Author's version of the paper published in
EUROGRAPHICS '84
K. BO and H.A. TUCKER (Editors)
Elsevier Science Publishers B. V. (North- Holland)

A PAINT PROGRAM FOR THE GRAPHIC ARTS IN PRINTING

P J Willis
School of Mathematics,
University of Bath,
Claverton Down,
Bath, Avon.
UK

There is considerable scope for the application of computer techniques to help the graphic designer working with printed products, provided that pictures of high technical quality result. Described here is the product of the first stage of a research programme aimed at providing computer assisted drawing aids for high quality colour pictures. The product is an interactive paint program. The paper first explains the nature of the application and the hardware available to the author. It then claims that transcription is central to the task of high quality painting. Next there is a description of the facilities available to the interactive user. Finally, a case study of the implementation of one particular command is used to illustrate the general design of the program and to illustrate why this results in a less rigid interaction than in conventional paint programs.

1. NATURE OF THE APPLICATION

The use of colour raster graphics equipment in the printing industry has been less widespread than it might have been, partly because printing produces pictures of substantially better quality than can be shown on a raster monitor and partly because the quantity of data required for a pixel level representation of very high resolution pictures far exceeds that which can be manipulated interactively. One obvious way of creating graphical designs, the paint program, has therefore not been widely adopted. Such programs have found more specialised use, for example in slide making systems. Digital television techniques certainly offer direct picture manipulation [1], but this requires dedicated hardware to produce the affects in real-time and the cost far exceeds that which could reasonably be spent by a design studio. Various applications of high quality graphical techniques to film images have been reported [2,3], but these are produced at some considerable elapsed computing time and cost and are not appropriate to interactive picture origination. Visual simulators [4] produce acceptable quality in real time, but only at enormous cost and by exploiting the limitations of the eye when viewing moving pictures.

Nearer to the specific need is the application of picture processing to the graphic arts[5], again non-interactively, and the digital retouching of colour pictures [6]. This latter application is the closest to the required paint program but it starts from a previously digitised image and offers little artistic freedom beyond layout technique. Some pioneering attempts have been made to apply current techniques in art and graphic design departments of educational establishments [7,8], yet industry has not found it economically sensible to follow this approach. Clearly there is a gap. not entirely technological, between the needs of the graphic designer in printing and the quality offered by paint programs, the simplest of which will run on home computers[9].

The author has been studying the needs of graphic designers working with printing-based media, with the intention of providing computer based drawing aids [10]. Typical application areas include the design of wall-coverings, packaging, posters, laminates and various plastic film products. The graphic designer works in a creative environment and so requires flexibility of drawing, colouring and picture manipulation, yet the final printed product reproduces pictorial material to much higher resolution than is viewable on a colour graphics

terminal. As part of the research programme, a suite of programs is being written to provide very high resolution colour pictures generated from data gathered during interactive sessions. This suite is called UltraPaint. The interactive part of the suite is a paint program called PolyPaint: this is working and is the component described here.

To emphasise the resolution required, some simple calculations are necessary. Low quality printing, such as is found in newspapers, may achieve around 100 points per inch, while high quality colour printing will achieve 500 points per inch. A typographical designer may wish to work with a 1000 line raster for characters at natural size. Hence frame stores resolving 1000 by 1000 pixels are an order of magnitude lacking in resolution for the direct pictorial representation of such images. A single colour picture 10 inches square could easily require 25 million samples and perhaps as many as 100 million samples. Each of these samples might, for the explicit representation of four-colour printing, require four bytes of data. Hence we require individual pictures in the 0.1 to 1.0 Gigabyte range. Clearly there is no equipment which will allow users to manipulate this amount of data interactively and there are certainly no raster displays which will allow the on-screen quality to match that of printing.

A first step in the provision of a suitable design suite is therefore to recognise that the picture viewed on screen is simply a convenient representation of the actual picture. In sharp distinction to many paint programs, in which the on-screen image is recorded on disk and ultimately output to a television caption system or a slide-maker, PolyPaint's picture is presented to the user solely for the purpose of visual feedback and is not in any sense the required product. This leaves the question of where the "actual' picture is and how the user's interaction affects it.

With very high resolution pictures the amount of data is too large for interactive updating to be feasible. PolyPaint therefore makes no attempt at this and is largely decoupled from the rest of the UltraPaint system. Instead, PolyPaint records a transcript file of the user's actions and passes this file to UltraPaint. The latter logically replays the transcript (as a non-interactive batch job) with a synthesised very high resolution frame store and it is the data thus generated which constitutes the "actual" picture. This data is then realised by being used to control an engraving machine to produce cylinders for the printing process. The transcript mechanism is thus central to the operation of the system and will be discussed following a description of our hardware. The paper will then continue to describe the facilities provided in PolyPaint. Finally, some comments on the design of this paint program will be made.

2. HARDWARE USED

The experimental hardware consists of a general purpose mini- computer, in fact an ICL Perq, linked to a framestore which resolves 512 by 512 pixels. The frame store has eight bits per pixel and these are mapped through a colour look-up table to eight bits per gun. There is a Z80A microprocessor in the frame store and this is programmed with a range of graphics primitives for producing lines, manipulating colour and so forth. Communication between the host and the display is across RS-232 at 9600 baud and is therefore not especially demanding of bandwidth or specialised interfacing.

The Perq graphics screen is currently unused as such, the display and keyboard merely serving as a conventional terminal. Indeed none of the program is Perq-specific and it can easily be transported to other mini-computers.

The user's main input device is a data tablet configured to resolve 1024 by 1024. This is the standard Perq-1 tablet and is thus linked to the host via the IEEE interface. However, we have little doubt that performance would be essentially unchanged if this too were an RS-232 link. The Perq's Winchester disk and floppy disk are used for storage of pictures and program in a conventional manner and no special storage structure is required.

3. TRANSCRIPTION

As stated earlier, transcription is central to the aims of the project and so will now be discussed. The aim of transcription is to obtain a record of the user's actions in a form which

will allow the entire interactive session to be recreated. In its simplest form this permits a replay option in which the picture is created on screen, stroke by stroke, seemingly without intervention. While this makes an entertaining demonstration it also has a more serious purpose in that it is a record of the user's intentions. This is in contrast to an image, which is a record of the result of applying those intentions to a discrete space of predetermined resolution. The image contains no useful information about what the user was trying to do and is heavily output device dependent. We have concentrated on tablet-screen interaction initially and have organized PolyPaint so that recourse to a VDU is not necessary for an interactive session, although this is available for test purposes. Hence the transcript file only has to record tablet activity. The tablet generates 60 Hz interrupts which yield the (x,y) coordinates of the puck and the status of the four buttons on the puck. The implementation is in portable Pascal under the Perq Operating System and the combination prevents easy access to the tablet interrupts. Instead the user sees only the state of the tablet at the time of asking and has no simple means of synchronizing to specific events. This prevents the use of realtime transcription of all (interrupt) activity. In fact, recording the full tablet state 60 times a second would rapidly produce a very large transcript file, making it an impractical approach.

We therefore seek to record only significant events. To determine which events are significant we ask which events contribute to the construction of the picture. In fact, as will be seen, most significant actions are signalled by the pressing of a puck button to make a selection. Those that are not in this category are typified by the holding down of a puck button, for example when sketching. The first category is thus transcribed by keeping a note of the changes of button state and writing the (x,y) coordinates of the puck and the number of the selected button as a transcription record. We note in passing that PolyPaint neither requires nor permits a state in which more than one button is pressed. The second category, those events in which the button is held, are transcribed once every time PolyPaint cycles its main task scheduling loop. In addition, transcription occurs when the button is first released, to distinguish separate sequences. A transcript file thus consists of a file of (button,x,y) records in which the button value is zero if no button is pressed and is the number of the button otherwise. Only significant records actually appear which means that rate-determined effects cannot be adequately recorded. However, effects which are determined by the cycling of PolyPaint rather than of a real-time clock, will still be represented.

The replaying of a transcript thus requires only a minor change to the "Read Tablet and Puck" routine: if the transcription flag is true the values are read from the transcript file, otherwise they are read from the actual device. Obvious hazards with the occurrence of replay or transcription menu selections within the transcript file are easily prevented.

The significance of transcription lies in the recording of tablet information, the highest resolution part of the system, for later computation and reinterpretation.

4. POLYPAINT FACILITIES

In this section the main facilites presented to the user will be outlined.

4.1 User's view

The user sees a screen divided in to three areas. The lower part of the screen contains the command menu, the right hand edge shows the complete colour palette, and the remainder of the screen is the drawing area. PolyPaint allows for framestores which have adequate local memory for two complete screen images: in these cases the user sees the same main presentation and can select the alternative frame to use as a full screen picture area. As the menu disappears when this happens, selection is made by toggling a reserved button on the puck. No distinction is made between the drawing area and the picture area except in regard to size, and all operations which apply to one apply equally to the other.

Two other buttons are used. One of these is consistently used as a selector (choosing a brush, paint, mode of working etc) and the other is consistently used for updating (drawing, blending paints etc).

There is no use of menu scroll, in order to keep all options on screen. However there is one command menu entry which allows typing from the keyboard, for items not appearing on the menu. This is currently only used for test purposes.

Commands are selected by pointing and then pressing the selector button. This causes the rectangular cell surrounding the command to change colour. The colour reverts to the default background when the command is completed or, in the case of toggled modes such as transcript on/off, when the command is next selected. Both the background and highlighted colours are part of the palette and are thus changeable to suit the user. However to minimize inadvertent changes they are distinguished by being in the bottom row.

The colour palette consists of an array of small solid rectangles, each corresponding to one of the available colour values. It thus shows the current look-up table state. Each rectangle has associated with it four 'lights', these being small squares inside each of the four corners. Lights which are off assume the colour of the rest of the rectangle. Those which are on are white. One light is used to mark the currently selected colour and a second light is used to mark the previous colour. Hence only one of each of these will be on at any time. A third light is used to indicate all colours which have been used in painting. This has proved an invaluable aid when trying to manipulate colours within the palette without changing tonal values in the picture, because the user can readily identify which colours to leave untouched. This light only becomes illuminated when that particular colour is first used in the picture: merely selected the colour is not enough. It thus serves as an accurate indication of colours in use except where a colour has been used and then entirely overpainted. The purpose of the fourth light is discussed in section 4.3, following. We will now briefly describe specific command menu entries to show the options open to the user.

4.2 Brush shape

Six brush shapes are provided. Each brush is defined as a boolean array covering 16 by 16 pixels. Paint is placed in the framestore only at pixel sites corresponding to TRUE entries in the array. Within this size constraint any pattern is therefore possible, but solid disks tend to be the most useful. It is usually helpful if one of these is large so that it can be used to colourwash large areas or to erase rapidly.

4.3 Drawing style

Pour mutually exclusive styles are available. These are sketch, straight line, spray and infill. The first three use the currently selected brush shape and all use the currently selected paint colour.

'Sketch' requires the user to press and hold the update button while drawing with the puck. A trail of paint is left behind. This is implemented by repeatedly drawing a straight line from the old position to the new one, ensuring that no gaps appear due to rapid puck movement.

'Straight line' requires the user to select two end points. A rubber band line is shown after the first selection and until the second. There is an optional constraint which can be applied to straight lines. This forces them to be colinear with the nearest of three previously defined vanishing points. Hence a simple rendition of arbitrary perspective is possible and the artist can employ tho usual techniques for constructing perspective pictures from known vanishing points.

'Spray' generates an accumulating random pattern of pixels around the current puck location. To provide a degree of control, there is a Mask command which prevents spraying on top of the colour current at the time of selecting Mask. Any number of colours may be so masked and the masking may be undone. A fourth light is used on the colour palette entries to indicate which colours are currently masked.

'Infill' works by sampling the colour value at the indicated start point and then changing all 4-connected pixels of that colour to the current colour. The alternative form of infill, flooding until a specified boundary colour is reached, is less satisfactory for general use as it requires the user to point to the boundary colour as well as the start point. As the boundary could be only one pixel wide this is not necessarily easy to do.

4.4 Colour manipulation

Colour selection is performed by pointing at any area of the screen, except for the command menu, and then pressing the selector button. Hence colour may be read from the palette or from existing parts of the picture. Either way, the colour palette lights are correctly updated.

The red-green-blue components of the current colour can be manipulated directly, using corresponding command menu entries. These offer a coarse selection of the absolute value using a thermometer scale, with a second scale giving small increases or decreases. Direct numerical manipulation is not offered. Moving paint within the palette is performed simply by pointing to a palette colour and presslng the update button. This causes the current colour to replace the one pointed at and, for user convenience, this becomes the new current colour. Colour blending can be performed by selecting two colours in the palette. The Blend command causes the r-g-b components to be individually linearly interpolated between the two extremes and the calculated values are written to all intervening palette colours. Combining blend with colour movement, taking sensible note of which colours are in use, permits colour management to be a relatively simple task.

4.5 Area operations

There is a group of commands which operate on rectangular areas of the picture. These all require the user to indicate two diagonally opposed corners of a source rectangle and the lower right corner of a destination rectangle.

The basic operation is Copy, which simply repeats the source data at the destination location. This is a naive copy pixel by pixel. There are also variants to provide ninety degree rotation clockwise or anticlockwise and also inversion. A further command permits scaling down by a factor of two. Mirroring about a horizontal or a vertical axis is also provided.

Finally, there are two commands for providing shaded areas: one Horizontal Wipe and one Vertical Wipe. These each produce a solid rectangle shaded from the previously selected colour to the current colour. Shading is always from the first edge indicated (old colour) to the second (current colour) so that there are effectively a total of four such wipes. These commands do not literally shade, rather they use the colours between the corresponding palette entries. In general this produces a candystripe effect and true shading is then achieved by using the Blend operation to update the palette.

4.6 Other operations

A Transcript command can be toggled to turn transcription on and off. The Replay command reinitialises the system before replaying the current transcript file. A Type command is' used to send interaction via a keyboard. An Erase command clears the drawing area to a background of the current colour. An Exit command is used to leave the system**.**

5. DESIGN ISSUES

5.1 General comments

PolyPaint is designed for use by artists rather than by computer specialists and a prime consideration was to make the interaction as natural and unrestricted as possible. This prevents it being written in the most obvious manner, with each command invoking a corresponding procedure, as many commands require a sequence of actions. Such a procedure would have to monitor for break conditions at all stages, in case the user had a change of mind, and even so would not allow the user to perform other actions until the command was completed. Hence a user wishing to produce a shaded rectangle using the Wipe facility would not be able to change the colour range once he had started.

We already have experience of writing a paint program in which rapid interaction was not hindered by the underlying technology [10]. In the new program, it was decided to remove constraints on the user brought about by a serial view of the expected interaction. Specifically, the need to complete, or escape from, an action once it was started, rather than simply change to a new action, was viewed as undesirable. It was also felt that the

program should have a concept of one task per window, rather than one task at a time. This gives some of the advantages reported by others [11], yet still allows a simple, small implementation.

As a result, each service routine is written to be the completion of a command assuming that parameters have already been gathered. In that sense, the service routines are still indivisible operations, but the interactive gathering of parameters is performed by polling each time around the main scheduling loop. Each window has its own parameter gathering. Only when a complete set of parameters has been gathered is the service routine invoked. In some cases there are no parameters (for example, changing brushes is performed by pointing at the command menu entry which contains a picture of the brush and so no further interaction is needed) and then the service routine is invoked immediately.

Complementing this is the use of the three window areas, one each for the drawing, the colour palette and the command menu. Each cycle, PolyPaint reads the puck buttons and the tablet position and then determines the current window. It then updates the position of the screen cursor. A Pascal Case statement is used to divide the program according to which button is pressed, if any, and within these sections a further Case distinguishes the window. Hence the program rapidly reaches the code for the current button/window combination. One extra benefit is that this achieves a certain amount of clipping automatically: coordinates are necessarily inside the current window. Further identification of the correct action to take is generally performed by reference to a small number of modal variables. For example, the choice of whether to action sketching, line drawing, filling, spraying, moving or colour wiping is made by reference to a single variable which determines which of these modes is current. Similarly, gathering parameters (the coordinates of areas of the screen) is performed by reference to a modal variable which distinguishes which parameter is being sought. The particular virtue of this approach is that a contradictory menu selection, such as requesting a Shrink operation while having an unfinished Rotate, can be accomodated by reinitialising the appropriate modes. Similarly a non-contradictory operation such as changing colour while in the middle of a colour wipe is quite safe as this occurs with a different window/button action and does not use the relevant modes. Each cycle of the main loop checks for whether there are enough parameters for any particular mode. If so, the service routine is invoked to perform the action. If not, no action is taken. Mode changing is always performed by invoking a specific procedure, partly to ensure no unexpected side-effects but also to permit satisfactory resetting of modal variables and parameter counts. It is this approach which allows incomplete commands to be abandoned simply be selecting alternative commands rather than by using a break technique.

5.2 Case study of the Copy command

By way of illustrating the programming technique employed in PolyPaint, we will now discuss in full how one particular command, Copy, is performed. This explanation will not describe the primitive pixel by pixel copy. Rather it will concentrate on how PolyPaint actions the parameter gathering and calling of the service routine and the particular way that the use of windows, modal variables, button state and button change of state affected the actual program.

The main loop consists of reading the tablet and buttons, determining the window, updating the cursor and then actioning any commands. The latter is performed by using a Case statement to distinguish which button, if any, is held down: that is, the main program division is first by the state of the buttons. If the user now moves the puck, this main loop will keep the (x,y) position up to date and will also change the window parameter whenever the cursor moves into a new window (recall that the windows are the drawing area, command area and palette area). Having positioned the cursor over the Copy entry of the command window he presses the select button on the puck, possibly holding it down for a short while.

At the instant that the selector button is pressed, action is required. If it continues to be held down then no action is necessary (this is in contrast to the update button, which can be used for continued operations such as sketching). For this reason button state change variables

are provided and reassigned each time that the puck is read . Reference to the button state change variables identifies that this button has only just been pressed and thereby avoids needless invocation of code just because the user happens to continue to hold the button down. It also avoids unnecessary transcription. Use of the selector corresponds either to selecting an entry in the command menu or to picking colour from the picture area. The window parameter, updated at the start of the main loop, determines which# of these has happened. If it is a command it is a simple matter to decode the current (x,y) coordinates to determine which command. There are 30 of these so a 31-way Case branch on the the result of the decode reaches the appropriate section of program. The one extra route is to allow for being within the command menu window but missing a valid entry, for example by pointing at one of the box margins. Having reached the appropriate command code, we have also reached the point at which PolyPaint defers calling service routines unless they can be completed immediately. For example, the Erase-picture command can he actioned immediately and so the corresponding section of the 31-way branch calls the routine to service this. On the other hand, the Copy command requires the user to indicate three points in tire drawing area, in order to identify diagonally opposite corners of the source rectangle and the lower right corner of the destination rectangle. We are currently in the command area, otherwise we could not have reached this part of the code, so this is not the place to do it. Instead, PolyPaint invokes the Change Mode procedure to indicate that a Copy is pending and, as a side effect; the Copy command menu entry is illuminated to show the user that it has been selected. Any previously selected mutually-exclusive command is deilluminated.

This is the only action performed at this stage and the program continues with its main loop. Even if the user continues to hold the button down the code will not be reinvoked. The user's next action will be to move the puck in to the drawing area. The update button is then used to indicate the first corner of the source rectangle. Although only the destination area is being changed, the update button is consistently used for all three corners that the user is required to select, partly because it is easier to use and partly because there is a sense in which Copy as a whole is an updating action. When the first selection is made, PolyPaint can immediately decode the window/button combination as before, and it will similarly ignore continued holding of the button. Parameter gathering is now underway and is handled by use of a global defining the currently sought parameter number. At the start this has value one. Branching on this parameter leads to storage of the first (x,y) coordinates in a standard place. It also has the effect of increasing the parameter to two. Therefore, next time the button is pressed inside the drawing window, the second (x,y) coordinates are stored and the parameter is increased to three, ready to collect the third pair. Notice that the update button can be used inside other windows without disrupting this process and that any other button can similarly be used In any window. Collection of the third and final set of (x,y) values invokes the pixel by pixel copy routine to complete the action sought. Completion of the latter invokes the Change Mode routine to allow the program to revert to its default (sketch) mode.

6. CONCLUDING REMARKS

A transcribing paint program has been described. It has been written to permit the interactive user to draw in a flexible manner. In particular, it does not impose unnecessary selection sequences on the user and so needs to give little prompt assistance. The program uses the concept of window-based tasks which are substantially independent of one another. Which task is actioned at any given instant depends on where the user is pointing and so the user remains essentially unaware of the window/task management.

The program is the interactive component of a high-quality paint system currently being developed. Transcription is seen as a convenient way of recording the user's intentions at the highest resolution that the input device permits. It is also a satisfactory basis for inferring picture data which cannot be resolved on a framestore.

7. REFERENCES

1. (Editorial material)
The digital studio
Electronics and Power, September 1982, p. 592.

2. Crow, F.C.,
Shaded computer graphics in the entertainment industry
IEEE Computer, March 1978, pp 11-22.

3. Reynolds, C.W.,
Computer animation with scripts and actors
ACM Computer Graphics, 16, 3, July 1982, pp 289-296.

4. Schacter, B.J.,
Computer image generation for flight simulation
IEEE Computer Graphics and Applications, Oct 1981, pp 29-68.

5. Vanhatalo, S.,
Digital picture processing in graphic arts
Proc. 1st Scandinavian Conf. on Image Analysis,1980, pp 153-156

6. (Editorial material)
Technical trends from electronic DRUPA
Lithoprinter Week, June 30th, 1982, pp 9-20.

7. Colonna, J-F,
From display of computer results to artistic creation
Proc. Computer Graphics 83, (pub. Online Oct 1983), pp 219-232

8. Spars, A.,
Introducing computer graphics into an art and design studio:
trials and tribulations
Proc. Computer Graphics 83, (pub. Online Oct 1983), pp 507-518

9. Pappas C.H. and Hurray W.A.,
Painter Power
Byte, May 1983, pp 263-265.

10. Hanson J.B. and Willis P.J.,
A graphic arts display system
Proc. Electronic Displays 1982, 3, pp 13-21.

11. Beach R.J., Beatty J.C., Booth K.S., Plebon D.A., Fiume E.L.,
The message is the medium: multiprocess structuring of an
interactive paint program
ACM Computer Graphics, 16, 3, July 1982, pp 277-287.