important feedback by zooming the menu off the screen because the tracker icon indicates the current mode and brush size. However, zooming the menu off the screen does have one serious drawback. The zoom cannot be implemented using menu selection. This unfortunately means that the zoom feature is not available if a stylus is used instead of the four button puck because zoom requires two additional buttons (other than the selection button) and the stylus only has one button (stylus "up" or stylus "down"). If the stylus were the only available device, an alternative would be needed; if necessary, zooming could be accomplished using the auxiliary keyboard.

2.4 Process Structuring in Paint

Most paint programs use interrupt-driven tablet procedures for painting algorithms with a looping policy that repetitively tests for user input actions. Overly clever code is often necessary to provide efficient implementations of these algorithms because many actions must be interleaved in a complex pattern to successfully achieve interactivity.

Paint systems are complex programs that combine computer graphics algorithms with real-time interaction. Concurrent activities include tracking the paint brush, providing timely user feedback, canvas painting, and user control of those actions. This property of many asynchronous concurrent activities makes a paint program well suited to a multiple-process design. Our implementation of Paint exploits the multiple-process capabilities of Thoth.

2.4.1 The Thoth Operating System

The architecture of an operating system greatly influences the design (or existence) of the multiple-process programs built upon it [30]. In this section we describe the operating system used for the development of Paint. A basic understanding of the process and message passing primitives is necessary to appreciate their influence on the design of multiple-process programs.

Thoth is a portable message-based real-time operating system [20,21] developed at the University of Waterloo. It was designed to meet real-time constraints and to be portable across a variety of 16-bit computer systems. Thoth has been ported to the Texas Instruments TI990, Data General Nova, Modcomp IV, Honeywell Level 6, Advanced Micro Devices AM16 (Z8000-based), and Motorola 68000 computers. Portability was achieved using Zed [43] (a high-level BCPL-derivative implementation language), a portable compiler, and a software development system [19].

Message passing in Thoth is accomplished using three primitives: a blocking send, a blocking receive, and a non-blocking reply. The syntax for these primitives is as follows:

```
destination_id= .Send( message, id );

requestor_id= .Receive( message; id );

sender_id= .Reply( message, id );
```

A process is identified in a message communication by a unique process identifier assigned by the system when the process is created. A message can be received from any process, rather than a specific one, by omitting the id argument to .Receive. All messages in Thoth have a fixed length of eight words.

Message communication between two processes is accomplished as follows. The first (sender) process sends a message to the second (receiver) process. The sender becomes blocked until the receiver replies to it. Messages sent to a process are queued until that process performs a receive. When a process does a receive, it is passed the first message in its queue; if no messages are queued, it blocks until some process performs a send to it. When the processing required by a message is completed, the receiver replies to the sender, unblocking it and allowing it to continue on its way.

Create, ready, and destroy primitives permit the dynamic creation and destruction of processes. Process management in Thoth is designed to be relatively inexpensive, encouraging the use of many small processes with short lifetimes, being created and destroyed as needed. The process primitives are relatively cheap; the cost of a message communication (send-receive-reply) between two processes is about one millisecond.

Another means for communication between processes in Thoth is via shared data. This mechanism is attractive for problems involving several processes operating on shared data structures where the processes perform little computation in comparison to the amount of data accessed [20]. A team is "a set of processes that share the same address space, memory free list, code segments, and data segments", providing "a means of associating a set of closely interacting processes in terms of data and code sharing" [20, p. 25]. There are no semaphores or monitors in Thoth. Although processes in a team do share memory, the memory is never used for synchronization. Process synchronization is accomplished entirely by passing messages. In fact, all synchronization of memory access is accomplished through message passing.

Processes are executed in pseudo-parallel by the scheduler. A process switch occurs when a process of higher priority than the active (executing) process becomes ready (unblocked) or the active process blocks due to a message communication or by awaiting an interrupt. A higher priority process can be readied either by an interrupt or by the active process. The scheduler maintains a queue of ready processes within each priority level. The processes in each ready queue are ordered by the time at which they became ready, the most recently readied process being last in the queue. When a process switch occurs, the scheduler causes the first process on the highest priority non-empty ready queue to be executed.

Two types of process scheduling are available in Thoth, a natural-break scheme and a priority scheme. If all processes are created with identical process priorities, then natural-break scheduling will occur.

Natural break scheduling allows the program designer to determine how to time-share the processor among the various processes. This scheme requires careful consideration of the time necessary to complete an algorithm. Breaks are created to provide a voluntary relinquishing of the processor. Normally these break points occur at the end of an iteration or at some other significant points in the algorithm, hence the term "natural break". In Thoth, scheduling decisions are made each time a process blocks on a message primitive, with the blocked process guaranteed to be placed at the end of the ready queue when it is eventually unblocked, giving other processes an opportunity to run. Thus, a voluntary relinquish can be accomplished with a superfluous message send. A sponge process is employed to accept these messages as quickly as possible. This sponge process is a repeat-forever receive-reply loop:

```
Natural_break()
  {
    repeat
      {
        sender_id = .Receive( message );
        .Reply( message, sender_id );
      }
  }
```

Natural break scheduling is sufficient for simple algorithms, but becomes an annoying and sometimes impractical scheme for more complicated algorithms.

In contrast, the priority scheduling scheme requires that a priority ranking be imposed on the processes. Highest priority processes will execute, when ready for execution, before lower priority processes. Response time can be guaranteed in interactive programs by arranging that processes that control interactive devices have the highest priority, followed by user interface processes, and finally by processes that execute algorithms requiring large amounts of computation.

2.4.2 Anthropomorphic Design

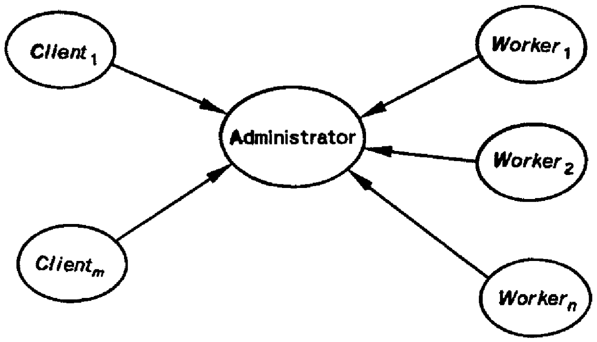
Anthropomorphic design [10,23] is an important concept in the structure of Paint. Job classifications and reporting roles in a large corporate hierarchy provide the model for the organization of Paint. Each process in Paint is assigned a role and communication channels are created between processes whose roles require cooperation. The processes in Paint are modelled after the concepts of administrators, workers, secretaries, agents, and overseers.

The anthropomorphic technique of endowing processes with human characteristics has been used in several previous projects. In Reynold's Actor/Scriptor Animation System, message-passing actor processes "are the embodiment of imaginary players in a simulated movie" [56]. Kahn and Hewitt model the entities on a display screen as a society of interacting people [37]. In Kahn's Director program [37], artist procedures are first told what to do and then go about performing those actions independently. Similar concepts exist in Smalltalk [31,38]. Many of these ideas can be traced to Hewitt's earlier work at MIT on the actor formalism [33].

2.4.3 The Administrator Process Concept

An administrator process maintains a resource and relies on worker processes to manipulate that resource [30]. Worker processes relieve the administrator process of much of its computational burden, allowing it to service requests as quickly as possible. Each worker process performs some computation, then sends a message to its administrator indicating the results and requesting more work.

Figure 6. Process diagram for an administrator process with several worker processes and several client processes.

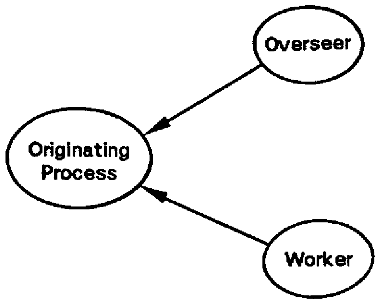


The administrator assigns a job to a worker by replying to the request. A work request is queued by the administrator process until a worker is available to process it. Workers may also be queued by the administrator waiting for work to do. The administrator process receives messages from its worker processes and from client processes, which request information or service related to the resource. The general coding structure of an administrator is an infinite loop:

```
Administrator()
  {
    repeat
      {
        requestor_id = .Receive( message );
        select( message[TYPE] )
          {
            case FROM_CLIENT:
              {
                if( info_available )
                      Supply_info_and_reply( message, requestor_id );
                else
                  {
                    if( worker_queued )
                          Supply_work_and_reply( message, remembered_worker_id );
                    else
                          Queue_work_for_later( message, requestor_id );
                    Remember_client_for_reply_later( message, requestor_id );
                  }
              }
            case FROM_WORKER:
              {
                Update_info( message );
                if( reply_to_client_necessary )
                      Supply_info_and_reply( message, remembered_client_id );
                if( work_queued )
                      Supply_work_and_reply( message, requestor_id );
                else
                      Queue_worker( message, requestor_id );
              }
          }
      }
  }
```

The process diagram in Figure 6 shows the relationship between an administrator process and its worker and client processes. Arrows drawn between processes show the directions messages are sent. An administrator should never be send-blocked; it must be able to process requests as soon as they occur. This characteristic results in an administrator process having only incoming arrows in a process diagram.

Figure 7. Process diagram for an overseer process and its associated worker processes.



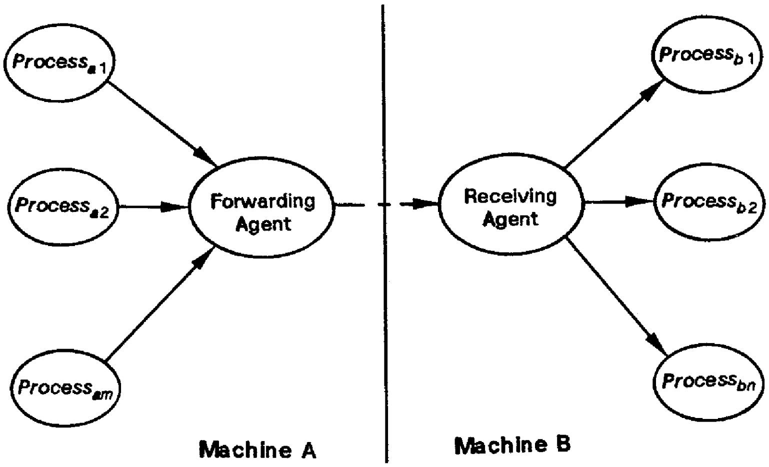
2.4.4 The Overseer Process Concept

An overseer process performs two functions. It relieves a worker process from the distractions of interruptions, and it acts as a watchdog in case the worker process runs out of control [7]. The worker process is programmed only to perform a single function. Typically it embodies an algorithm for a distinct task. The overseer process executes concurrently with the worker, looking for changes in the worker's work orders.

Figure 7 illustrates the relationship between an originating process and the worker and overseer processes that it creates. The worker and overseer processes may interact with other processes, although this is not shown in the diagram. Both the overseer and worker processes report their results by sending messages to the originating process.

2.4.5 The Agent Process Concept

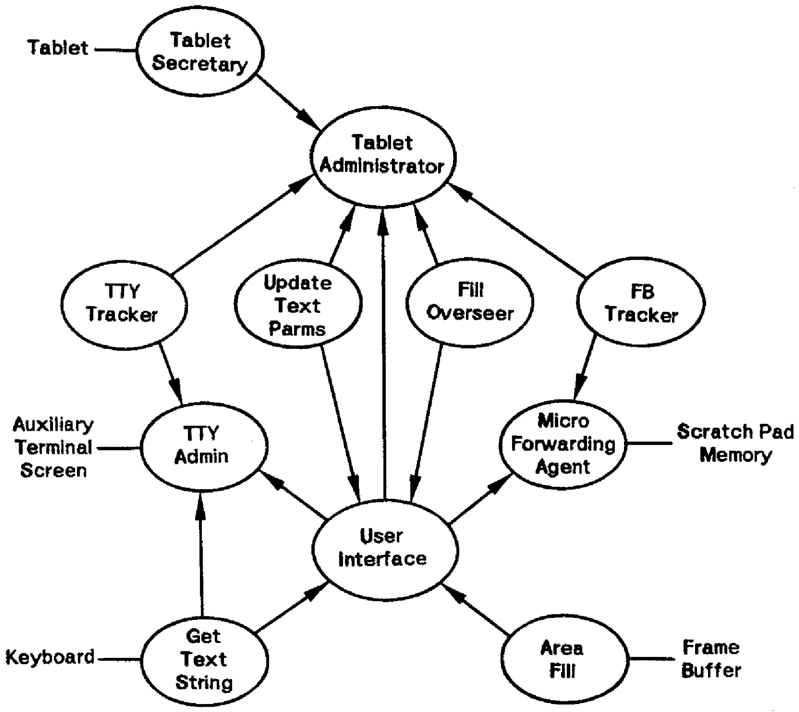
Figure 8. Process diagram for agent processes providing a communication channel between processes executing on two separate machines.



Cheriton defines an agent as "a process whose function is to act as a representative of other machines, and to use the resources of its machine on their behalf" [20, p.58]. An agent isolates the communication (one-way) between two machines to a single process on each machine, thereby simplifying the interfaces between the machines. An agent interacts with other processes on its machine using the standard process communication mechanisms, so the remainder of the system does not know about the existence of the external resources. The process diagram in Figure 8 illustrates communication between two machines using agents.

2.4.6 The Design of Paint

Figure 9. Process diagram for Paint.



Several process abstractions are used in Paint: hardware and resource administration processes, algorithmic worker processes, forwarding agent processes, and user interface control processes. Administrator processes support the graphics tablet and the auxiliary terminal screen. Worker processes implement various painting techniques, such as drawing a brush stroke and performing area fill. Forwarding agents provide communication channels to the two microprocessors used by Paint while user interface processes provide control over the actions of Paint.

Processes which perform user-directed actions can be transient, continuously running, or blocked waiting for a work order. The frame buffer tracker process providing iconic feedback is an example of a continuously running process. It is transient, however, in the sense that it is killed off when the user cannot interact on the frame buffer screen, attention having been diverted to the auxiliary screen. It is recreated when interaction is again permitted with the frame buffer.

The process diagram in Figure 9 shows an overview of Paint in its entirety. Processes are identified as nodes with arrows indicating message sends between processes. Hardware devices can be thought of as special processes which reply to requests sent by software processes, thus providing a uniform framework for thinking about all processes.

2.4.7 Tablet Administrator Process

Some type of graphics pointing device is needed in an interactive painting program to indicate the position of the paint brush. The pointing device used in our implementation is a graphics tablet. The tablet is used for a number of functions: positioning the paint brush, selecting menu items, specifying the seed point for an area fill, and aborting a fill operation in progress.

The tablet administrator manages the graphics tablet. It accepts several types of messages:

- update the current tablet coordinates
- request the current tablet coordinates
- request the next tablet coordinates different from the current set
- request the coordinates for the next tablet sample in which a button is depressed
- request the coordinates for the next tablet sample in which a change occurs in the button state
- request the coordinates for the next tablet sample with changed coordinates or changed button state
- change the sampling mode of the tablet

The following pseudo-code illustrates the inter-process communication of the tablet administrator:

```
Tablet_administrator()
  {
    repeat
      {
        requestor_id = .Receive( message );
        select( message[TYPE] )
          {
            case UPDATE_COORDINATES:
              {
                Update_info( message );
                .Reply( message, requestor_id );
                if( request_queued && tablet_hit_occurred )
                    Supply_info_and_reply( message, remembered_id );
              }
            case REQUEST_COORDINATES:
                Supply_info_and_reply( message, requestor_id );
            case WAIT_FOR_SOME_TYPE_OF_HIT:
                Queue_request_for_later( message, requestor_id );
          }
      }
  }
```

The tablet secretary supplies the tablet administrator with updated coordinates. In order to continue servicing other types of requests when a particular request, such as one for coordinates with a changed button state, cannot be satisfied immediately, the tablet administrator buffers a single request of each type until a reply is possible. It is up to the program designer to ensure that two processes do not simultaneously make requests of the same type. This simple buffering scheme could be generalized to a full-fledged queuing mechanism, but we do not need it. Typically, there is a cursor tracker process and one other client process requesting tablet input simultaneously. We avoid conflicts by arranging that the tracker processes use a request type not used by other processes.

2.4.8 Tablet Secretary Process

The tablet secretary's duty is to filter the tablet coordinates for the tablet administrator. The tablet secretary provides the tablet administrator with a view of the "ideal" tablet.

The Summagraphics Bid Pad One tablet has several modes of operation, the most useful one of which is to sample the tablet at a constant rate providing a continuous stream of coordinates to the host [70]. This is not an ideal situation from the user program's point of view. The stream contains a lot of useless, redundant coordinates. The tablet secretary eliminates these redundant coordinates, minimizing the expense of handling them.

Small deviations in the tablet coordinates often occur due to the sensitivity of the tablet rather than the motion of the tablet puck. This causes such disturbing effects as a tracker that bounces between two adjacent pixel positions on the screen when the puck is being held perfectly still. The tablet secretary smooths this jitter by eliminating coordinates whose change from the previous set falls below a threshold value.

The tablet secretary can also eliminate large jumps in the tablet coordinates caused by problems with the electrical biasing of the tablet. These glitches, if not dealt with, cause problems in painting. It is worth noting that the Bit Pad, like almost all commercially available tablets of which we are aware, suffers in this respect. It seems to be an unfortunate fact of life that programmers must correct for the mistakes of the hardware designers.

Finally, the programmer can specify what rectangular portion of the tablet is to be used for input. The tablet secretary filters out all coordinates outside this viewport.

Part of the tablet secretary process implemented for Paint resides in firmware for the MLCP microprocessor (Multiline Communications Processor) [34], which serves as a front end for the Honeywell Level 6. The MLCP bundles input so that there is one interrupt per set of coordinates, eliminating redundant coordinates, thereby vastly reducing the number of interrupts and the number of messages being passed. The remainder of the tablet secretary resides in a host process.

By performing all of the above functions, the tablet secretary provides the tablet administrator with a more reasonable stream of input than the raw data sent by the tablet itself. We also achieve some device independence through the tablet secretary. Replacing the tablet with some other pointing device, such as a mouse or simply a different brand of tablet, would merely require rewriting the code for the tablet secretary process to transform the device input into our virtual tablet input.

2.4.9 Tracker Processes

The iconic trackers provide very important feedback to the user. In order to guarantee responsive behaviour, a free running process is created to track the tablet by positioning a tracker (cursor) on the screen. A tracking process is created whenever the user is permitted to interact with one of the screens. At other times, such as when a picture is being saved or restored, the tracking process is not needed and is destroyed. There are in fact two separate tracking processes, one for tracking the frame buffer screen cursor, and another for tracking the auxiliary screen cursor. Because the user is permitted to interact with only one of the two screens at any given time, at most one of these two processes exists at one time.

For the frame buffer cursor, the tracker process decides which tracker icon to display, depending on which of the three portions of the screen the tracker is located in. What icon is used in a particular part of the screen is determined from a global variable that can be manipulated by the various user interface processes.

The frame buffer tracker process is a simple loop:

```
Fb_cursor_tracker()
  {
    repeat
      {
        message[TYPE] = REQUEST_FOR_CHANGED_COORDINATES;
        .Send( message, Tablet_administrator );
        if( In_canvas_area( message[X], message[Y] ) )
            cursor_icon = Canvas_icon;
        else if( In_menu_area( message[X], message[Y] ) )
            cursor_icon = Menu_icon;
        else if( In_spectrum_area( message[X], message[Y] ) )
            cursor_icon = Spectrum_icon;
        Position_fb_cursor( cursor_icon, message[X], message[Y] );
      }
  }
```

The request for changed coordinates ensures that as soon as the pointing device is repositioned, the tracker will also be repositioned. The auxiliary screen tracker process is similar to the frame buffer tracker, except that it does not need to determine what icon to use. The use of an iconic tracker on the auxiliary screen would require keeping a copy of the auxiliary screen internally and implementing the tracker in host software. We are satisfied to use the terminal's hardware cursor instead.

2.4.10 Fill Processes

The area fill process is a straightforward adaptation of the Basic Fill algorithm described by Alvy Ray Smith [66]. Given a seed point within a bounded region, the algorithm colours all the pixels in the interior of the region. It is a scanline algorithm that maintains a stack of seed points. The algorithm fills the scanline to the left and right of a seed point and scans the neighbouring lines (one above and one below), stacking seed points for the regions eligible for filling. The algorithm proceeds thusly popping seed points as their scanline regions are filled. The algorithm terminates as soon as the stack is emptied.

Smith's paper presents the details of the algorithm in a much simpler form than he actually implemented [67]. Presumably this is because the details necessary to synchronize all the actions in a complete program would overwhelm the reader. The multiple process approach avoids any complications in the handling of concurrent activities, so the algorithm could be implemented directly as published, vastly improving our understanding of the program.

An important concern is user control of the algorithm when it begins to "bleed", or fill the region outside the expected area, due to a break in the boundary. An abort capability is needed. Therefore, if the user hits a puck button during an area fill, the fill is aborted immediately, allowing the user to minimize the damage caused by a leak. Rather than have the fill algorithm periodically check for an abort request, the multiple process program structure provides an elegant alternative using an overseer process.

When the user selects a seed point, Paint's user interface process creates two new processes: an area fill process, which executes the basic area fill algorithm, and a fill overseer, whose responsibility is to detect a panic signal from the user. The fill overseer, once created, sends a message to the tablet administrator requesting the next tablet hit and awaits a reply, allowing the area fill process full rein. Normally, the area fill runs to completion, sends a message to the user interface, and awaits destruction. If, on the other hand, the fill overseer is replied to with a button hit message, it in turn sends a message to Paint's user interface, and awaits destruction. This can be viewed as a race between the two processes: the first one to complete sends a message to the user interface, which is blocked awaiting a message from either. When the user interface receives a message, it destroys both processes. Either the area was completely filled or the operation was aborted, but it makes no difference to the user interface. In both cases, it performs the same cleanup operations. The usual problems of race conditions do not exist.

The abort mechanism is available in most implementations of fill. The point being made here is that the multiple-process structure permits a simpler, more elegant implementation.

2.4.11 Ikonas Microprocessor Forwarding Agent

A number of activities in Paint have migrated from the host, a Honeywell Level 6, to the custom bit-slice microprocessor embedded in the Ikonas frame buffer system. The multiple-process design made this migration of processes easy to accomplish.

The tracker and draw brush processes have migrated to the Ikonas microprocessor as microprograms written in Microcode C [32]. Because Zed and C are both BCPL derivatives, conversions between these two languages are straightforward. Messages containing brush or tracker definitions and positioning coordinates are implemented as data transfers to the Ikonas scratch pad memory, which is accessible to both the host and the microprocessor.

Following our anthropomorphic design methodology, a forwarding agent process remains on the host to accept Thoth messages and coordinate the synchronization of the data transfers to a receiving agent within the microprocessor. A Thoth process accomplishes a send to a process in the Ikonas microprocessor by sending a message to the forwarding agent process. One word of scratch pad memory is used to synchronize message transfers between the two machines. When the receiving agent in the Ikonas microprocessor is prepared to receive a message, it writes the value MICRO_IDLE into the synchronization word and then awaits a change in the value of that word. When the forwarding agent receives a message from a Thoth process, it waits for the synchronization word to contain MICRO_IDLE, then copies the message into the message buffer in the scratch pad memory, and writes the message type into the synchronization word, thus accomplishing the send. Having does this, the forwarding agent immediately replies to the Thoth process. When the Ikonas receiving agent detects a change in the value of the synchronization word, it copies pertinent information out of the message buffer, accomplishing the receive, and then writes the value MICRO_IDLE into the synchronization word, accomplishing a reply and indicating that the next message can be transferred. The receiving agent then processes the message appropriately while the next message is being transferred.

The migration of code to the microprocessor brought about vast performance improvements in Paint. Firstly, the microprocessor has much faster access to the Ikonas bus than does the host. Secondly, true parallelism exists between processes executing within the host and those executing within the microprocessor, although processes within the microprocessor execute sequentially.

Because the microprocessor has much quicker access to the frame buffer memory than does the host, we have moved display activities, such as the dynamic rubber band box, grid display and erasure, and clearing of the canvas area, to the Ikonas microprocessor. Some of these features would otherwise be unusable because users would not tolerate the long delays that would result.

The area fill process is the next candidate for migration into microcode. The same overseer concept used with multiple processes will be used to oversee operation of the microprocessor. Thus, we are confident that the anthropomorphic design methodology provides a reliable control and synchronization scheme for multiple processes executing in multiple processors. Migration corresponds to hiring outside contractors instead of using different branches of the same organization.

2.4.12 User Interface Processes

The user interface implemented in Paint uses both the frame buffer display and an auxiliary terminal screen for menu information. Several processes may interact with the auxiliary screen, simultaneously attempting to display information. If they are allowed to write directly to the screen, the output from the various processes can be arbitrarily interleaved as two concurrent processes execute in parallel. The interleaved messages can be very difficult for the user to interpret, as was discovered during development. This problem is alleviated by directing all output for the auxiliary screen through a terminal screen administrator process that synchronizes the message display. Messages are displayed sequentially by the terminal screen administrator one by one as they are received.

At times, it is convenient to allow the user to enter input from both the tablet and the keyboard concurrently. One such case in Paint is the specification of a text annotation that is to be placed into the picture. The user controls display parameters by menu selection using the tablet and enters a text string using the keyboard. We permit the user to perform these two operations concurrently. This provides a simple, flexible user interface that allows the user to change his mind regarding the display parameters, even in the midst of typing the string.

Programming both sources of input is difficult on most operating systems because each input request causes the program to wait for an interrupt. In our multiple-process implementation, this natural user interface has been provided simply and elegantly using small processes with short lifetimes. Two processes are created: the Get_text_string process, which monitors the text string entry on the keyboard, and the Update_text_parameters process, which monitors the menu item selection on the tablet. These processes send menu update and text string messages to the user interface control process. When the user indicates completion by termination of the string with a carriage return or by menu selection of the return item, the user interface destroys both text processes and continues on its way. These two transient processes merely gather input and do no processing themselves. This guarantees consistency in the state of the world when they are destroyed.

2.4.13 Impact of the Multiple Process Design

We have experimented with both types of process scheduling in Paint. We are able to provide reasonable response and interaction using either scheme, although natural break scheduling involves more overhead due to periodic voluntary relinquishment of the processor. We are currently using the priority scheduling technique in Paint. We prefer this scheme because it is not necessary for the programmer to introduce breaks in worker processes to reschedule the processor. By setting the process priorities appropriately, we can guarantee real-time interaction. The highest priorities are assigned to the processes involved in user feedback: tablet, cursor, and microprocessor forwarding agent processes. The middle priorities are assigned to user interface control processes, such as the fill overseer and auxiliary terminal screen administrator. The lowest priorities are assigned to worker processes, such as the area fill process. This guarantees that user feedback is provided as quickly as possible, interaction is provided as soon as the feedback is completed, and painting algorithms are allowed to execute when no interaction requests are pending.

The use of multiple processes greatly simplifies the design of an interactive program such as Paint. The anthropomorphic design methodology allows the programmer to design multiple-process programs in a simple, natural, effective way. The multiple-process implementation allows user control of painting actions in a simple and elegant way. The implementation of the fill abort mechanism in Paint illustrates the effectiveness of such a multiple-process design. The use of multiple processes to handle multiple input devices is another example of the simplifying benefits of this design. Dynamic creation and destruction of processes allows easy enabling and disabling of devices.

The availability of many, cheap, dynamically created processes has greatly facilitated the design and implementation of Paint. The ability to handle concurrent processing easily has proven to be a major advance over more traditional sequential techniques.

2.5 User Interface Design Considerations

A major emphasis of our work has been on providing a good user interface in Paint. The user interface can make or break a program. An otherwise good, efficient program will be rejected by the users if its user interface is poor. This is especially true for highly interactive programs, such as Paint. Thus, the significant effort invested in the user interface of Paint should ultimately pay off.

Much experimentation has been done throughout the development of Paint's user interface. We have gone through many iterations in the implementation, testing our ideas on users. Many changes have been made as a result of suggestions and criticisms received from users. We ran two Paint contests, one very early in the development of Paint, and the second just recently, a full year after the first. The contests were successful; they provided us with much feedback from people with a variety of backgrounds, allowing us to evaluate the program's effectiveness. The contests also provided us with a substantial collection of pictures for show. Paint has been used to prepare slides for several talks (about 150 slides in all), again providing us with feedback on Paint as a technical illustrator.

A major goal in our user interface design is that Paint be "instantly" usable. The program must not require a significant training period. In the following sections, we discuss how we achieved our goal to provide a simple, flexible, user friendly, easy-to-learn user interface.

We have attempted to follow several principles in the design of Paint's user interface to realize our goals:

- provide timely user feedback
- ensure that the program state is always visible on the screen
- input by pointing rather than input by typing
- keep it simple. Alan Kay's maxim is "simple things should be simple; complex things should be possible" [68]
- be consistent
- avoid modes whenever possible

2.5.1 User Feedback

It is imperative to provide good feedback to avoid the user becoming confused or panic stricken [28]. The feedback must be immediate. We guarantee immediate feedback in Paint by programming the handling of input devices and feedback as separate processes that run at higher priority than any others.

Our main vehicle for user feedback is the iconic tracker on the frame buffer display. The tracker icon displayed at any given time depends on the state of the program and the position of the tracker on the screen. This concept is in use elsewhere. Newswhole [3,74] and many of Xerox's systems including the 8010 Star Information System [68], Okra [48], Draw [4,5], and Markup [49,50] use a variety of tracker shapes to indicate the state of the system. We make use of the iconic tracker to provide cues to the user as to what he is doing or should do next. Changing the tracker icon is an excellent way to convey information to the user. The tracker is displayed at the center of attention and it moves, so any modifications to it will catch the user's eye very quickly. Visual continuity is maintained. The user does not have to look elsewhere on the screen or to another screen to determine the state of the program. Our extensive use of carefully chosen iconic trackers has eliminated the need to provide state information on an auxiliary terminal. Training of new users becomes remarkably easy using this method of feedback.

Tracker definitions are 32 by 32 pixels in size with one bit of depth. We have implemented the trackers in microcode software, rather than using the Ikonas hardware cursor. The tracker is implemented as an overlay. It has one of two colours depending on the value of the pixel it is overlaid on. Although we originally implemented an exclusive-or tracker, we no longer use the exclusive-or tracker for several reasons. Firstly, the tracker can get lost on certain colours because the exclusive-or sometimes yields a very similar colour. Secondly, it interferes visually with the structure of the picture. Thirdly, it is more difficult and more costly to implement. The only problem we see with an overlay tracker is that one has to move the tracker after dropping a brush image to see the change in the picture.

The tracker icons are carefully designed to convey maximal information to the user, without becoming so large or detailed as to interfere with the picture or cause poor response. Typically, a tracker icon is 16 by 16 pixels or smaller in size. Indeed, one of the most difficult tasks was the design of the trackers. The water faucet for fill conveys the idea of pouring colour onto the screen. A pointing finger is used for menu selection. A notched arrow head indicates a colour selection. The bounding box tracker for a text annotation identifies the exact size and location of the text. Confirmations are requested using the okay tracker icon.

In painting actions, the tracker tries to communicate all relevant information concerning what will happen when the yellow button is depressed. The tracker reflects the current paint brush, indicating to the user exactly which pixels will be modified if a brush image is dropped into the picture at the current position. Because we implemented the trackers in microcode software, this property still holds true when the picture is zoomed for detailed work. This has proven to be a great aid for the user. For small brushes, we place a box outline around the brush to make the tracker visible. Various indicators may be exclusive-ored with the brush image to show the drawing mode in effect. For example, after the first point has been selected in connect-the-dots mode, a diagonal line protrudes from the tracker to distinguish it from freeform mode. The painting tracker icons also indicate what sort of constraints are being applied at any given time. When the user is attempting to do detailed work with a small brush, the tracker may visually interfere with the picture. We try to avoid this problem by not cluttering the area nearest to the center of a painting tracker with detail. In all, we have more than a dozen distinct painting tracker icons (see Figure 2).

Various other forms of feedback are provided as well. Extensive feedback regarding the brush is provided in the vicinity of the brush triangle. While this feedback, especially the brush size feedback, is somewhat redundant, it is very helpful during brush selection operations because it appears near the focus of attention and because the menu tracker does not provide any brush feedback. The previous selection bar on the brush triangle is particularly useful for making slight modifications to the brush size, for example, to match a previous brush. It was added for precisely this reason.

An audible beep on the auxiliary terminal reassures the user that a hit has occurred and his request is being processed, be it a menu selection, a colour selection, or a fill abort. This form of feedback catches the user's attention without disturbing his visual focus of attention. We have been careful not to overload this feedback mechanism because a continuously beeping terminal is very annoying and not very informative.

An arrow points to the current menu selection. Arrows are also used on the grid and text menus to indicate the parameters currently in effect. During file saves and restores, a disk and arrow move down the menu, tracking the progress of the operation, providing a placebo similar in effect to Tilbrook's Buddha [3,74]. When attention should be focused on the auxiliary terminal, a picture of a terminal is displayed on the frame buffer menu.

All of these methods combine to provide complete and up-to-date information about the state of Paint.

2.5.2 Input Techniques

Input is provided to Paint using a graphics tablet for pointing and a keyboard for text entry. We discuss below how each device is used and the alternatives available.

We try to make as little use of the keyboard as possible in Paint. There are numerous reasons for avoiding the keyboard. For many individuals, typing is not one of their better skills. Pointing is a more natural action than typing. Switching from one device to another is distracting. One looses visual continuity as one searches for the correct positioning of fingers on the keyboard or gropes for the stylus that was laid down somewhere five minutes ago. Even when we resort to using the auxiliary terminal screen, we still use the tablet for menu selection. The keyboard is only used for entering text when annotating a picture, and for entering the pathnames of files being saved or restored.

The tablet is the primary input device in Paint, being used for everything from positioning the paint brush on the screen to entering commands by menu selection. A tracker on the screen indicates the position of the puck or stylus on the tablet. Absolute rather than relative positioning is performed. That is, there is a one-to-one mapping between the screen and the tablet. Actions are specified by menu selection or by using the buttons on the puck as function keys.

Although we use a tablet in Paint, some other pointing device, such as a mouse, would work adequately. A tablet supplies absolute coordinates to the host, whereas a mouse supplies relative coordinates. The tablet can be used in mouse mode [25], but a mouse does not provide a corresponding absolute position mode. Thus, a mouse is not suitable for functions such as tracing. As soon as the mouse is lifted, the origin is lost. We believe, as do some other researchers [25], that a tablet is a more flexible input device than a mouse. A disadvantage of the tablet is that one is restricted to a fairly limited area for input (14 by 14 inches on a Summagraphics Bit Pad One, for example). Ideally, the tablet would be an entire desktop.

There is an ongoing battle among users about whether a stylus (pen) or a puck is better for use on a tablet. Many users, the artistic ones in particular, maintain that the stylus has a more natural feel. However, the stylus does possess some physical problems with the cord and with pen-down detection when held at an angle. Some of the advantages of a puck are that it stays where it is left when one is not touching it, it doesn't have to be picked up like a stylus, and it has several buttons on top (rather than just one), which can be used to provide a variety of functions. These characteristics also hold true for mice [68]. For these reasons, the puck is preferred over the stylus by most users. The split is not as simple as artistic versus non-artistic. Some artists like the multiple buttons on the puck. Jane Veeder points out, "[some] artists like to parallel process too" [77].

We prefer four button pucks over thirteen button pucks for several reasons. The four button puck has large, easy to locate buttons, unlike the thirteen button puck with its small, difficult to locate buttons. The capacity of short-term memory is very limited. Thirteen things are too many to remember at once, as has been shown in a number of tests [45].

A very difficult user interface problem is the allocation of buttons to functions to be performed. There are numerous actions that we would like to occur at the push of a button on the puck, but there are only four buttons. We must keep in mind our objectives of simplicity, consistency, and instant usability when considering solutions to this problem.

The first solution involves simultaneously depressing a number of the four buttons to transmit any of sixteen unique codes to the host. A debouncing scheme would be required since one would invariably press one of the buttons before the rest. With the layout of buttons on our pucks, pressing several buttons simultaneously does not feel natural and is, in fact, quite difficult to do. Trying to remember combinations of buttons is even worse than remembering single buttons.

Another means of extending the use of the four buttons is to make a sequence of button hits represent a command. The host could determine the end of a command sequence by timing or by waiting until it recognizes a valid command sequence, assuming that no valid sequence prefixes another. A means for cancelling a partially entered sequence is needed. Even short sequences can be difficult to remember and easy to foul up. Confusion will result if the computer and the user do not have the same idea of when a command is complete. This might be of considerable concern for novice users.

Yet another means of extending the functionality of the buttons is to assign different meanings to a button depending on the state of the program and position of the cursor.

These extensions violate our goals of simplicity, consistency, and instant usability. They are a great strain on the user's short-term memory. It can be difficult to remember which button combination does what where.

We have chosen to avoid overloading the semantics of the buttons. We use the four buttons in a simple, consistent manner throughout Paint, at the cost of extra keystrokes at times. We use the yellow button for all selection operations including painting (the selection of locations at which to copy brush images into the picture). The white button always increases the zoom factor, while the green button provides a zoom toggle. The blue button changes the constraints. We have made available what we consider to be the four most necessary functions on the puck. In fact, some users are willing to give up all but the selection function in order to use a stylus.

On the Xerox Star, the temptation to overload the semantics of the buttons was eliminated by eliminating buttons [68]. After testing several one, two, and three button mouse designs on naive users, Xerox decided upon a two button design that everyone found simple and trouble-free to use. Smith et al. point out several morals to this story, one of which is that

> What is simplest along one dimension (e.g. number of buttons) is not necessarily conceptually simplest for users; in particular, minimizing the number of keystrokes may not make a system easier to use [68, p. 276].

2.5.3 Menus

Paint started out life with one fixed iconic menu permanently displayed at the right edge of the frame buffer display, and a colour spectrum along the top. The portions of the screen where the menu and colour spectrum reside cannot be used for painting. A feature of our iconic menu is that it was painted using Paint. Modifications are easy; one just repaints the menu. The menu has gone through many iterations, just like the rest of Paint. The menu icons were carefully designed, like the tracker icons, to convey the important characteristics of the operations they invoke. Novice users learn the meanings of the icons very quickly. Many factors affected the design of the menu. An important concern, which was not realized until we tried to make a videotape, is how the menu looks after NTSC encoding. We reworked our menu design considerably to make it look good on NTSC, removing or widening single pixel lines and avoiding large phase shifts in colour. To our delight, we found that the less complex menu items actually enhanced visibility on the higher resolution RGB monitor.

As more options were added, the menu became rather full. Enlarging the menu was an unacceptable solution because this would reduce the size of the canvas area for painting. To avoid cluttering the display any more, we implemented additional menus on the auxiliary terminal screen for the grid and text options.

Our experimentation with using two screens simultaneously has taught us an important lesson. Using two display screens is not a good idea. Turning one's attention away from where most of the activity is going on to a second screen for brief periods of time is very distracting. Visual continuity is lost. A scheme in which additional menus pop up on the main screen for brief periods of time when needed, such as is done in Stone's Griffin [69] and Singh's Benesh editor [63], would be better.

2.5.4 Modeless Interaction

Larry Tesler defines a mode in an interactive computer system as "a state of the user interface that lasts for a period of time, is not associated with any particular object, and has no role other than to place an interpretation on operator input" [68].

Modes can be confusing and dangerous. They can be as troublesome for an experienced user as for a novice. There are numerous stories about the user who enters some input expecting one result, but achieves an entirely different effect, sometimes a very destructive one, because he is not in the mode he thinks he is in. Smith et al. relate such an incident about Bravo, a highly modal editor [68]. Tesler has eliminated modes from editing in Smalltalk [72].

Modes are not necessarily bad. Some modes simplify the specification of commands. We do make use of modes in Paint. The modes in Paint work for several reasons. Firstly, they are visible, like they are on the Xerox Star [68]. The tracker always indicates the state of the program, changing shape when the user switches modes (recall Figure 2). The command menu is always active, so one can escape a mode simply by choosing another command. Thus, the user never gets stuck in a mode. We avoid modes when they are not necessary.

Many actions, such as zoom, colour selection from the spectrum, and brush selection, are modeless and can be executed no matter what mode the program is in. These features allow even novice users to understand and work with modes easily.

2.6 Frame Buffer Support for the Techniques Used in Paint

Some of the techniques implemented in Paint are only made possible through the use of advanced hardware features of the Ikonas RDS-3000 frame buffer system [35,36]. In this section, we discuss the frame buffer support for these techniques.

The colour lookup tables, crossbar switch, overlay option, and multiple write masks combine to provide much flexibility for the way in which the Ikonas frame buffer memory can be used. The custom bit-slice microprocessor has very fast access to the Ikonas memory, allowing large complicated updates to be done in real time.

The three features of the Ikonas frame buffer that we exploit in Paint are the overlay option, the write masks, and the fast microprocessor. We store the picture being painted in eight bits, which we run to each of the red, green, and blue colour tables, and use the colour table in a normal manner to provide a colour palette of 256 colours. We use the crossbar switch to select and restrict which bits of the frame buffer are to be read out to the display.

The write mask permits us to write to only particular bits within a pixel. These selective writes allow us to partition the frame buffer into planes that can be written independently of one another. The Ikonas has eight write masks on its memory boards, permitting multiple processors to write to different planes simultaneously.

The overlay option allows one to mix two images on video output. If any of the eight overlay bits for a pixel is non-zero going into the colour lookup table, the overlay bits are mapped through the red, green, and blue portions of the overlay colour table, rather than mapping the red, green, and blue data bits through the normal colour table.

The overlay option allows us to implement, simply and efficiently, a software tracker that is not a part of the picture being painted. We do not have to concern ourselves with the location of the tracker in memory when we update the picture because it is stored in a different set of bits, which can be protected using the write mask. We have implemented the tracker display in the microprocessor for speed. We allocate one bit plane in the overlay bits for the tracker. We display a tracker by writing it into that bit plane. To erase it from the screen, we write zeros into that bit plane where the tracker used to be. We achieve a dynamic tracker by erasing the tracker from its previous location and displaying it at its new location each time it is moved.

Prior to acquiring the overlay option for our system, we exclusive-ored the tracker with the picture in the RGB data bits for the frame buffer because we did not want fewer than eight bits of depth for the picture itself. Whenever we updated the picture (painted a brush), we had to check for intersection with the tracker and erase the tracker by exclusive-oring, performing the update, and then exclusive-oring the tracker back in. Painting was significantly slower than it is now and the cursor flickered as painting occurred.

We overlay other sorts of information on the picture at times. The grid lattice points are displayed in an overlay plane separate from the tracker overlay plane. A third overlay plane is used for displaying reference points during a grid scaling operation.

The write mask permits writing to a particular overlay plane or the picture memory without affecting other portions of the frame buffer. The overlay colour map is loaded so that the tracker plane has priority over the grid lattice plane, which has priority over the grid scaling plane.

Some operations have been implemented in the Ikonas microprocessor for speed. These include the tracker display, painting brush strokes, turning on and off the grid, and a rubber band box facility. Without the microprocessor, these operations would be too slow and the program would be unusable.

3. State of the Art in Information Provider Systems

3.1 Introduction

In this chapter, we survey the currently available information provider systems. The systems we discuss fall into two categories: page creation systems designed for videotex (Telidon [13,53] or AT&T's Presentation Level Protocol [1]), and other illustration systems.

We choose not to limit our study to videotex page creation systems alone because many systems whose function is to create illustrations have been developed over the years. Although not oriented specifically towards the videotex environment, many of these systems share the information provider system's goal to facilitate the creation of rich and complex pictures having concise definitions. Indeed, some of these systems accomplish this goal better than the videotex page creation systems currently available and there is much that can be learned by studying them.

3.2 Videotex Page Creation Systems

There are several videotex page creation systems on the market. In Canada, these systems are more commonly referred to as information provider (IP) terminals. The two primary manufacturers of page creation systems for Telidon are Norpak with the IPS-2 and the new GC 1000, and Northern Telecom with VIPS. Cableshare, marketing the Picture Painter system, is a third Canadian company producing IP terminals. In the United States, AT&T has announced a Frame Creation Terminal (FCT) for videotex.

Norpak's IPS-2 uses a DEC LSI 11/03 CPU with dual floppy disks, an RGB graphics monitor, a monochrome alphanumeric menu monitor, a customized alphanumeric keyboard, and an optional digitizing tablet [12,58]. The frame buffer used by this system is basically a Telidon terminal equipped with a graphics cursor. The resolution is 256 by 200 by 4 bits. The keyboard has additional function keys and cursor control keys for both monitors. The IPS-2 supports the Canadian Department of Communications (DOC) 699-E specification for Telidon [13]. This system sells for about $25,000 CDN [60] ($17,500 US plus $1,800 US for the tablet [12]).

The IPS-2 includes graphics editing facilities for creating and modifying points, lines, arcs, circles, rectangles, and polygons. Arcs, circles, rectangles, and polygons may be filled or unfilled. Several texture patterns are available both for the outlines and interiors of elements. The IPS-2 includes operations for deleting, moving, copying, scaling, rotating, reflecting, and reordering the primitive elements. All elements enclosed in a user-specified rectangle can be temporarily grouped to undergo these operations together. The user indicates the element to be operated on by pointing at it with a cursor on the screen that can be manipulated using the tablet or cursor control keys on the keyboard. A sketch facility permitting freeform drawing is included, but its use is discouraged because pictures generated using this mode are stored inefficiently. A fairly primitive facility exists for inserting text into pictures. Only a few text sizes are supported and the text is not proportionally spaced. Six colours (three primaries and three secondaries), six shades of grey, black, and white are the colours available for both graphics and text. A primitive file facility permits the storing and recalling of pages from floppy disk.

The IPS-2 is both menu and command driven. That is, commands may be selected from menus or they may be entered on the keyboard. All of the available option and command menu items are permanently displayed on the menu monitor, causing it to be fairly cluttered and confusing. The keyboard must be used for menu selection and command entry. Because the tablet suffers from jitter and spikes (random fluctuations in input values caused by hardware problems), use of the cursor control keys on the keyboard to position the graphics cursor is encouraged. The numerous input methods serve to overwhelm the naive user.

Norpak's new page creation terminal, the GC 1000, is simpler, cheaper (priced at $12,000 CDN versus $25,000 CDN for the IPS-2 [60]), and supports DOC's newer 709 Telidon protocol [53] and AT&T's Presentation Level Protocol (PLP) [1]. The GC 1000 is a general-purpose 6809-based microcomputer with support for both the Basic and Pascal programming languages [60]. Disk storage is provided by two mini-floppies. The GC 1000 has a 4-bit-deep 256 by 212 memory-mapped display with a colour lookup table that permits the simultaneous display of any 16 of 32,000 possible colours. It uses one RGB monitor for all output and the keyboard, which has additional function, colour, and cursor control keys, for input. A tablet is optional.

The GC 1000 provides the same basic page creation facilities as the IPS-2 with additional support for the new Telidon protocol, including the ability to define colour palettes and line widths [52]. The system is completely command driven from the keyboard. Some of the advanced editing features, such as scaling, rotation, and reflection, seem to be missing. The text facilities are still poor with only a few sizes supported and no proportional spacing.

Northern Telecom's VIPS page creation system, designed by Bell-Northern Research (BNR), uses a Z80-based Cromenco System 3 with a monochrome menu monitor, an RGB graphics monitor, a keyboard, and either a capacitance or pressure sensitive tablet [12,58]. The resolution is 256 by 256 by 4 bits. Like the IPS-2, this system supports DOC's 699-E Telidon protocol [13]. The VIPS system sells for $18,000 to $24,000 US.

The page creation capabilities of VIPS are comparable to those of the IPS-2. The system is menu driven and much of the graphics input is done using the tablet. Text is entered using the keyboard. Unlike the Norpak systems, a previously created image is edited by text editing a human readable form of the PDIs that form the picture [58]. Unique to VIPS is its minibase program, which is capable of merging text and graphics from various files to create Telidon pages [58]. Included in the text input file along with the text are composition commands for formatting the text (characters per line, lines per page, etc.) and specifications for the locations where graphics PDIs (stored in separate files) are to appear. This is an important feature because much page creation involves pages of mostly text.

Both Norpak and Northern Telecom are reported to be coming out with an Apple-based information provider terminal [58].

Cableshare's page creation system is called Picture Painter [18]. Its hardware consists of an 8-bit microcomputer with dual floppy disks, or one Winchester and one floppy disk drive, a Telidon RGB monitor, an alphanumeric keyboard, and a graphics tablet. This system supports DOC's 699-E and 709 Telidon protocols [13,53] and AT&T's PLP [1]. This system is less expensive than the Norpak and Northern Telecom systems. There is also support for Picture Painter on DEC VAX and PDP-11 computers.

Picture Painter is completely menu driven from the tablet. A menu layout sheet is fastened to the tablet with the menu along the bottom and a grid over the drawing area as an aid in the creation of graphics images. Once Picture Painter is invoked, the keyboard need not be used again because the menu on the tablet includes items for each character for text entry. The keyboard may be used for text entry if desired. Picture Painter includes much the same facilities for creating and editing images as the Norpak and Northern Telecom systems.

AT&T's new Frame Creation Terminal (FCT) consists of a 16-bit microcomputer, dual 8" disk drives, a colour monitor with 256 by 256 resolution, a keyboard, and a graphics tablet [2]. This system supports AT&T's Presentation Level Protocol [1] and sells for $34,000 US. The FCT's page creation and editing capabilities appear to be comparable to those of the Canadian information provider terminals. The PLP is fully supported including user definable colour palettes and adjustable line widths. The FCT's text capabilities include proportional spacing.

The main problem that these page creation systems share is that they deal with graphics objects at too low a level. The user is forced to manipulate PDIs. This is analogous to assembly language programming. A high level object structure, corresponding to a high level programming language, permitting the definition and manipulation of complex objects is needed.

Other problems with these page creation systems include their very poor text capabilities and their poor user interface design. We discuss the details of these problems in Chapter Four.

3.3 Other Illustration Systems

There are two classes of illustration systems: interactive illustrators and language-driven illustrators. We are primarily interested in interactive systems, but we mention some language-driven systems in this survey for completeness.

Interactive illustration systems fall into two categories: those that maintain high-level concise object definitions, and those such as paint programs that operate on raw frame buffer images such as paint programs. The first class of illustrator is of interest to us because that is the class which videotex page creation systems fall into. Illustrators in the second class are of interest because of their natural user interfaces for picture creation. Borrell provides a survey of twenty-two digital paint systems, including Norpak's IPS-2 and Northern Telecom's VIPS [12]. We discussed a paint program in detail in Chapter Two. In this section, we will restrict our discussion to the first class of illustration systems, those that generate concise object descriptions.

There are many such illustration systems in existence. We will briefly mention a few that have significantly influenced the design described in Chapter Four.

A group at Brown University, headed by Prof. Andries van Dam, is developing a system for the design, development, and presentation of computer-based documents that combine text and graphics on a high resolution raster colour display [26]. This system includes a sophisticated picture layout program that permits the creation and editing of graphics and text objects. The editing capabilities allow an object or group of objects to be deleted, moved, copied, centered between objects, scaled, or recoloured. User definable grids aid in layout placement and perspective drawing. A significant amount of their effort is devoted to a document layout and presentation system that is more sophisticated than the videotex information data base systems we have used.

Xerox PARC has done much research in the area of interactive illustration systems, especially on how to combine text and graphics within documents. Markup [49,50] and Draw [4,5] are two interactive illustration systems designed at Xerox PARC that run on the Alto personal computer [73]. Griffin [69] is an illustration system that runs on a Dorado personal computer. All three permit the creation of pictures to be included in documents. In all three systems objects are described by a set of control points and a set of attributes. The control points are interpolated by either straight lines or spline curves to form the boundary of the object. Attributes define such things as line thickness, colour, and closure. The Xerox 8010 Star Information System is a personal computer designed for office uses such as document creation, data processing, and electronic filing, mailing, and printing [57,68]. Document creation on the Star includes text editing and formatting, graphics editing, mathematical formula editing, and page layout. These Xerox systems are noteworthy for their innovative and effective user interface design.

We next discuss some non-interactive language-driven systems. Beatty and Booth's PICTURE [8] language is a simple two-dimensional illustration language containing a variety of commands for specifying elementary geometric forms. It has very little control structure and has only two data types: number and string. Illustrations produced using PICTURE are merged with the text of a document by REDPP, a postprocessor for the TRIX/RED text formatter. White's Pic [79] is a general-purpose language augmented with graphics facilities based on PICTURE. Implemented as a preprocessor to the C programming language, Pic accepts most C constructs, providing the flow control and data structures that PICTURE lacked. Kernighan's PIC [39] language is a preprocessor for the Unix text formatter TROFF, providing facilities for drawing illustrations in documents. The block-structured declarative graphics language IDEAL by Van Wyk [76] is yet another illustration language allowing one to describe two-dimensional figures that can be typeset along with text by a document preparation system.

4. A Prototype Picture Creation System

4.1 Introduction

In this chapter, we describe the design and a partial implementation of a prototype picture creation system capable of producing pictures in a form suitable for display on a Telidon terminal or similar device.

We summarize our efforts as "Paint Meets Data Structures and Telidon". In the design, we have tried to retain much of Paint's natural user interface and artistic flavour. At the same time, we provide a high-level hierarchical data structure to define objects within a picture. Videotex output, in the form of Picture Description Instructions (PDIs) for Telidon [13,53] or AT&T's Presentation Level Protocol [1], is easily generated from this data structure through a post-processing step.

We use the same development system for the implementation of our prototype picture creation system as for Paint. It is implemented in Zed on Thoth on a Honeywell Level 6 minicomputer. An Ikonas RDS-3000 frame buffer is used for graphics output and a Summagraphics Bit Pad One provides input.

4.2 The Object Structure – The User's View

A key aspect of a picture creation system is the user's view of the underlying structure of the objects within a picture and how that structure is manipulated by the user. Different object structures provide the user with various levels of power to manipulate an image. In this section, we examine the alternatives and we describe the flexible, powerful object structure that we have chosen.

A completely flat structure, such as exists in many information provider systems, represents the picture as simply a collection of primitive objects (polygons, circles, rectangles). It does not give the user much power to manipulate "chunks" of the picture. For example, if a picture contains a car and some other objects, then to perform some operation on the car, such as moving it, the user must select every polygon comprising the car. The selection process can be aided by a pointing command that selects all the primitive objects satisfying certain criteria.

For example, the user may be able to select all the primitive objects completely inside a rectangular region by specifying the boundary of the rectangle. Baudelaire's Draw system developed at Xerox PARC [4] uses this object selection technique. Explicitly selecting all primitive objects presents several problems. The more cluttered the picture, the more difficult it is to select the desired objects. It may not be possible to select precisely the desired objects because there may not exist a rectangle that encloses only those objects.

Figure 10. A complex object and its underlying structure.

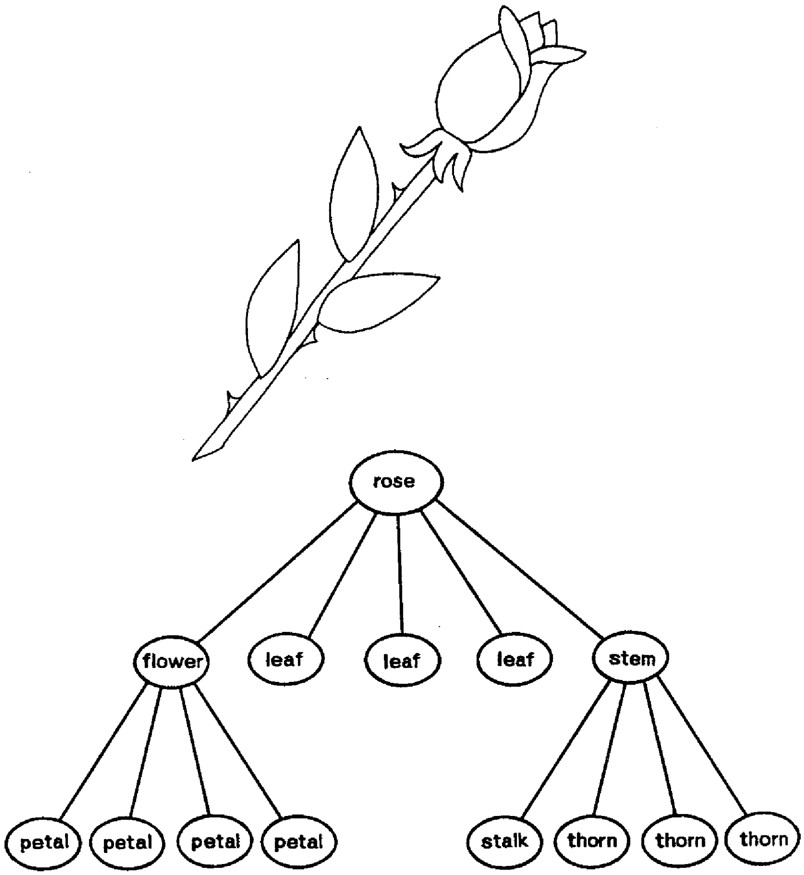
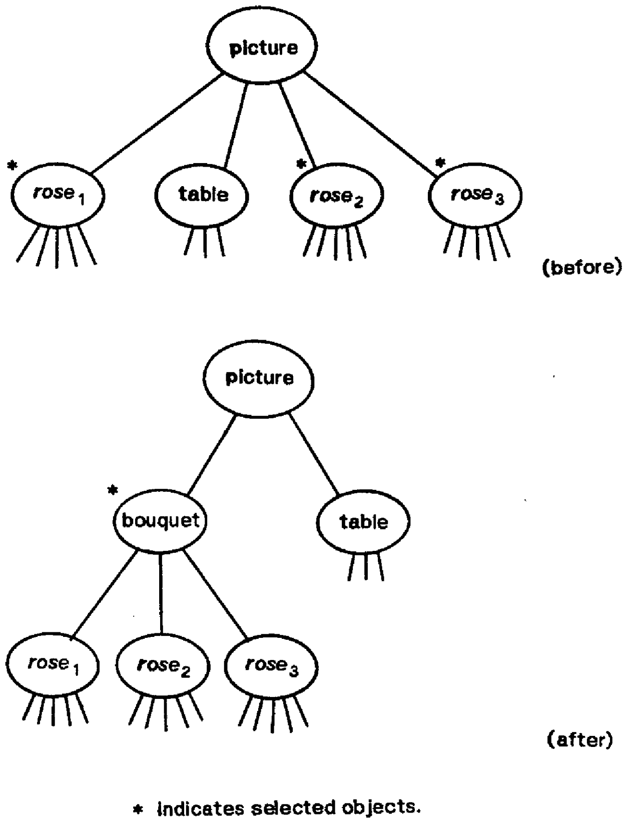


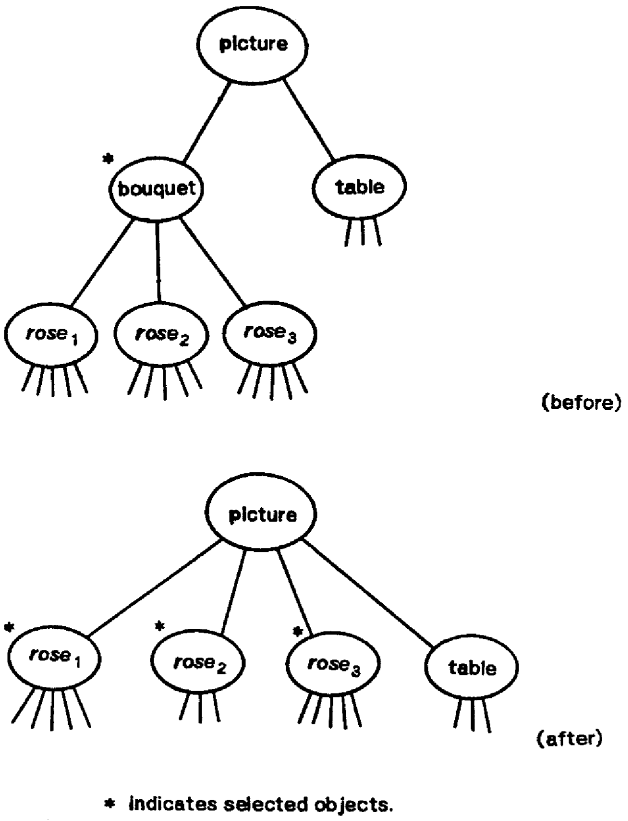
Figure 11. A clustering operation.



one another in the final image. The components of an object can be reordered by applying a single command. The required command simply moves the specified object to the start of the overlap order for the level, causing that object to be drawn first.

When the user performs a selection by pointing, the extent of the object being pointed at must be resolved. The user may be attempting to select a primitive object or a complex object of which the primitive is a small part. Several ways exist for defining the extent of a selection.

Figure 12. An unclustering operation.

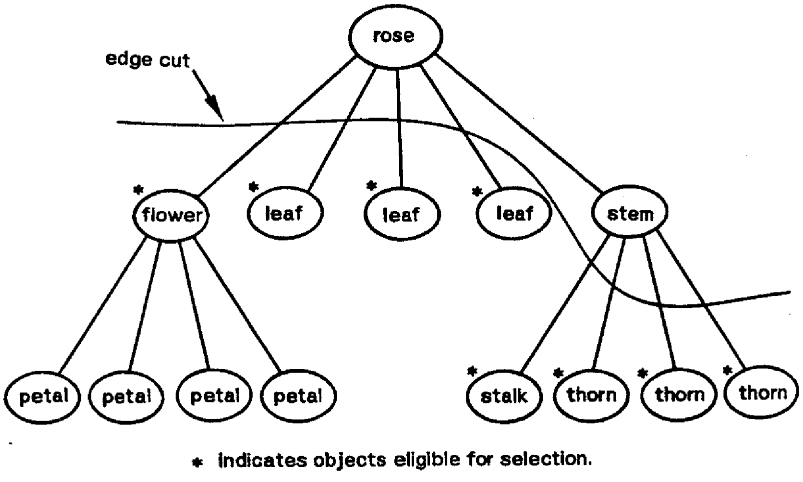


One such method is the select-extend technique used in the Bravo text editor [40] and the Xerox Star [57]. Clicking (depressing and releasing) one button on the mouse performs a selection, which for the Star may be a character, word, sentence, or paragraph depending on the number of clicks. Clicking another button on the mouse extends the selection to the position (character, word, sentence, or paragraph) being pointed at. Holding the button down dynamically changes the selection as the mouse is moved. Releasing the button freezes the selection. The selected region is highlighted using inverse video.

It is not immediately obvious how to expand this technique from a one-dimensional stream of text to a two-dimensional picture possessing a hierarchical structure. One possibility is to depress a button on the puck to select the primitive being pointed at. Then, holding the button down and moving away from the selected point extends the selection, moving up the hierarchy to include more primitives. Moving back toward the original selection point traverses back down the same path in the hierarchy to include fewer primitives, thus selecting a smaller set of objects. Extensive feedback must be provided to inform the user of the original selection point and the extent of the selection during the operation. This feedback may be in the form of highlighting the selected objects or by somehow tagging them. A problem with this technique is that selection of an object is almost always a complex task involving extension.

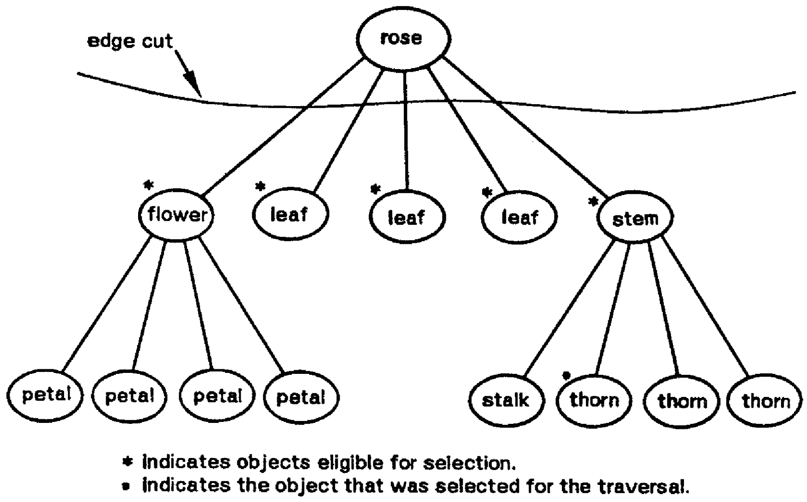
An alternative technique, which we have implemented, involves pre-defining the level of selectability. When an object is pointed at, the system knows what level of detail to use. The user is provided with a means for traversing the hierarchy to set selectability levels. The level need not be the same for all objects. The instance tree for an object hierarchy is a tree whose nodes represent the objects in the hierarchy and whose edges indicate the component relationships among those objects. Selectability is determined by an edge-cut of the instance tree for the hierarchy, such that the root of the instance tree is disconnected from all leaves (primitives) of the tree by the cut set of edges. The selectable objects are the roots of the resultant (sub)trees containing the leaves (primitives) of the instance tree. Figure 13 shows an example of such an edge-cut and the corresponding selectable objects.

Figure 13. An edge-cut and the corresponding selectable objects.



When an object is selected, any operation performed on it is applied to all primitives that comprise that object. The user changes the selectability (modifies the edge-cut) by selecting some objects and traversing one level up or down the tree from the selected objects (see Figure 14). One puck button indicates movement up the tree, another movement down the tree. Feedback is provided by tagging the objects that are eligible for selection with a small diamond shape. For example, if a rose is selectable as a whole, there is one tag on it. If the flower, stem, and leaves of the rose are individually selectable, each is tagged. Selections are simple to accomplish using this technique. A single click (depressing and releasing) of a button will select an object whose tag is nearest to the tracker. Depressing and releasing the same button, while pointing at the tag of a selected object, deselects that object. Selected objects are distinguishable from unselected objects by the hollow center present in their diamond-shaped tags.

Figure 14. A change in selectability by traversing one level up the tree.


edge cut
rose
flower
leaf
leaf
leaf
stem
petal
petal
petal
petal
stalk
thorn
thorn
thorn
* indicates objects eligible for selection.
• indicates the object that was selected for the traversal.

Which selection technique is best depends on the user's work habits. It he is methodical, working from the bottom up, then the pre-defined selectability method will satisfy his needs well. On the other hand, if he works in a haphazard manner, the select-extend method may be more suitable.

The object structure manipulation and selection techniques described above do not provide an explicit representation of the object structure to the user. The user can only determine the structure of the hierarchy by traversing the selection levels. An alternative, or perhaps a supplement, to the schemes presented above involves explicitly displaying a tree-like representation of the hierarchy. Wetherell and Shannon present some algorithms for drawing "tidy" trees [78]. The user could then perform selections or manipulate the object hierarchy by explicitly manipulating a picture of the object structure.

Several problems exist with such a scheme. The available canvas area for creating a picture must be reduced in order to display the object structure continuously. This would be unacceptable, so the object structure would have to be overlaid on the picture when the user wished to manipulate or examine it. Besides taking a long time to be generated, a picture of the object structure may cover much of the screen or maybe not even fit on the screen if the picture is complex. This causes a problem because the user is likely to want to examine the object structure and the picture simultaneously to determine the correspondence between the nodes in the hierarchy and the objects in the picture. Another, more basic problem is that the typical user of such a picture creation system would not be a computer scientist and may shy away from pictures of data structures that he does not comprehend.

4.3 Capabilities Needed

In this section, we identify the capabilities that should be included in a picture creation system. We attempt to provide for a simple, natural, consistent user interface, but we do not discuss the details of a complete user interface design in this section.

The user must be able to create and modify primitive objects. As in Griffin [69], the boundary of a primitive object is defined by a set of control points that are either connected by straight lines or by curves. For curved boundaries, splines can be used to interpolate the control points [29]. The user defines the control points implicitly by painting a stroke, or explicitly by selecting each control point. Editing facilities are provided for adding, deleting, and moving control points. The user can define various attributes of the primitive object. The boundary is painted with a brush that has a shape, size, and colour, as in Paint. The characteristics of the brush are selected by the user from menu items, again as in Paint. The boundary of an object may or may not be closed. If it is closed, then its interior may be filled with a colour, which need not be the same as the boundary colour, or it may be transparent. After creating a primitive object, the user should be able to edit it by adding, deleting, or moving control points, or by changing any of its attributes.

The cluster/uncluster/reorder operations described in the previous section are required to permit the user to group primitive objects to create more complex objects that are organized in a tree-structured hierarchy. The user also needs to be able to use the hierarchy to determine the extent of a selection. The previous section discusses how this can be accomplished.

Once the user has created some objects, he requires global editing features for moving, copying, scaling, and rotating objects in order to make corrections to pictures and to ease the construction of repetitive illustrations. These operations are accomplished easily by a similarity transformation that maps a source vector, P1 P2, into a target vector, Q1 Q2. The user defines such a transformation by specifying the four points, P1, P2, Q1, Q2. More complex operations, such as stretching, slanting, and symmetry, are possible using an affine transformation defined by six points that map a source triangle, P1 P2 P3, into a destination triangle, Q1 Q2 Q3. Such transformations are accomplished easily because the objects are defined geometrically. Baudelaire uses these two generalized transformation operators to accomplish all transformations in Draw [4].

A noun-verb command form, such as is used in the Xerox Star [68] and other systems, permits the specification of any number of objects to be operated on simultaneously. The objects to be manipulated are first selected and then the operation is specified. Operations, such as transformations and object structure manipulations, can be applied in an obvious manner to all selected objects whether primitive or complex.

It is unclear what sort of editing capabilities are needed for groups of objects. We wish to support such operations as changing all the red objects to blue, or changing all the squares to circles. An attribute modification can be easily applied to a random set of selected objects. One merely specifies the new attribute value. Changes to boundary definitions, on the other hand, are only meaningful for multiple instances (copies) of a single object. The editing could be performed on one prototype object with the system taking care of applying it to the rest. It the user wishes to perform an operation on all objects having a particular property, he may well become bored if forced to hand select many objects having that property. A selection operator of the form "select all objects satisfying property X" is desirable here. Unfortunately, the specification of "X" may itself be rather complicated.

While the ability to specify control points anywhere on the screen provides tremendous flexibility and freedom, it does not provide the control that is often necessary in the creation of a picture. When creating an illustration, one often wants to constrain in some way either the position or the shape of the object being created. One may wish to create exactly horizontal or vertical lines, one may wish to make some line parallel to or the same length as another line, or one may wish the object being created to be joined to an existing object.

The constrained drawing techniques implemented in Paint are directly applicable to a picture creation system and provide additional control over the placement of the points that define the directions and positions of the lines and curves that form objects in a picture. Grid constraints provide control for performing particular transformations by constraining the points defining the transformation. Another useful form of constraint is a gravity constraint, whereby a selection yields the nearest control point on the nearest curve. This technique, implemented in Draw [4], is useful for interconnecting objects or curves.

An important feature of any illustration system is its capability for dealing with textual information. It may seem strange to be very concerned about text in a picture creation system. However, as pointed out by Mills, words are a very effective means of communication that serve to provide the context for, or constrain the meaning of, a picture [46]. Since we desire to use the system to provide information, text *must* be an integral part of the system.

Minimally, a picture creation system should provide text facilities similar to those in Paint. One needs the ability to place a single line of text into the picture. A variety of typefonts and sizes should be supported. Facilities for left, right, or center justification of text horizontally, and for top, center, baseline, or bottom justification vertically should be provided. The user should be able to manipulate a line of text, edit it, and change its attributes just as for any other object.

A superior text facility would include more advanced text formatting capabilities, such as those found in NROFF/TROFF [54]. These capabilities include a paragraphing facility that provides for automatic filling and justification of lines of text. Brader surveys conventional text formatters and their capabilities [14].

File save and restore capabilities are also essential in a picture creation system. Simply saving the PDIs necessary to regenerate a picture is not sufficient. Enough information must be stored to enable the complete reconstruction of the entire object data structure for the picture, including the hierarchy for object selection, so that a user can restore a picture and pick up working where he left off.

The file restore mechanism should include facilities for merging user-selected pieces of pictures stored in separate files, as is done in Griffin [69]. This would allow one to easily reuse portions of pictures, such as logos. One could build a library of symbols (decals) that even an untalented user can combine to create art work.

The colour selection facilities should allow the user to take full advantage of the available hardware. Since the designers of Canada's Telidon and AT&T's Presentation Level Protocol have finally recognized the usefulness of colour tables and provide PDI instructions for manipulating a colour table in the most recent specifications [1,53], the picture creation system should permit the information provider to make the most of this and similar hardware features.

A colour lookup table maps pixel values (indices) stored in image memory into digital red, green, and blue intensity values that specify the actual colour displayed on the screen. Colour lookup tables can be utilized in interesting ways to significantly improve the quality of an image. Frequently, only a few colours are needed to generate a picture, but the set of required colours varies from image to image. The colour table allows one to select the subset of colours to be used from a vast set of possible colours. Using a colour table, one can provide better colours without increasing the amount of memory in the terminal or the quantity of data transmitted.

In order for the information provider to take advantage of the colour lookup table, he must be provided with a flexible mechanism for selecting the colour values to place in the table. Colour values are specified to the hardware in terms of red, green, and blue (RGB) intensities. While it is possible for the user to mix a colour in RGB space, Alvy Ray Smith points out that other colour models, such as the hue, saturation, value (HSV) scheme, are more intuitively satisfying and convenient to an artist [64]. A recent article by Berk, Brownston, and Kaufman concludes that the Colour Naming System (CNS), which involves specifying colours using English colour vocabulary (for example, light grayish red-orange), is a better scheme for colour specification than the RGB or HLS (hue, lightness, saturation) systems and results in fewer mistakes by users [9]. This article does not examine the separate issue of trying to match a colour through an interactive colour modification process. We propose to use HSV sliders for mixing new colours and modifying old colours in the colour map in a manner similar to what Alvy Ray Smith implemented in NYIT's Paint system [65]. It may prove desirable to use portions of the colour table for such functions as antialiasing. For this reason, the system, not the user, should manage the actual allocation of slots within the colour table.

The entries in a colour table can be manipulated to provide a limited, but very effective, form of animation [11,61]. Facilities for developing colour table animation sequences, such as those provided by Shoup in SuperPaint [62], would provide the information provider with a valuable tool for creating dynamic images.

One feature that proved to be an invaluable tool in Paint is the zoom operation. This simple hardware feature, although it does not provide greater resolution, magnifies the image to facilitate detailed work. This facility would be equally useful in a picture creation system.

An extremely important feature, which is ignored in many interactive systems, is an undo function. An undo command, such as the one supported in Singh's Benesh Dance Notation editor [63, section 4.3.2], permits the user to easily recover from mistakes or to change his mind after the fact.

In Paint, there is no undo function; one corrects mistakes by repainting that portion of the picture. The lack of an undo function has proven disastrous at times, such as when a user accidentally draws a thick line completely across a complex picture.

In a picture creation system where objects are geometrically defined, the addition of new information into a picture rarely causes problems since such actions can be undone with the corresponding delete command. Deletions, on the other hand, pose a difficult problem, especially when multiple items can be manipulated in a single command. An undo seems crucial for deletions. An undo is beneficial for the object manipulation commands as well because it is in general difficult to reposition something exactly the way it was before an operation.

Various forms of undo permit various degrees of recovery. A simple one-level undo permits one to reverse the effect of the immediately preceding command. This is sufficient in most cases. A more sophisticated scheme maintains a circular buffer of information pertaining to previous commands. This permits one to reverse the effects of one or more commands, the number of commands depending on the size of the buffer and perhaps the type of the commands. For consistency, if an undo is supported for some commands, it should be supported for all commands.

Finally, the picture creation system must be capable of generating PDI instructions from the object data structure in order to display the same picture on a videotex terminal. A preview facility should be available in the picture creation system to permit the information provider to examine the execution of the output PDI instructions to ensure that the image is displayed in a suitable fashion on a videotex terminal. A very useful feature in the preview facility would be the ability to breakpoint or single-step the displaying of a picture in order to track down errors in much the same way that an interactive debugger allows one to breakpoint or single-step a program. Such a preview facility is necessary because an important aspect of videotex pictures is the ordering and timing of the building of objects on the screen.

4.4 The Menu Scheme

In this section, we discuss the design and implementation of a menu scheme for our picture creation system. The menu portion of the user interface in the picture creation system is very different from the one implemented in Paint.

During the development of Paint, we eventually discovered that we could not fit any more menu options into the space allocated on the graphics screen for the one fixed iconic menu. We rejected the idea of expanding the size of the iconic menu to fit more items because this would have forced us to decrease the size of the working canvas area. The auxiliary alphanumeric screen provided an easy solution for implementation because we did not have to change the existing user interface at all, just add to it.

Several major factors influenced the design of the newest menu scheme. Firstly, we wished to avoid using multiple screens. Thus, we decided to attempt displaying all menu information on the one graphics screen, as is done in systems such as Griffin [69]. Secondly, we have even more menu items to display in the picture creation system than in Paint. Hence the space problems are even more severe. Thirdly, users did not like the inability of Paint to fill the entire screen with a picture. Fourthly, it is not desirable to have all menu items displayed all the time. The concept of progressive disclosure practised in the user interface for the Xerox Star [57,68] involves only displaying those menu items that are currently selectable, reducing the apparent complexity of the system in order to prevent the user from becoming overwhelmed and confused.

Numerous techniques for handling menu information are observed in interactive systems. Singh combines several schemes in his Benesh Movement Notation editor [63]. He uses a set of fixed text menus for commands in the background, manipulating the colour table to control whether individual menus or entries are visible. In the foreground, he uses dynamic menus that pop up at or near the tracker. These menus contain Benesh symbols, which can be selected for placement in the working frame of the score. Buxton discusses several menu selection techniques for dynamic menus [15]. In Griffin, Stone uses dynamic menus that appear as needed at user specified locations (not at the tracker), sometimes overlapping with other menus [69]. Baudelaire uses continuously displayed fixed menus in Draw [4,5]. Newman uses dynamic menus in Markup that are displayed only when the user explicitly instructs the system to do so by holding down a button on the mouse [49,50].

Reserving portions of the screen for each menu clearly won't work on a single display screen in a picture creation system. Dynamic menus provide much more flexibility.

The flexible menu scheme we have chosen to implement closely resembles the one used by Stone in Griffin [69]. Menus are overlaid on the canvas area, so the entire screen is available for the picture. The menus are organized in a tree-structured hierarchy. Following the progressive disclosure principle, at any given time we display only those menus from which it is reasonable to make a selection.

The top level command menu is always displayed somewhere on the screen. A menu selection may cause additional menus to pop up while causing other lower level menus to disappear. The menu user interface is designed such that after traversing down through several levels of menu, the user does not have to explicitly retrace his steps to get back out to a higher level menu. He only needs to select the desired menu item on an outer level menu to cause this new action to be invoked. The system automatically takes care of removing from the display screen all the lower level menus that are no longer needed.

In general, we use text menus in our picture creation system rather than iconic menus, which were used in Paint. One reason for this switch is that designing good icons, which convey the desired information, is a very difficult task. Mills discusses the problems of conveying information by pictures [46,47]. Each menu contains a list of items representing commands, options, or attributes, which are displayed in a horizontal row or a vertical column. Each menu item consists of a black box with a white outline and white text. Actually, a grey-scale is used for the text providing two bits of antialiasing to make it more legible. The box size is fixed within each menu, but varies from menu to menu. Currently selected items are highlighted in inverse video (white box with black outline and black text). Only one item may be highlighted in any particular menu. The black-white combination for the interior and outline of the boxes makes the menus very visible on any background.

A couple of exceptional menus do not fit the above description. These are the brush and colour selection menus. As in Paint, we use a brush triangle for selection of the brush size. The interior of the triangle is the current colour. It is outlined in white and black to make it visible on all backgrounds. The colour menu consists of eight coloured paint pots outlined in white forming the working palette. The colour mixing portion of the system allows the user to choose which eight colours appear in the working palette. Other colours not currently in the working palette may occur in the picture because the colour map contains more than eight entries.

Each menu has a (default) display position associated with it. The user can reposition any menu anywhere on the screen. Menus may overlap or run off the screen if the user chooses to have them do so. However, menu items that are not visible on the screen cannot be selected. New menus pop up on top of old ones, never under them. When the user selects a position where several menus overlap, the corresponding item in the topmost menu is selected. When a menu that overlaps with another is erased, menus are redrawn as necessary to fill in any holes while maintaining the correct overlap order.

When the tracker is positioned over a menu, the buttons on the puck have different meanings than when the tracker is not positioned over a menu. On a menu, the yellow button is used for selection of items. The white button is used for moving a menu. When it is depressed, the tracker icon changes to an arrow that follows the puck movement until the white button is released, at which time the menu is repositioned so that its top left corner is at the last position of the tip of the arrow. The menu will appear in its new position each time it is displayed until the user moves it again.

The ability to reposition menus permits the user to fine tune the user interface to his current needs. He can reposition the menus he uses to be near his current working area, thus optimizing the hand movements required to make menu selections and avoiding long or awkward motions. Menu repositioning is especially important because what may be a perfectly natural positioning for one user, who is right-handed, may turn out to be very awkward for a left-handed user. Paint suffers from this problem. It is essentially a left-handed system. Conveniently, the menu repositioning feature also solves the problem of how to work on portions of the picture that would normally lie underneath menus.

The flexible dynamic menu scheme presented here poses several implementation problems. The main one is whether menus can be displayed or erased while still restoring the underlying picture quickly enough. This problem prevents one from implementing the above menu scheme on many systems. Three features of the Ikonas frame buffer provide enough power to make it work.

Firstly, the overlay memory option allows one to mix two images on video output, with the overlay memory taking precedence over the corresponding image (RGB) memory for those pixels where the overlay bits are non-zero. Thus, we can display a menu by writing it into the overlay memory and erase it by erasing it from the overlay memory without affecting the frame buffer image of the picture. The overlay memory and the image memory use different colour maps on video readout so there are no dependencies between the two. An overlay feature could be implemented using the colour lookup table instead, at the cost of reducing the number of bits planes available for the underlying picture.

Secondly, we are able to store a frame buffer image of each menu in the system because our pictures only require eight of the twenty-four bits available in the image memory. This allows us to display a menu simply by copying an image from one spot in memory to another.

Thirdly, the presence of a fast programmable bit-slice microprocessor sitting directly on the Ikonas bus allowed us to implement a specialized version of the RasterOp operator described by Newman and Sproull [51] for very quickly performing the memory copy that accomplishes the displaying of a menu.

The data structure used in the implementation of our menu scheme is an array containing one entry for each menu. The entry contains the following information about the menu:

- its index in the array
- a flag indicating whether it is currently displayed
- the number of items in the menu
- a flag indicating whether the menu is displayed horizontally or vertically
- the width and height of each box in the menu
- where the frame buffer image for the menu is stored
- where on the screen the menu is to be displayed
- a pointer to an array of strings of text for the menu items
- which one, if any, of the menu items is highlighted
- pointers to the next and previous entries in the menu display list

A circular doubly linked list of menu entries is maintained with the displayed menus at the front linked according to their order of display, the most recently displayed menu at the start. This list allows us to search for a menu hit in the correct order and to redisplay the menus correctly when one is erased from the display screen.

Enough information is maintained in the menu data structure to allow higher level software to manipulate menus through the following small set of primitives:

```
Menu_on( menu_index );

Menu_off( menu_index );

Highlight( menu_index, item_number );

Unhighlight( menu_index; item_number );

is_this_a_hit = Menu_hit( x, y; &menu_index, &item_number );

Menu_move( menu_index, x, y );
```

An additional routine (Create_menus) takes care of initialization of various pointers and creating the menu images in the extra frame buffer memory. It was noted earlier that the brush and colour selection menus are exceptions. Create_menus is the only routine that needs to handle these menus as special cases.

Additional menus are easily added to the system by modifying a few tables. This menu data structure does not reflect the hierarchical organization of the menus in our picture creation system. This is handled by the higher level user interface software.

4.5 The Object Data Structure – Implementation Details

In a previous section, we discussed the user's view of the structure of objects in a picture and how he can manipulate that structure. In this section, we discuss the underlying internal data structures and implementation details that make all this possible for the user.

In the section on the user's view of the object structure, we defined an instance tree for the hierarchical structure of the objects. One of the underlying data structures implements that instance tree. It contains a node for each instance of an object defined by the user. Links between nodes indicate the hierarchical component relationships among the objects. Each node in the data structure contains two links, one to its brother (right) and one to its first son (down). The information that is kept about objects on a per instance basis includes the bounding box for the object, whether it is tagged, whether it is selected, the position where its tag is displayed, a pointer to the transformation information for the instance, and a pointer to other information about the object that need not be kept on a per instance basis. This additional information is kept in nodes in the object DAG described below.

Transformation information may or may not be present for any particular node. The transformation information is in the form of a transformation matrix that is concatenated with the current transformation matrix as the node is walked during a traversal of the object DAG.

We have implicitly introduced the concept of instancing objects. It is expected that some pictures will be rather repetitive, containing multiple copies of objects each transformed by translation, scaling, or rotation. A data structure that does not replicate all the data for the object each time it is instanced is desirable. Such a data structure reduces memory requirements and is more flexible because operations may be performed simultaneously on all instances of an object. For example, one might be able to change the colour of all red balls to blue in a single operation.

We require a data structure in addition to the instance tree to support this instancing capability. We use a directed acyclic graph (DAG) whose nodes represent objects and whose edges represent component relationships. In this case, however, a node may be pointed to by multiple nodes. There is one in-pointer to a node for each instance of an object. The links are accomplished using right and down pointers in the nodes. Each object DAG node contains an instance count, which is the number of in-pointers to the node. This instance count is necessary to determine whether to delete the node when an object instance is deleted. The node also contains a pointer to a data information node pertaining to the object.

This data is present only for primitive objects (leaves). A data node contains the object type (text or graphics), all the attribute information about the primitive object, and a circular doubly linked list of its control points.

For graphics objects, the attribute information includes whether it is closed, whether its control points are joined by straight lines or spline curves, the size of the brush used to paint its outline, its outline colour, whether it is filled, and its fill colour. For text objects, the attribute information includes the text string, typefont, size, and justification. Text objects only possess one control point, the position of the text object on the screen. The attribute information is not actually stored in the data node itself. Rather, pointers to the appropriate attribute nodes are stored in the data node. This allows separate objects to share common attributes and makes it easy for a change of attribute to take effect for many objects.

It is not clear whether the object DAG needs to be stored as a separate data structure from the instance tree. All of the traversal information is stored in the instance tree and each node in that tree points to its corresponding node in the object DAG. Thus, the down and right pointers maintained in the object DAG are redundant. We do not currently manipulate the object DAG except to construct it. We could eliminate the DAG if we move the instance count into the data node and have the instance tree nodes point directly to the data nodes. Although the current implementation includes an explicit object DAG, we will probably eliminate the DAG in the next implementation, unless we discover that it is needed to accomplish instancing operations beyond those we have considered.

A more general structure would support multiple "views" of the object DAG, thus allowing a user to group objects in different ways corresponding to different conceptualizations of the picture. This auxiliary structure would be kept as an and the data structures while it is executing. The multiple process capabilities allowed us to build these fairly sophisticated debugging tools without modifying the code for the program itself.

4.7 On the Use of Tablets

Tablets have not been used with a great amount of success in some information provider systems, notably Norpak's IPS-2. Pickersgill has the following comment about the tablet on the IPS-2:

> When the tablet arrived, the task was made easier. Moving the cursor was not "spot-on" accurate even then, but the cursor could be moved to the general area and then fine-tuned to the exact spot by the keyboard. An arduous procedure, but one that was necessary if there was to be any precise control of the image. This was like using a skipping ballpoint pen to compose a love letter. The emotions and thoughts are tumbling out in a rush and the damned tool won't deliver the goods. [55, p. 5]

This is partly the fault of the hardware designers, but most of the blame lies with the software team's inability or refusal to write programs that deal with the tablet in a reasonable fashion. Programmers must accept it as a fact of life that most (all) tablets are flaky. These hardware problems do not need to be passed on to the users of a system. They can be compensated for in software.

Tablets typically suffer from two problems: jitter (very small perturbations in the coordinate values not due to movement of the puck or stylus), and spikes (very large perturbations in the coordinate values not due to movement of the puck or stylus). Both of these problems occur in *every* tablet of which we are aware. Both of these problems can be eliminated or significantly reduced by applying a filtering process in software to the coordinates as they are received from the tablet. One merely throws away all coordinates where the change from the previous set is less than some threshold value (a small number like 2 is usually good here) or greater than a second threshold value (a large number like 300 in a 2000 coordinate system usually suffices here).

A second problem is that most tablets provide a continuous stream of coordinates. Most of these coordinates are duplicates that must be thrown away by the program because of their lack of new information. Hardware manufacturers should build tablets that automatically throw away these duplicate coordinates, if desired, in order to reduce the tremendous overhead involved in processing this useless input. Because most of them do not, the tablet interrupt handler should do the job.

Another complaint made by users of information provider systems is that they cannot accurately place endpoints to achieve horizontal, vertical, or parallel lines. They find the step function cursor control keys much better for this. For this reason, some of them almost never use the tablet. Again, this is not a problem inherent in the tablet. It is merely that the designers have not programmed the tablet correctly. The constrained drawing facilities implemented in Paint provide exactly these facilities using a tablet. One merely has to filter the tablet input correctly. For horizontal lines, we ignore the y-values of the input. For vertical lines, we ignore the x-values.

The tablet is a much more natural input device than a keyboard because pointing is a more natural action than typing. It must first be programmed correctly before it will become usable by the information provider. A picture creation system that uses tablet input correctly will surely be more readily accepted than one that relies on keyboard input. The same is true of joysticks, trackballs, mice, and the other common pointing devices available for use in information provider systems.

4.8 User Interface Considerations

In this section, we briefly discuss some of the issues in the design of the user interface for our picture creation system. Many of the concepts mentioned here are discussed extensively in Chapter Two, so we gloss over some details here.

We do not have a complete design. We only have some ideas that we are experimenting with in the small portion of the picture creation system that we have implemented. We have yet to implement a complete picture creation system. Only then can we determine the success of our design by subjecting the system to user acceptance tests (more properly said, we subject the accepting user to system tests).

The goals of the user interface design for our picture creation system are exactly the same as those for Paint. We wish to provide a system that is "instantly" usable by naive users and that provides a medium where the artist's creative processes can flourish.

In Chapter Two, we listed a set of principles that we followed in Paint in order to realize our goal of providing a simple, flexible, easy to learn user interface. These same principles guide the design of the user interface for our picture creation system: timely user feedback, program state always visible on the screen, input by pointing not by typing, simplicity, consistency, and modelessness.

As in Paint, we use the iconic tracker as the primary mechanism for providing program state information and other user feedback in the prototype implementation of our picture creation system. This technique was, we feel, a tremendous success in Paint. Because of the added complexity of a picture creation system, it is even more important to provide the user with extensive feedback and cues.

Our first hand experience with Paint and currently available information provider systems, such as Norpak's IPS-2 and Northern Telecom's VIPS, has taught us several lessons about designing interactive picture creation systems.

All three make use of a graphics screen, a standard alphanumeric screen, a keyboard, and a tablet. But each system uses those devices in different ways. The IPS-2 and VIPS use the alphanumeric screen and keyboard extensively. In fact, there is a cursor control keypad on the keyboard of the IPS-2, so that the tablet need not be used at all. Paint, on the other hand, uses the keyboard as little as possible and uses the alphanumeric screen sparingly.

Our experiences with Paint and the information provider systems indicate that the use of multiple screens is a bad idea. Having to shift one's focus of attention from one screen to another during an operation causes the loss of visual continuity. The IPS-2 and VIPS are particularly bad in this respect because all menu selections are performed on the alphanumeric screen with only the placement of graphic elements done on the graphics screen.

Our experiences have also provided us with some guidelines concerning the use of input devices. A keyboard is a particularly poor input device and should be avoided when possible. Pointing, which can be accomplished using a tablet or mouse, is a natural action. Typing on a keyboard is not natural; it requires a considerable amount of skill. A keyboard is not even necessary for some text entry as this too can be accomplished using menus. Multiple input devices should not be used during an operation. On the IPS-2 and VIPS, the user is constantly switching devices, entering a command from the keyboard and then coordinate data from the tablet. Tactile continuity is destroyed by such actions. The IPS-2 does allow one to use just one input device, namely, the keyboard. This does not improve the tactile continuity, however, because one still has to grope for the cursor control keys.

The IPS-2 provides a variety of ways to input data. Commands can be typed on the keyboard or selected from menus using the menu cursor control keys on the keyboard. Coordinate data can be typed, provided by the tablet, or entered using a second set of cursor control keys on the keyboard. A joystick is also available for inputting information. This great variety of input techniques and devices only serves to confuse the user. Foley and Wallace claim that

> Input languages which adopt a one-device philosophy, using a lightpen or tablet as the sole hand-activated device for action sequences, emphasize tactile continuity and have great attractiveness when they can be applied [28, p. 253].

We conclude that, once invoked, a picture creation system should use a single graphics screen for all output (menus, messages, and graphics) and a pointing device, such as a tablet, as the primary input device. A keyboard may be used for inputting text strings, however its use should be extremely limited. We have adopted this philosophy in our implementation.

Pickersgill says the following about his experiences using Norpak's IPS-2:

> My first impression before the arrival of the tablet was that there was a total absence of "feel". The tactile sense was completely absent. Consequently the whole creative process was shifted to an intellectual consideration of the connection between which keys to punch and the image that would be produced by doing so. The burden of creation was shifted from one mode of consciousness to another, different, and inappropriate one. [55, p. 4]

This is exactly what we aim to avoid in our picture creation system. By making the program completely tablet driven where the user can paint the outlines of objects, we hope to provide the sense of "feel" that is so important to the artist. Pickersgill goes on to complain that the greatest problem with graphic images produced for Telidon is their "lack of spontaneity" because the input process requires that the artist decompose the image into points, lines, arcs, circles, and polygons [55]. We hope that our method of input, whereby the artist can define objects by painting their outlines, will be less restrictive.

We have developed a very flexible menu scheme to support the user interface. Use of the progressive disclosure principle helps to guide the user through complex actions. The relocatable menus allow the user to fine tune the system to his liking.

We recognize the desirability of avoiding the use of modes, a principle espoused by Buxton and Tesler [17,72]. The noun-verb command form, used in the Xerox Star [68] and elsewhere, helps us achieve modelessness. When we do use modes, however, we make them clearly visible to the user by changing the tracker icon when he switches modes.

5. Proposal for a Hardware/Software Configuration

In this chapter, we discuss a system configuration for an information provider system. We have been developing our prototype picture creation system using a Honeywell Level 6 minicomputer and an Ikonas frame buffer system. The frame buffer is a much more powerful and expensive system than is needed for a picture creation system. The computer runs a time-sharing system and the hardware is expensive. A time-sharing system cannot guarantee real-time response with large numbers of users, an important factor in interactive programs. In practice, the programs are run with only one user logged on to the system. Instead of a general time-sharing system, an inexpensive, stand-alone system that can guarantee real-time response is all that is needed.

Below we present a list of requirements that should be satisfied by the picture creation system configuration:

- The system should support the current and projected Telidon technology. It should at least support the minimal requirements of DOC's 709 protocol [53] (and AT&T's PLP [1]) and preferably the full standard.
- The system should be user friendly. The information provider workstation should be quiet and comfortable to use. The interaction devices should be configured so that they are easily accessible.
- The system should be programmer friendly. It should provide a good software development environment that aids the programmer in developing good information provider system software and encourages alternative approaches and iterative design.
- The system should incorporate at least as much computer software engineering as is found in a good commercial word processing system.

The hardware/software configuration for a picture creation system must include the following components: a frame buffer, a colour monitor, a graphics tablet, a keyboard, a processor, memory, disk storage, an operating system, and software tools to support the development of the system.

The 709 Telidon protocol requires that terminals are minimally 256 by 200 by 4 bits resolution. We expect that soon there will be Telidon terminals on the market having 512 by 512 by 8 bits resolution with a colour lookup table, so we are targeting our design towards such terminals. Thus, the frame buffer used in a picture creation system for developing pages must have at least this resolution and a colour lookup table. Additional depth in the frame buffer would be very useful, especially if the techniques of Chapters Two and Four are implemented.

The frame buffer memory must be readable. Write only memory is inflexible, prohibiting almost all pixel manipulation operations. For example, RasterOp operations [51] cannot be done. Techniques, such as antialiasing, are not possible if the frame buffer cannot be read. It may seem silly to mention such an obvious requirement, but the frame buffers in current Telidon terminals are not readable by the host.

The frame buffer should have a user-definable hardware cursor, not simply a crosshair. This permits the implementation of iconic trackers that provide very effective user feedback.

Additional features that would be very useful are hardware zoom and write masks for selectively modifying individual bit planes.

The processor for the system should have at least a 16-bit wide data path. 8-bit microprocessors are not powerful enough. A large address space (more than 16 bits) is needed in order to allow lots of memory on the system. A picture creation system is a complex program. During execution it must maintain large data structures defining the objects in the picture. The amount of data space required increases with the complexity of the picture. Computer memory is relatively cheap. Rather than forcing the programmer to deal with memory shortage problems, we think it is more cost effective to provide a large amount of memory.

A reasonable quantity of auxiliary disk storage is required to store the large number of pictures that will be generated using the system. Floppy disks are slow, hampering the loading of large programs and large data files. They are also quite noisy. A hard disk, such as a Winchester, is a better alternative having more capacity and greater speed and (perhaps) generating less noise. In all cases, the disks should be quiet if they are to reside in the same room as the user of the system since their noise can be very annoying and discomforting to the user, a situation we wish to avoid.

Some sort of communications port to the outside world is needed in order to be able to transfer the picture files to the host computer that will be used for distributing the information to viewers.

A picture creation system is a complex program involving many concurrent asynchronous activities. The operating system must guarantee real-time response for these activities. Because of the multiple-process nature of the picture creation system we are designing, it would be especially useful if the operating system supported multiple processes (for user software) with an efficient means of communication among processes, such as message passing. The Port system [44] under development in the Software Portability Group at the University of Waterloo is such an operating system for single user workstations.

A file system supporting a rational organization for files containing pictures is needed. If possible, this should be the same organization as used for the data bases of Telidon pages in the information delivery system. The file management systems in Norpak's IPS-2 and Northern Telecom's VIPS impose severe restrictions on the organization of files. Files are identified by short names and there is no directory structure. As a result, in a production environment the users are dependent on paper logs to keep track of their files [58]. A tree-structured file system, such as is found in Unix, Thoth, and Port, is more reasonable. Such a system allows the user to organize his files appropriately. In Thoth and Port, the restrictions on file names are loose enough (for example, 31 characters maximum) that meaningful file names can be chosen. Also, complete status information is maintained including who created a file, when it was last modified, its size, and who has permissions on it.

A high level programming language is needed for the development of the picture creation system. The capabilities of the system being implemented should not be limited by the language it is written in. BCPL-based languages, such as C and Zed, have proven themselves to be reasonable development languages. Fortran and Pascal each have their own set of problems that make them unsuitable for programming the picture creation system.

Finally, the cost of the system is an important factor. It must be competitive with currently information provider systems, which sell for $20,000 to $25,000. We feel that a system costing $20,000 to $35,000 can provide sufficiently increased capabilities to be cost-effective competition for current information provider systems.

References

[1] American Telephone and Telegraph Company, *Presentation Level Protocol – Videotex Standard,* 1981.

[2] American Telephone and Telegraph Company, "Frame Creation Terminal", advertising brochure, 1982.

[3] R.M. Baecker, D.M. Tilbrook, M.I. Tuori, and D. McFarland, NEWSWHOLE, videotape, SIGGRAPH Video Review, No 1, May 1980.

[4] P. Baudelaire, "Draw Manual", *Alto User's Handbook,* Xerox PARC, September 1979, pp. 97-128.

[5] P. Baudelaire, Draw, videotape (not for public distribution), Xerox PARC, September 1975.

[6] P. Baudelaire and M. Stone, "Techniques for Interactive Raster Graphics", *Computer Graphics,* Vol 14, No 3, July 1980, pp. 314-320.

[7] R.J. Beach, J.C. Beatty, K.S. Booth, E.L. Fiume, and D.A. Plebon, "The Message is the Medium: Multiprocess Structuring of an Interactive Paint Program", *Computer Graphics,* Vol 16, No 3, August 1982, pp. 277-287.

[8] J.C. Beatty, J.S. Chin, and H.F. Moll, "An Interactive Document System", *Computer Graphics,* Vol 13, No 2, August 1979, pp. 71-81.

[9] T. Berk, L. Brownston, and A. Kaufman, "A Human Factors Study of Color Notation Systems for Computer Graphics", *CACM,* Vol 25, No 8, August 1982, pp. 547-550.

[10] K.S. Booth and W.M. Gentleman, "Anthropomorphic Programming", *Conference on Issues for Large Scale Computing,* Salishan Lodge, Oregon, March 1982.

[11] K.S. Booth and S.A. MacKay, "Techniques for Frame Buffer Animation", *Graphics Interface '82 Conference Proceedings,* May 1982, pp. 213-220.

[12] J. Borrell, "Digital Paint Systems", *Computer Graphics World,* Vol 5, No 4, April 1982, pp. 61-67.

[13] H.G. Bown, C.D. O'Brien, W. Sawchuck, and J.R. Storey, *Picture Description Instructions (PDI) for the Telidon Videotex System,* CRC Technical Note No 699-E, November 1979.

[14] M.S. Brader, "An Incremental Text Formatter", M.Math thesis, University of Waterloo, 1981.

[15] W.B. Buxton, "An Informal Study of Selection Positioning Tasks", *Graphics Interface '82 Conference Proceedings,* May 1982, pp. 323-328.

[16] W.B. Buxton, "Lexical and Pragmatic Considerations of Input Structures", presented to SIGGRAPH Workshop on Graphics Input Interaction Techniques, to appear in *Computer Graphics,* Vol 16, No 2, 1982.

[17] W.B. Buxton, personal communication.

[18] Cableshare Inc., "Master the Art of Videotex Graphics with the Cableshare Picture Painter", advertising brochure, 1982.

[19] T.A. Cargill, "A View of Source Text for Diversely Configurable Software", PhD thesis, University of Waterloo, 1979.

[20] D.R. Cheriton, "Multi-process Structuring and the Thoth Operating System", PhD thesis, University of Waterloo, 1979.

[21] D.R. Cheriton, M.A. Malcolm, L.S. Melen, and G.R. Sager, "Thoth, a Portable Real-time Operating System", *CACM*, Vol 22, No 2, 1979.

[22] D.R. Cheriton, "High-Level Network Graphics Protocols", *Tutorial: Distributed Graphics and Communications* SIGGRAPH '82, July 26, 1982.

[23] D. Dyment, "A Corkscrew for the Software Bottleneck", *Micros*, Vol 1, No 2, October 1980, pp. 21-24.

[24] W.K. English, D.C. Engelbart, and M.L. Berman, "Display-Selection Techniques for Text Manipulation", *IEEE Transactions on Human Factors in Electronics*, Vol HFE-8, No 1, March 1967, pp. 5-15.

[25] K.B. Evans, P.P. Tanner, and M. Wein, "Tablet Based Valuators that Provide One, Two, or Three Degrees of Freedom", *Computer Graphics*, Vol 15, No 3, August 1981, pp. 91-97.

[26] S. Feiner, S. Nagy, and A. van Dam, "An Integrated System for Creating and Presenting Complex Computer-Based Documents", *Computer Graphics* Vol 15, No 3, August 1981, pp. 181-189.

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

[28] J.D. Foley and V.L. Wallace, "The Art of Natural Graphic Man-Machine Communication", *Proceedings of the IEEE*, Vol 62, No 4, 1974, pp. 250-259.

[29] G.E. Forsythe, M.A. Malcolm, and C.B. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

[30] W.M. Gentleman, "Message Passing Between Sequential Processes: the Reply Primitive and Administrator Concept", *Software-Practice and Experience*, Vol 11, 1981, pp. 435-466.

[31] A. Goldberg and D.H.H. Ingalls, "The Smalltalk-80 System", *BYTE*, Vol 6, No 8, August 1981, pp. 36-48.

[32] R.P. Gurd, "A Microcode C Compiler for a Bit-Sliced Microprocessor", M.Math thesis, University of Waterloo (in preparation), 1982.

[33] C. Hewitt, P. Bishop, and R. Steiger, "A Universal Actor Formalism for Artificial Intelligence", *Third International Joint Conference on Artificial Intelligence*, Stanford University, 1973, pp. 235-245.

[34] Honeywell Information Systems Inc., *Series 60 (Level 6) Multiline Communications Processor (MLCP) Programmer's Reference Manual*, No. AT97, May 1977.

[35] Ikonas Graphics Systems Inc., *Ikonas Programming Reference Manual,* 1980.

[36] Ikonas Graphics Systems Inc., *Ikonas User's Guide,* 1980.

[37] K. Kahn and C. Hewitt, "Dynamic Graphics Using Quasi Parallelism", *Computer Graphics,* Vol 12, No 3, August 1978, pp. 352-362.

[38] A. Kay and A. Goldberg, "Personal Dynamic Media", *Computer,* IEEE, Vol 10, No 3, March 1977.

[39] B.W. Kernighan, "PIC – A Language for Typesetting Graphics", *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation, SIGPLAN Notices,* Vol 16, No 6, June 1981, pp. 92-98.

[40] B.W. Lampson, "Bravo Manual", *Alto User's Handbook* Xerox PARC, September 1979, pp. 31-62.

[41] M. Levoy (editor), *Tutorial: Two-Dimensional Computer Animation,* SIGGRAPH '81, August 3, 1981.

[42] D.E. Lipkie, S.R. Evans, J.K. Newlin, and R.L. Weissman, "Star Graphics: An Object-Oriented Implementation", *Computer Graphics,* Vol 16, No 3, July 1982, pp. 115-124.

[43] M.A. Malcolm, et al., *Zed Reference Manual,* Thoth Computer Research Foundation, University of Waterloo, 1980.

[44] M.A. Malcolm, B. Bonkowski, P. Elder, S. Neely, I. Telford, and P. Didur, *Waterloo Port User's Guide,* University of Waterloo, June 1, 1982.

[45] G. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", *Psychological Review,* Vol 63, No 2, 1956.

[46] M.I. Mills, *A Study of the Human Response to Pictorial Representations on Telidon,* Telidon Behavioural Research Report 3, Canadian Department of Communications.

[47] M.I. Mills, "Cognitive Schemata and the Design of Graphics Displays", *Graphics Interface '82 Conference Proceedings,* May 1982, pp. 3-12.

[48] B.A. Myers, "Displaying Data Structures for Interactive Debugging", Xerox PARC, Internal Publication, CSL-80-7, June 1980.

[49] W.M. Newman, "Markup User's Manual", *Alto User's Manual,* Xerox PARC, September 1979, pp. 85-96.

[50] W.M. Newman, Markup, videotape (not for public distribution), Xerox PARC, June 1975.

[51] W.M. Newman and R.F. Sproull, *Principles of Interactive Computer Graphics,* Second Edition, McGraw-Hill, New-York, 1979.

[52] Norpak Limited, "GC 1000 Picture Creation System User Reference Summary", May 1982.

[53] 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,* CRC Technical Note No 709-E, February 1982.

[54] J.F. Ossanna, "NROFF/TROFF User's Manual", *UNIX Programmer's Manual 2: Supplementary Documents,* Bell Laboratories, Murray Hill, NJ, May 15, 1977.

[55] P. Pickersgill, "An Assessment of the Production of Graphics on Telidon IPS", August 1981.

[56] C.W. Reynolds, "Computer Animation with Scripts and Actors", *Computer Graphics,* Vol 16, No 3, August 1982, pp. 289-296.

[57] J. Seybold, "Xerox's 'Star'", *The Seybold Report,* Seybold Publications, Vol 10, No 16, 1981.

[58] J. Seybold, "Telidon and Videotex: Publishing in an All-Electronic Environment", *The Seybold Report,* Seybold Publications, Vol 11, No 6, 1981.

[59] J. Seybold, "Aesthetics vs. Technology, Part II", *The Seybold Report,* Seybold Publications, Vol 11, No 11, 1982.

[60] J. Seybold, "ANPA/RI 1982: Company-by-Company Review of the Show", *The Seybold Report on Publishing Systems,* Seybold Publications, Vol 11, Nos 22/23, 1982.

[61] R.G. Shoup, "Color Table Animation", *Computer Graphics,* Vol 13, No 2, August 1979, pp. 8-13.

[62] R.G. Shoup, "SuperPaint ... The Digital Animator", *Datamation,* May 1979.

[63] B. Singh, "A Graphical Editor for Benesh Movement Notation", M.Math thesis, University of Waterloo, 1982.

[64] A.R. Smith, "Color Gamut Transform Pairs", *Computer Graphics,* Vol 12, No 3, August 1978, pp. 12-19.

[65] A.R. Smith, "Paint", Tech. Memo. No. 7, Computer Graphics Lab, NYIT, Old Westbury, NY, July 1978.

[66] A.R. Smith, "Tint Fill", *Computer Graphics,* Vol 13, No 2, August 1979, pp. 276-283.

[67] A.R. Smith, personal communication.

[68] D.C. Smith, C. Irby, R. Kimball, B. Verplank, and E. Harslem, "Designing the Star User Interface", *BYTE,* Vol 7, No 4, April 1982, pp. 242-282.

[69] M. Stone, The Griffin Demonstration Tape, videotape (not for public distribution), Xerox PARC, 1981.

[70] Summagraphics Corporation, *Bit Pad One Users Manual,* Manual #64, Revision B, January 1980.

[71] I. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System", *Proceedings – Spring Joint Computer Conference,* 1963, pp. 329-346.

[72] L. Tesler, "The Smalltalk Environment", *BYTE,* Vol 6, No 8, August 1981, pp. 90-146.

[73] C.P. Thacker, E.M. McCreight, B.W. Lampson, R.F. Sproull, and D.R. Boggs, "Alto: A personal computer", Xerox PARC, Internal Publication, CSL-79-11, August 1979.

[74] D.M. Tilbrook, "A Newspaper Pagination System", M.Sc thesis, Department of Computer Science, University of Toronto, 1976.

[75] W.C. Treurniet and P.J. Hearty, "Presentation of Text on Videotex", *CMCCS Conference Proceedings,* June 1981, pp. 5-12.

[76] C.J. Van Wyk, "A High-Level Language for Specifying Pictures", *Transactions on Graphics,* ACM, Vol 1, No 2, April 1982, pp. 163-182.

[77] J. Veeder, personal communication.

[78] C. Wetherell and A. Shannon, "Tidy Drawings of Trees", *IEEE Transactions on Software Engineering,* Vol SE-5, No 5, September 1979, pp. 514-520.

[79] A.R. White, "Pic – C-based Illustration Language", M.Math thesis, University of Waterloo, 1981.

[80] H.A. Wilder and N.F. Maxemchuk, "Virtual Editing: II. The User Interface", *Proceedings – SIGOA Conference on Office Information Systems,* June 1982, pp. 41-46.

The following technical reports are in preparation by members of CGL and will be available by December 31, 1982.

CS-82-41	A Graphics Editor for Benesh Dance Notation *Singh, Beatty, Booth & Ryman*
CS-82-42	A Computer System for Smooth Keyframe Animation *Kochanek, Bartels & Booth*
CS-82-43	Frame Buffer Animation *MacKay & Booth*
CS-82-44	Colour Principles and Experience for Computer Graphics *Goetz & Beatty*
CS-82-45	A Powerful Interface to a High-Performance Raster Graphics System *Breslin & Beatty*
CS-82-46	Picture Creation Systems *Plebon & Booth*
CS-82-47	Anthropomorphic Programming *Booth, Gentleman & Schaeffer*
CS-82-48	A Scene Description Language *Lea & Booth*
CS-82-49	Varying the Betas in Beta-splines *Barsky & Beatty*

Reports in stock are forwarded free of charge. A nominal fee is charged for out of stock items. For an up-to-date listing of available reports, please write to

Technical Reports
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1