The Message is the Medium: Multiprocess Structuring of an Interactive Paint Program

Richard J. Beach
John C. Beatty
Kellogg S. Booth
Darlene A. Plebon

Computer Graphics Laboratory
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
(519) 886-1351

Eugene L. Fiume

Computer Systems Research Group
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada, M5S 1A7
(416) 978-2011

Abstract

An innovative design for an interactive paint program has been developed based on multiple processes and message passing. Traditional paint programs combine interrupt-driven support of a graphical input device, such as a mouse or tablet, with the coloring of pixels in a raster display. We advocate a different design methodology which is illustrated in our implementation. The multiple processes and message passing primitives provided by some real-time operating systems encourage the design of parallel-program architectures and anthropomorphic programming structures, analogous to artist procedures and the metaphors of Smalltalk.

The Thoth operating system was used to experiment with such an anthropomorphic design. Thoth provides a hospitable environment in which to investigate the distribution of algorithms between software and microprogrammed hardware processes, the performance and responsiveness of a multiple-process interactive program, and experimental user interfaces using an Ikonas 3000 frame buffer.

The paint program consists of processes which handle the graphics tablet, track an iconic cursor, paint a selection of brushes, fill regions of the image, draw lines, and implement the user interface. Some processes have been implemented both in software and microcode.

CR Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques — *user interfaces*; D.2.6 [**Software Engineering**]: Programming Environments; D.3.3 [**Programming Languages**]: Language Constructs — *concurrent programming structures*; D.4.1 [**Operating Systems**]: Process Management — *concurrency, deadlocks, multiprocessing / multiprogramming, scheduling, synchronization*; I.3.2 [**Computer Graphics**]: Graphics Systems — *distributed / network graphics*; I.3.6 [**Computer Graphics**]: Methodology and Techniques — *ergonomics, interaction techniques.*

General Terms: Algorithms, Design, Human Factors.

Additional Key Words and Phrases: administrator, agent, anthropomorphic programming, message passing, overseer, paint program, secretary.

Introduction

An interactive paint program has been built using multiple processes and message passing. The program has an anthropomorphic design, assigning independent processes to roles which might otherwise be considered in a human context. This design contrasts with traditional paint program implementations using sequential and interrupt-driven procedures. In our experience, the multiple-process implementation does not degrade responsiveness. To the contrary, process priorities may be altered to fine tune the program's performance, providing added flexibility. An anthropomorphic design clarifies the interaction of processes, promoting better understanding of a complex program. We have found the resulting program to be more reliable and the design more amenable to enhancement than with more conventional methodologies. A goal of the multiple-process design is the eventual migration of parts of the program to microcode. The decomposition into processes leads easily to microprogram modules. The paint program has proved to be an effective tool for creating pictures, an absorbing demon-

stration program for visitors to the Computer Graphics Laboratory, and a testbed for research on user interface design and Telidon information-provider workstations.

Our experience has been that graphics programs are easier to write, debug, and modify when structured as multiple processes. Anthropomorphism adds a clear understanding of this structuring to further simplify the task of system design.

Traditional Paint Programs

Several paint programs have been described in the literature [18,20,15]. There is little discussion in these papers of program structure, but the following general observations have been gleaned from conversations with their designers. Most paint programs use interrupt-driven procedures for the painting algorithms, and use general looping-structure to repetitively test for user input actions. Quite often, overly "clever" code has been used to provide a more efficient implementation of the algorithms.

General features of paint programs are described in the notes prepared by Smith [18]. Some pointing device, often a mouse or tablet, is used to locate the paint brush on the screen. When a button on the mouse or tablet is pressed the program modifies image pixels with the brush pattern. Several painting techniques can be implemented, such as rubber-stamping copies of the brush, inking the brush pattern along the path traced by the pointing device, or simulating an air brush by tinting pixels in a varied pattern about the brush.

The user interface for paint programs frequently relies on an auxiliary screen upon which menu items may be shown, and selected, and a keyboard for entering names of brushes, picture filenames, and sometimes menu actions.

The goal of our work was to implement an interactive paint program with similar features, but using a very different underlying program structure.

Message-Based Operating Systems

The design of multiple-process programs is influenced greatly by the architecture of the operating system upon which they are built [10]. With inexpensive primitives for creating and communicating among many independent processes, different views of program design are possible and practical. The following section describes the message-based operating system used in this project. An awareness of the process and message-passing primitives is necessary to appreciate their influence on the design of multiple-process programs.

Thoth Operating System

The portable Thoth real-time operating system [6,7] was developed at the University of Waterloo and serves as the development system for our work. Thoth was designed to meet real-time constraints, such as those of an interactive graphics program, and to be portable across a variety of mini-computer systems. It has been ported to the Texas Instruments TI990, Data General Nova, Modcomp IV, Honeywell Level 6, and Advanced Micro Devices AM16 systems. Portability was achieved by using a high-level BCPL-derivative implementation language, Zed [16], and a portable compiler and software development system [5]. Our multiple-process paint program was programmed in Zed on the Honeywell Level 6 system.

Several features of Thoth promote the design of programs with many small processes. Process primitives are relatively cheap; the cost of a message communication (send-receive-reply) between two processes is about one millisecond. Since process management in Thoth is inexpensive, processes often have short life-times, being created and destroyed as needed. A group of cooperating processes is called a team and shares memory. Processes are executed in pseudo-parallel by the scheduler. Process communication and synchronization is accomplished entirely by sending and receiving messages. There are no semaphores or monitors in Thoth. It is important to note that although processes in a team do share memory, the memory is never used for synchronization. In fact, all synchronization of memory access is accomplished using message passing.

The Thoth process scheduler provides a priority scheme for process execution. If all processes are created with identical priorities, then natural-break scheduling will occur. This permits the process designer to determine how processes will voluntarily relinquish the processor. All processes will execute until they block, normally due to a message communication primitive (which includes all input/output requests). If processes are allocated different priorities, then events which unblock processes will cause the highest priority unblocked process to be executed. Response time is guaranteed in interactive programs by arranging that processes which control interactive devices have the highest priority, followed by user interface processes, and finally those processes executing algorithms such as area fill which may require large amounts of computation.

Graphics Hardware

The paint program displays its images on an Ikonas 3000 frame buffer, which has 512×512 resolution with 32-bit pixels. The Ikonas system includes four color lookup tables, a cross-bar switch for selectively routing bit planes to the color map, a direct memory access interface to the host computer, and a bit-slice microprocessor. The microprocessor can manipulate the frame buffer image at very high speed. It communicates with the host computer via a scratchpad memory. A "Tiny C" compiler for the Ikonas microprocessor has been written at Waterloo [12]. Several microprocessor algorithms have been written in Tiny C including dynamic iconic tracker and line drawing algorithms.

Graphics input is provided by a Summagraphics Bitpad One tablet. The tablet is operated with either a pen-like stylus or a four-button puck. Novice users quickly adapt to using the pen stylus, although almost always switch to the

puck to avoid several physical complaints with the stylus cord and jitter. People with a fine arts background express a preference for the pen-like stylus, especially for free-hand sketching.

Anthropomorphic Design

Several previous projects, particularly in Artificial Intelligence, have used multiple processes endowed with human characteristics. Artist procedures in Kahn's Director program [13] provide a model of procedures which are told what to do, and then expected to perform those actions independently. The Smalltalk message-object programming system [14,11] provides a similar abstraction, in which an action that is described continues until instructed to change. A more recent use of this technique is reported by Reynolds in these proceedings [17].

This anthropomorphic design technique [3,8] provides a context in which independent program pieces can be built and interconnected in ways similar to the job classifications and reporting roles of a large organization. This analogy extends to multiple-process program design by assigning to processes roles which they can perform, and by creating communication channels between these processes. The concepts of administrators, workers, secretaries, agents, and overseers provide role models for the processes in our paint program.

The Administrator Process Concept

An administrator process maintains a resource and relies on worker processes to manipulate that resource [6,10]. The worker process performs a computation and sends the results to its administrator process. In our paint program, a tablet administrator looks after the graphics tablet. Other processes may request the current tablet position, or ask to be notified the next time the tablet button is depressed.

An administrator process receives messages from its worker processes and from other client processes which request information about the resource it administers. The general coding structure of an administrator is an infinite loop, as shown in Example 1.

Figure 1 shows the organizational chart for an administrator process and its relationship to both its worker processes and its client processes. It is important that administrator processes not be send-blocked, so that they may respond to requests as soon as they occur. In an organizational chart, this is evident because an administrator has only incoming arrows, representing received messages.

The Overseer Process Concept

An overseer process performs two functions. It relieves a worker process from the distractions of interruptions, and it

```
Administrator()
 {
   repeat
    {
     requestor_id = .Receive( message );
     if ( message[TYPE] == FROM_CLIENT )
      {
       if ( info_available )
         Supply_info_and_reply( message, requestor_id );
       else
         Queue_reply_for_later( message, requestor_id );
      }
     else if ( message[TYPE] == FROM_WORKER )
      {
       Update_info( message );
       .Reply( message, requestor_id );
       if ( reply_queued )
         Supply_info_and_reply( message, remembered_id );
      }
    }
 }
```

Example 1

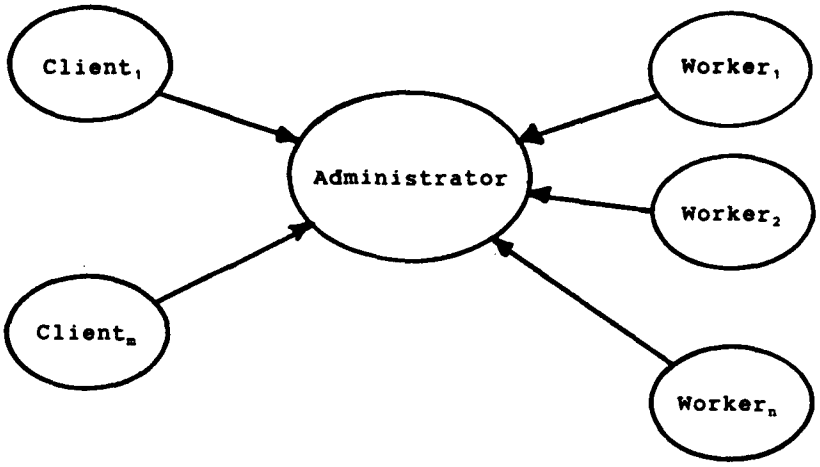

Figure 1. The organizational chart for an administrator process with several worker processes and several client processes. An arrow is drawn between two processes to represent the direction messages are sent.

acts as a watchdog in case the worker process runs out of control. The worker process is programmed simply to perform some task, such as filling a region of the painted image. The overseer process executes concurrently looking for any change in the job description for the worker process. In the paint program, the area fill process is aborted when an overseer process recognizes a panic signal from the user.

Figure 2 illustrates the process structure for an originating process with the worker and overseer processes which it creates. Although only the messages passed between these processes are shown, the worker and overseer may interact with other processes to perform their functions.

Design of the Paint Program

Painting programs are complex. They combine computer graphics algorithms with real-time interaction. Several concurrent activities are exhibited: tracking the paint brush, providing timely user feedback, painting (which may lag behind the user), and user control of these actions.

The paint program uses several process abstractions: hardware and resource administration processes, algorithmic worker processes, and user interface control processes. Administrator processes support the graphics input tablet, the auxiliary terminal screen, and the terminal keyboard. Since several processes will eventually require access to these administrator processes, they are designed to perform simple and well-defined actions. The message communication primitives provide synchronization between the parallel execution of all these processes. Thus, when several processes wish to request tablet coordinates, the tablet administrator sees only a sequence of requests.

Worker processes implement the various painting

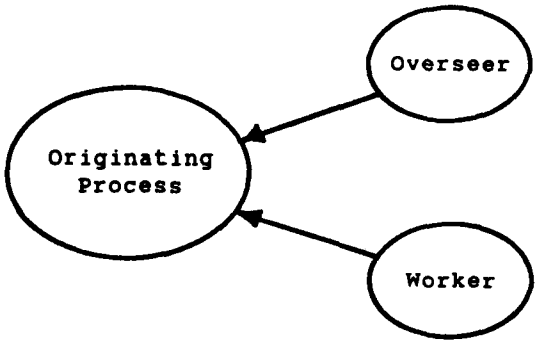

Figure 2. The organizational chart for an overseer process and its associated worker process.

techniques, such as free-form brushing, rubber stamping, and connect-the-dots with straight lines. Area fill is also implemented in a separate process, an organization which simplifies the algorithm substantially. There is no need to check periodically for user actions as this is performed by an overseer or administrator, at the expense of creating another process. This relies upon process creation and destruction primitives being cheap to use.

User interface processes control the actions of the paint program by creating transient processes to perform actions, such as filling a region, and by sending messages to awaiting worker processes, such as for drawing a line. The interaction sequence in the paint program becomes a question of interprocess communication. Processes which perform user-directed actions can be transient, continuously running, or blocked waiting for a work order. User feedback, such as tracking the tablet with a cursor, is an example of a continuously running process.

The organizational chart in Figure 3 represents an overview of the entire paint program, with processes identified as nodes, and arrows connecting two processes when one sends a message to the other. Hardware devices can be represented as fictitious processes that are sinks for messages. They are considered as processes which reply to requests sent by software processes.

Tablet Administrator Process

The interactive painting program requires input from a graphics pointing device [21,9], indicating where the user has positioned the paint brush. Several uses are made of the pointing device: painting pixels, selecting menu items, specifying the starting point for an area fill, and aborting the fill algorithm. As mentioned previously, a tablet administrator handles the graphics tablet. A tablet secretary is created to filter tablet coordinates to eliminate redundant coordinates and jitter (small deflections in the coordinate values due to

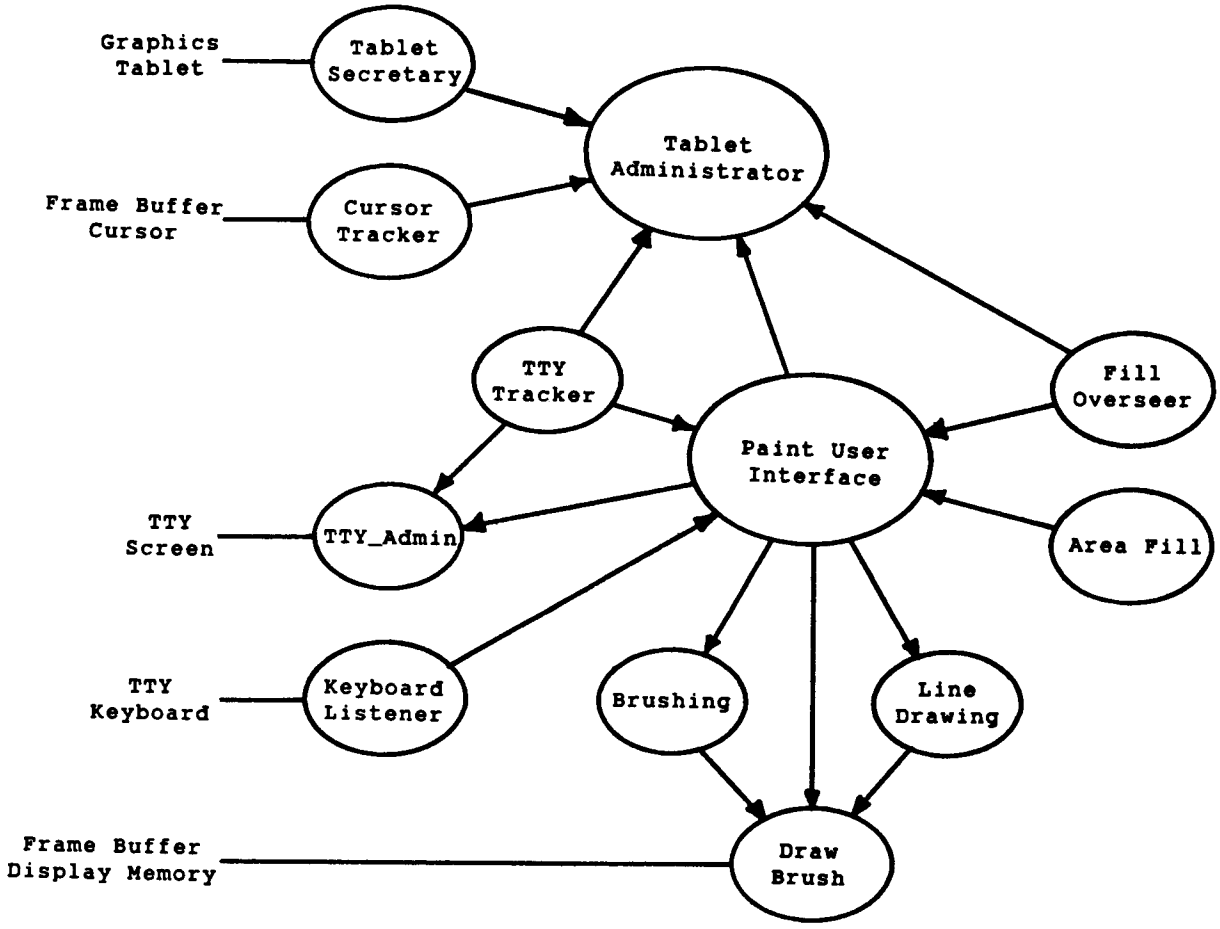

Figure 3. The organizational chart for the multiple-process paint program. Arrows represent the directions messages are sent.

the sensitivity of the tablet rather than motion of the tablet stylus). The tablet administrator accepts several message types: update the current coordinates from the tablet secretary, request the current tablet coordinates, request the next tablet coordinates which are different from the present position, request the coordinates for the next tablet hit when the stylus or button is pressed, and change the tablet sampling mode. The pseudocode in Example 2 illustrates the inter-process communication of the tablet administrator.

The Cursor Tracker Process

An important user feedback is the tracker which echos the motions of the pointing device. In order to guarantee responsive behavior, a free-running process is created to track the tablet by positioning a screen cursor. This tracking process will be created whenever the user is permitted to paint on the screen. At other times, for example, when an picture is being saved or restored, the tracking process is not needed and is destroyed. The tracker process is a simple loop, as shown in Example 3. The start-up parameter `Tablet_administrator` identifies the process which can provide the current tablet coordinates upon request. The request for changed coordinates ensures that as soon as the pointing device is repositioned, the tracking cursor will also be repositioned.

The Draw Brush Process

A basic action of the paint program is to copy an arbitrarily shaped and colored brush into the frame buffer display. A separate process to perform this drawing action has been implemented for two reasons. First, it was intended that this process would migrate from the host computer to the microprocessor in the frame buffer, and that a message-based protocol would enable the two processors to interact reliably. Second, it was realized that for complex brushes, the painting algorithm might be significantly slower than the hand motion of the user. When the user was ahead of the

```
Tablet_administrator()
 {
   repeat
    {
     requestor_id = .Receive( message );

     if ( message[TYPE] == UPDATE_COORDINATES )
      {
        Update_info( message );
        .Reply( message, requestor_id );
        if ( request_queued && tablet_hit_occurred )
          Supply_info_and_reply( message, remembered_id );
      }
     else if ( message[TYPE] == REQUEST_COORDINATES )
        Supply_info_and_reply( message, requestor_id );
     else if ( message[TYPE] == WAIT_FOR_HIT )
        Queue_request_for_later( message, requestor_id );
    }
 }
```

Example 2

```
Cursor_tracker( Tablet_administrator )
 {
   repeat
    {
     message[TYPE] = REQUEST_FOR_CHANGED_COORDINATES;
     .Send( message, Tablet_administrator );
     Position_cursor( message[X], message[Y] );
    }
 }
```

Example 3

painting algorithm, we needed a scheme for painting that would gracefully lag behind eventually catching up. Providing an independent process to draw the brush permits the paint program to display a brush periodically while tracking the user, building up a queue of coordinates for the brushing process to join together as quickly as it can.

The draw brush process accepts two forms of messages, one to establish the brush shape and color, and the other to indicate the position where it should draw. This process is implemented as a server process which receives work order messages. Example 4 shows pseudocode for the draw brush process. Since the brush shape definition may be larger than the Thoth eight-word message buffer, a pointer into team memory is passed in the message. Note that no critical race condition is introduced since messages synchronize processes; the message is not sent until the definition is ready, and the pointer is not used until the message is received.

The Brushing/Line Drawing Processes

The brushing and line drawing processes are very similar. The brushing process tracks the pointing device as long as the button is depressed, periodically sampling its position. The line drawing process connects "hits" with straight lines.

Fill Algorithm Process

The area fill process is a straightforward adaptation of the basic fill algorithm described by Smith [19]. His paper presents the algorithmic details in a much simpler form than he actually implemented. Presumably this is because the details necessary to properly synchronize all of the actions in a complete program would overwhelm the reader. The multiple process approach avoided any complications in handling concurrent activities and thus we could implement the published algorithm directly, 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. An abort capability is needed. 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.

An overseer process is created along with the fill process when a fill is begun. The fill overseer is a very simple process which requests the next tablet hit and sends a message to the user interface process. The overseer blocks awaiting the next tablet hit before sending a message. The area fill process executes until completion before sending a message. These two processes are in a race: the first one to complete sends a message to the user interface, which is blocked awaiting a message from either process. When a message arrives from either process, both processes are destroyed. Either the area was filled completely or the operation was aborted, but it makes no difference to the user interface. The usual problem of race conditions do not exist.

The User Interface Process

The user interface implemented in the paint program presently uses both the frame buffer display and an auxiliary screen for menu information. Since several concurrent processes may wish to interact with the auxiliary screen, an administrator process is needed to synchronize message display. Otherwise, as happened during development, messages can be arbitrarily interleaved as two concurrent processes execute in parallel. The interleaved messages can be extremely difficult to interpret!

When restoring pictures from files, especially during demonstrations, it is convenient to display a menu list of available pictures and use the pointing device to select one. At other times, a new picture name is to be entered from the keyboard. Programming both sources of input is difficult on most operating systems because each input request causes

```
Draw_brush()
 {
   repeat
    {
     .Reply( message, .Receive( message ) );
     if ( message[TYPE] == NEW_BRUSH )
      {
        Update_brush_shape( message[BRUSH_DEFINITION] );
      }
     else
      {
        Display_brush( message[X], message[Y] );
      }
    }
 }
```

Example 4

the program to wait for the input. In a multiple-process implementation, two processes can be created to request the input selection, and the first to succeed can send the selection to the user interface, which is waiting for either process to respond. Thus, a natural user interface feature can be provided simply and elegantly using small processes with short life-times. The processes needed for this feature are a keyboard listener and a cursor tracker for the auxiliary screen.

Impact on the User Interface

The major impacts of a multiple-process design on the user interface are the effects on responsiveness and user feedback.

Responsiveness of Multiple Processes

With several processes executing in parallel on a single central processor, it is imperative that appropriate process scheduling guarantee real-time interaction. Two scheduling methods have been used to experiment with apparent interaction times. The first is a natural-break scheme, in which a process continues execution until it blocks awaiting some event. The second is a priority scheme, in which each process is assigned a process priority and whenever a process is unblocked, the highest-priority unblocked process is executed.

The natural break scheme requires careful consideration of the time necessary to complete an algorithm. For example, the area fill algorithm would continue execution until the area was completely filled, thwarting all attempts to abort the fill process! Thus, breaks are created to provide a voluntary relinquishing of the processor. Normally, such break points occur at the end of an iteration or at other significant points in an algorithm. In the Thoth operating system, scheduling decisions are made when a process blocks on a message primitive. A voluntary relinquish can be implemented as a message send. A sponge process is used to accept these messages as efficiently as possible. The natural break process is simply a repeat-forever receive-reply loop. Example 5 shows pseudocode for this process. Natural break is sufficient for simple algorithms, such as area fill, but becomes both an annoying overhead and sometimes impractical scheme for more complex algorithms such as tint fill.

In contrast, the process priority scheduling scheme requires that a priority ranking be imposed on processes. The highest priority processes will execute, when ready for execution, before lower priority processes. Thus, processes responsible for user interaction are assigned the highest priorities.

In the paint program, the highest priorities are assigned to the cursor tracking processes, middle priorities are assigned to the user interface processes, and lowest priorities are assigned to the algorithm processes. This assignment guarantees that user actions to manipulate the input devices are responded to as quickly as the real-time scheduler permits. Decisions made by the user interface are carried out as soon as all feedback is complete. When no user interactions are pending, algorithm processes can execute. The disadvantage with this solution is that a consistent set of priorities must be assigned the various processes.

User Control of the Paint Program

The multiple-process implementation of the paint program provides for an elegant and simple way of controlling painting actions. Implementing the fill overseer as an independent process relieves the area fill process of any concern for the method of aborting the process. The overseer is a very simple program which expresses the idea of a human overseer waiting to respond to a problem. The benefit to the program designer is that natural and effective user interface actions can now be implemented in a natural and elegant way.

Multiple Input Sources for the User Interface

The use of multiple processes to handle multiple inputs is another example of the simplifying benefits of such a design. The selection of picture names by pointing or entry on the keyboard is modelled by a straightforward process for each form of input. The synchronization of message communication resolves which action occurred first, and the dynamic creation and destruction of processes permits input sources to be enabled and disabled as appropriate.

Migration of Processes to Firmware

A significant goal of the multiple-process paint program was to create a design which permitted the migration of processes from the host computer to a microprocessor embedded within the frame buffer.

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

Example 5

The first process to be moved into a microprocessor was the tablet secretary. The graphics tablet is connected to the host computer via a Honeywell MLCP interface, which uses a special purpose microprocessor to implement various terminal protocols. The tablet communicates by means of a simple asynchronous ASCII protocol. The filtering algorithm performed by the tablet secretary was implemented in the MLCP so that only "good" coordinates would be sent to the host computer and the tablet administrator. This reduces the number of messages by allowing only noticeable motions of the tablet stylus or puck to be transmitted.

The cursor and draw brush processes have migrated into the frame buffer microprocessor, as microprograms written in Tiny C. Messages which contain pointers to brush definitions and positioning coordinates are implemented as data transfers to the scratchpad area accessible to both the host and 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.

Real parallelism now exists between the processes within the host and those executing independently as the tablet secretary, cursor and draw brush processes within the two microprocessors.

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.

Extrapolated Designs Based on Multiple Processes

The use of independent processes makes it natural to think of performing actions in parallel. The artist procedure paradigm helps to design processes which perform a role until the process is told otherwise. Airbrushing is a natural application of this idea.

Airbrushing is a painting technique that causes pixels to be tinted in a seemingly random pattern near the brush position. A process to perform airbrushing might be created when the tablet stylus is depressed. The current coordinates of the airbrush can be moved by the tracking process. While executing, the airbrush process will continue to increase the tint of pixels. When the tablet stylus is lifted, the airbrush process would be destroyed.

Color table animation is a technique to provide dynamic images on a fairly static display such as a frame buffer. The animation process involves modifying over time the contents of the color lookup table. By arranging that areas of the image have unique color table entries, and then changing those entries from background to a highlight color, or by sequencing through several color values, apparent motion can be created. Controlling the timing of such changes is another application for an independent process. After the color table has been defined and the modification specified, a color table animator process can be created to perform the modifications [4]. When the animation is to cease the process can be destroyed. In the interim, various other actions can be performed without added complexity which would arise if this checking were accomplished within a conventional sequential program. This technique follows from the process structures in Dyment [8] and Gentleman [10].

A common visual workspace can be created using multiple-process techniques. This may be possible by supporting several input tablets or by joining several paint programs together. An example of the former might be a two-person game in which several tablets are controlled by copies of the tablet administrator. An example of the latter would be a teleconferencing application where messages within the paint program could be transmitted over a communication network.

Conclusions

The process structuring concepts described in this paper have made the paint program easier to write, modify, and debug than expected with conventional program structures. Anthropomorphism and process structure diagrams reveal the parallelism in interactive graphics programs. We are using these concepts to implement other graphics programs, especially where multiple processors can be exploited. Documentation graphics programs [1] will involve several processes for composing text, mathematical notation, tables, and various styles of charts and diagrams. A multiprocessor implementation of the plane sweep visible surface algorithm [2] will also rely on these multiple process concepts.

Figure 4 illustrates some results of the paint program, showing the Acadia Axeman, a butterfly, a schematic of the Ikonas 3000, the Waterloo crest, the Canadian space shuttle paying a visit to the Arts Library (subtitled "Overdue Notice"), and a tree. The final two pictures show the menu and demonstrate the zoom feature.

Acknowledgement

Much of the research reported in this paper relies heavily on the Thoth operating system which was designed and implemented by the Software Portability Group at the University of Waterloo. The help of Burt Bonkowski, Tom Cargill, Dave Cheriton, Morven Gentleman, Mike Malcolm, Gary Sager and Gary Stafford is particularly acknowledged. Paul Breslin wrote the MLCP microcode for the tablet driver and most of the initial frame buffer support programs. Preston Gurd's compiler has been an invaluable tool for developing high-level microcode. All of the people who acted as guinea pigs for early versions of the Paint program made it possible to test our ideas against actual users; their patience is appreciated. Jim Diamond, Steve Hayman, Steve MacKay, Bev Marshman and Julie Pakalns created the illustrations.

Our work was supported in part by the Natural Sciences and Engineering Council of Canada by Postgraduate Scholarships and Grants No. A3022 and No. A4037.

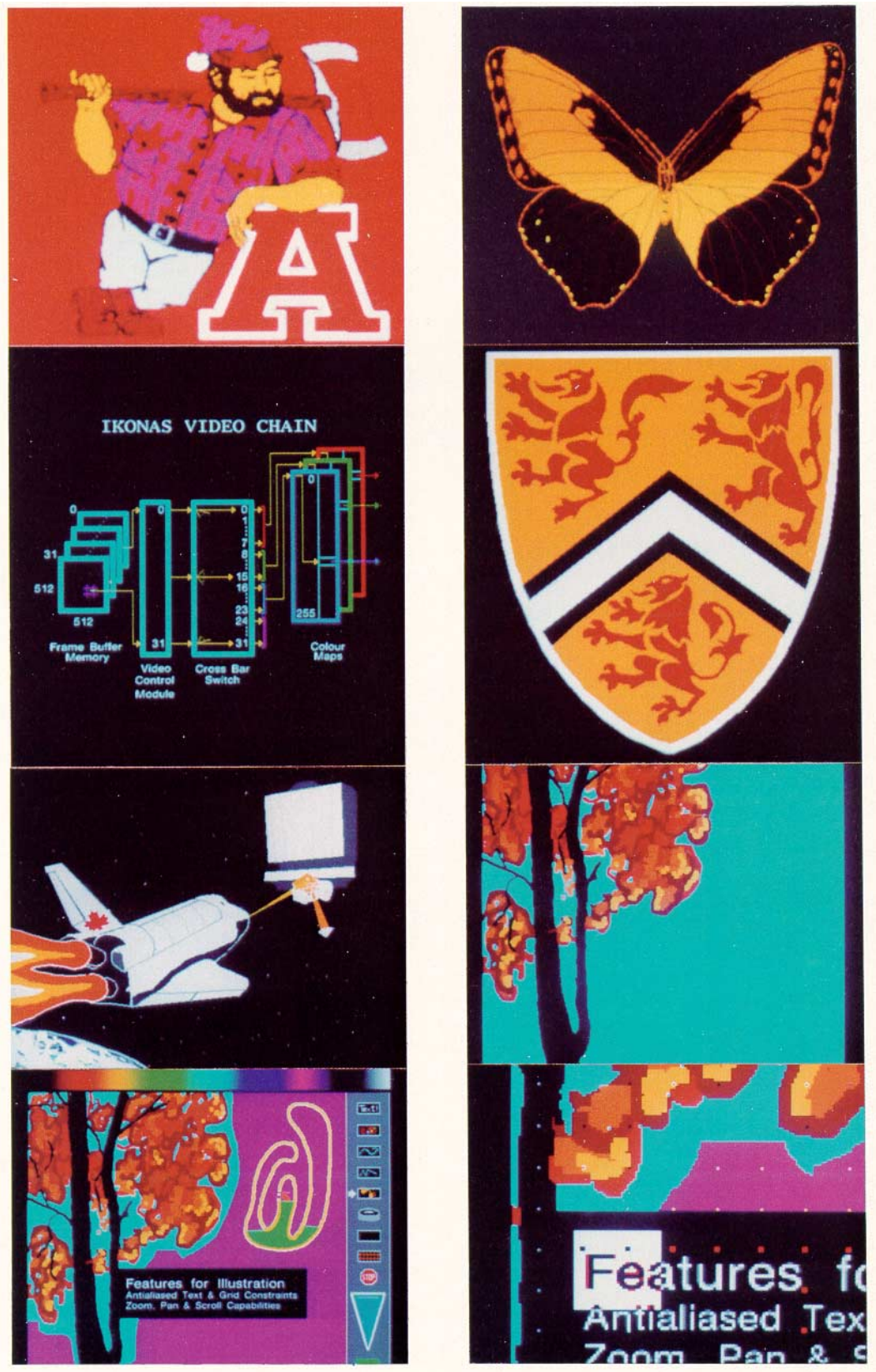

Figure 4. Examples produced using Paint.

Preparation of this paper was assisted by the use of USENET, ARPANET and facilities at the Xerox Palo Alto Research Center.

References

[1] Beach, R.J., Beatty, J.C., Booth, K.S., White, A.R., "Documentation Graphics at the University of Waterloo," International Conference on Research and Trends in Document Preparation Systems, Swiss Institute of Technology, Lausanne, 1981.

[2] Beatty, J.C., Booth, K.S., Matthies, L.H., "Watkins Algorithm Revisited," *CMCCS*, Waterloo, 1981.

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

[4] Booth, K.S., and MacKay, S.A., "Techniques for Frame Buffer Animation," Graphics Interface '82 Conference Proceedings, Toronto, 1982.

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

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

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

[8] Dyment, Doug, "A Corkscrew for the Software Bottleneck," *Micros* 1:2 (October 1980) pp 21-24.

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

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

[11] Goldberg, A., and Ingalls, D.H.H., "The Smalltalk-80 System," *BYTE*, Vol 6, No 8, Aug. 1981.

[12] Gurd, R.P., "A Tiny C Compiler for a Bit-Sliced Microprocessor," Master's thesis, University of Waterloo (in preparation), 1982.

[13] Kahn, K, and Hewitt, C., "Dynamic Graphics Using Quasi Parallelism," *Computer Graphics*, Vol 12, No 3, Aug. 1978.

[14] Kay, A., Goldberg, A., "Personal Dynamic Media," *Computer*, IEEE, Vol 10, No 3, Mar. 1977.

[15] Levoy, M., "Computer Animation Tutorial Notes," SIGGRAPH '81, 1981.

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

[17] Reynolds, C.W., "Computer Animation with Scripts and Actors," *Computer Graphics*, this issue.

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

[19] Smith, A.R., "Tint Fill," *Computer Graphics*, Vol 13, No 2, Aug. 1979.

[20] Shoup, R.G., "Color Table Animation," *Computer Graphics*, Vol 13, No 2, Aug. 1979.

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