"USIPS": A Telidon Picture Creation Station

Eric Neufeld and Paul Sorenson
University of Saskatchewan, Saskatoon, Saskatchewan

ABSTRACT

A Telidon Page Creation station is in an advanced implementation stage at the University of Saskatchewan. The system takes advantages of certain features of the Telidon design philosophy to create flexible and powerful facility for both page creation and modification.

KEYWORDS: Telidon, videotext, page creation, information provider.

1. Introduction

Usips, the University of Saskatchewan Information Provider System, currently being developed in the Department of Computational Science, is a significantly enhanced version of *vips*, a primitive Telidon page creation system produced by the University of Athabasca. It also incorporates many features of *ge*, a Telidon graphics editor also produced by the same University, in an interactive fashion.

Presently *usips* features real-time verification of geometric primitives. Primitives created may be scaled, translated and rotated in three dimensions. The Sutherland-Hodgeman algorithm has been implemented to perform clipping of points, lines and polygons [1]. At any time, a user may quickly locate any object in a Telidon page by identifying a few of its attributes and may modify or delete it or insert new objects before or after it.

As well, our system features a powerful interactive screen-like text editor. Portions of text also are easily located and modified or deleted, etc. Once fully implemented, *usips* will be a complete page creation/editing station. *Usips* is intended to be part of a fully integrated system. Unlike other systems which require that pictures be created on one computer system and then moved to another, the pages will be produced on the same system as the database from which they will be displayed.

2. Telidon

Telidon is the Canadian version of "Videotex" which is the generic name given to various methods of efficiently storing alpha-geometric information.

There are different approaches to the problem of efficiently storing alpha-geometric information. If resolution is not of prime concern, a raster display may be divided into clusters of pixels, all of which are shaded the same colour. This method, known as *alpha-mosaic*, is used in European videotex systems. This reduces the amount of information which must be stored about the object by a constant factor. However, besides the loss of resolution, it is also impossible to present smooth curves.

Telidon, on the other hand, is the name given to an encoding protocol designed by the Department of Communications of the Government of Canada [2]. Simply stated, the Telidon encoding protocol is a redefinition of the ASCII character set. In this redefinition, the ASCII characters become Picture Description Instructions (PDI) to a device we may call a Telidon terminal in graphics mode. The PDI may be transmitted to the Telidon terminal by a communications processor or by a typist working at a keyboard attached to the terminal, although this usually would be impractical.

In graphics mode, the Telidon terminal acts as a command decoder for the PDI. Each redefined ASCII character becomes either an instruction to the Telidon terminal to begin drawing one of eight geometric primitives or text, or an instruction to reset the status of the Telidon terminal. The characters which represent geometric primitives are followed by an arbitrary number of data encoded Telidon screen coordinates. Conceptually, the status instructions set various attributes of the Telidon terminal - colour or black and white, height and width of text, the density of shading, texture of lines. Thus, if you set the status of the Telidon terminal to the colour yellow, the next object that is drawn will be yellow.

Since the PDI are executed as soon as they are received by the Telidon terminal, the

Telidon encoding protocol is such that a Telidon picture or page "unfolds" as a *sequence* of geometric figures and text which may overlap one another as they are drawn, rather than as a raster or pixel by pixel image. This unfolding is fairly slow.

This sense of "sequence" or "unfolding" of a Telidon page strikes the viewer of Telidon pages immediately. An artist viewing pages of geometric designs consisting of polygons rotating in 3-space created at the University of Saskatchewan commented immediately that it is not just the final image that the user sees which is important, but also the *way* that the image unfolds. It must present itself smoothly, but at a reasonable speed, etc. Artists working at Telidon page creation stations take advantage of this feature in order to create dynamic effects such as the illusion of motion. This aspect of Telidon is not purely aesthetic. Many proposed implementations of Telidon are advertising-oriented, and no advertiser wishes to present jarring or unpleasant pages to his public.

This feature of sequence greatly simplifies the design of a Telidon page creation and editing station. Since the image unfolds as a sequence of sub-images which might overlap, it is not necessary to worry about such problems as hidden line removal. If at any point it is necessary for either the program or the artists to display the image, it is only necessary to write the sequential PDI file to the Telidon terminal as trivially as any ASCII file might be written to any standard device.

The sense of sequence also makes possible the elegant implementation of a traversible stack as the main data structure for the station. Since the user/artist has an idea of the position *in time* of any object in a Telidon page he is creating, the idea of "forward" and "backward" searches are very natural, and reasonably simple to implement.

3. Usips

In terms of hardware and firmware, *usips* consists of an Electrohome colour monitor and Norpak decoder which together act as the Telidon terminal as described above, as well as a Volker-Craig 4404 terminal and keyboard and Summagraphics bitpad and pen which are "hard-wired" together. The bitpad contains a command menu, from which commands (DRAW, SET COLOUR, etc.) may be selected, as well as a drawing area. All these peripherals are connected to the Department's Vax 11/750 running on the UNIX operating system.

4. Invoking Usips

A Telidon artist wishing to use *usips* simply logs into his UNIX account in the normal way and invokes *usips* like any other program. A status monitor illustrated in Fig. 1 is displayed on the VC4404 terminal. The user enters the name of the file to be created or edited. Apart from actual text entry this is the only time the user has to use the keyboard. All other activities transpire through the use of the bitpad.

5. Drawing Objects

The artist selects various attributes for his drawing from the commands on the bitpad menu, shown in Fig. 2. The most recent change is highlighted on the status monitor. He may change his mind as often as he likes. However, the artist can be sure of what will appear when he actually wishes to draw an image since all the information is on the monitor. We are also considering optionally displaying a small rectangle on the colour monitor to show current machine status. However, this interferes with the display.

When the artist wishes to draw, he selects one of the geometric primitives or the TEXT function from the bitpad menu. He then takes the bitpad pen to the drawing area of the bitpad. Each point he draws is echoed on the colour monitor with a small blinking dot. At any time, the artist may fine tune the point, by moving it up, down, right or left one pixel at a time. Or he may alternately step backward and forward through the set of points, moving them slightly until achieving the desired positioning. This idea was borrowed from [3].

When satisfied with the positioning of the coordinates, the artist selects the COPY command which then "connects the dots" and fills the object in according to specifications. The user may then reselect various attributes and redraw the object on top of itself by repeatedly selecting the COPY command from the bitpad menu. This allows for attractive and precise visual effects, such as outlining an object in a different colour or doing colour mixing by overlaying a shape on itself in different colours and different fill patterns. In this sense, the system has a notion of a "current object" which may be drawn and drawn over and over again until the artist begins a new image.

Another way of fine tuning the position of an object is via the MOVE command. Notice that the status monitor contains all the standard motion parameters: scale, rotate and translate. Every time the artist selects the MOVE command, all the motions are applied to the object and it is redrawn. The object's colour, etc., may be changed as it is moved, again allowing for very attractive visual effects.

```
UNIVERSITY OF SASKATCHEWAN TELIDON SYSTEM
FILE- ID[                                                          ]

PICTURE STATUS                TEXT STATUS              CURSOR STATUS

FIGURE   [rectangle ]         SIZE          [3]        COLOUR [white  ]
COLOUR     [magenta ]         HEIGHT        [0]        BLINK      [On ]
LINE     [Solid     ]         ROTATION      [0]
BLINK              [Off]       SPACING     [0]            ERASE STATUS
TRANSPARENT        [Off]       CHAR SET    [ 1]
FILL     X[  0]Y[   0]                                  COLOUR [black  ]
HIGHLIGHT        [Black]     MOTION STATUS              BLINK      [Off ]
DISCRETE
OUTPUT DEVICE      [02]      ROT   [  0][  0][   0]
                             SCALE [ 10][ 10][  10]
CMD [                ]       AT    [ 50][ 40][  50]
COORDINATES[         ]       TO    [  0][  0][   0]
```

Figure 1 Status Monitor

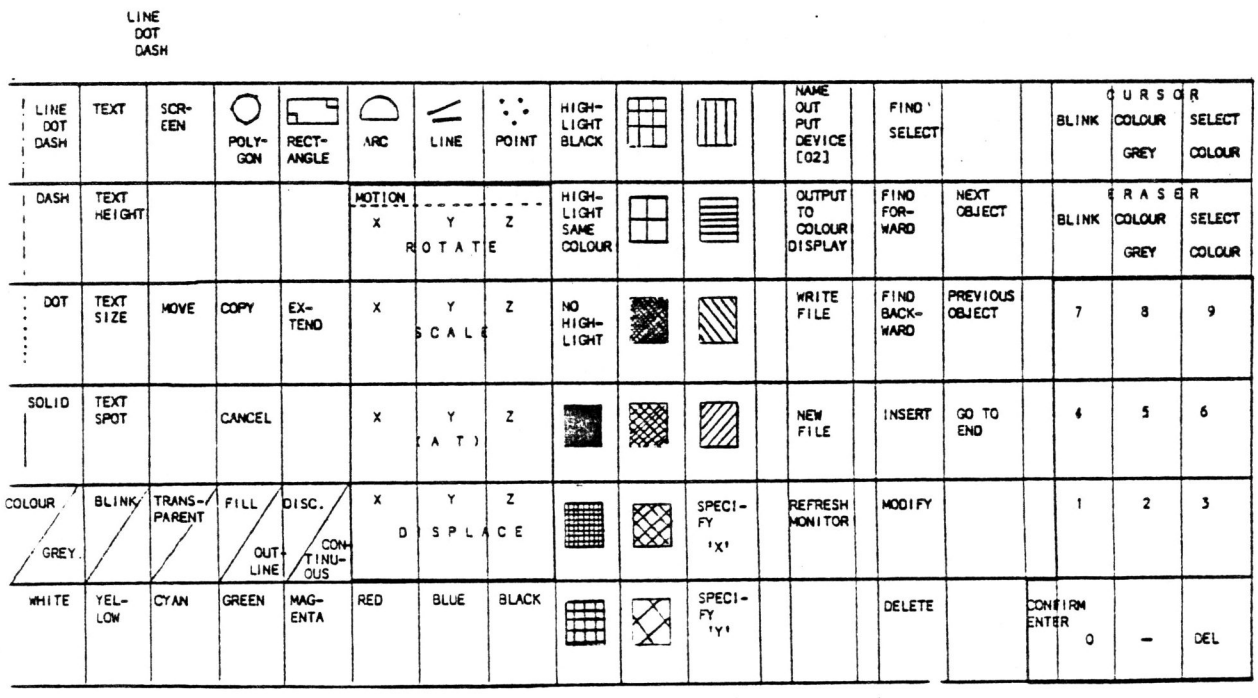

Figure 2 Bitpad menu layout

The setting of the motion parameters, incidentally, was greatly enhanced by the creation of a paper "numeric keypad" on the bitpad menu. A 4 by 3 section of the bitpad menu was made to look just like a standard numeric keypad, complete with a minus, DELETE and CONFIRM-ENTER "keys". This eliminated the nuisance of moving from the bitpad back to the keyboard and back to the bitpad. Whenever the user selects a command which requires additional numeric input, he simply enters numbers via the keypad. The numbers appear in the set of square brackets opposite the appropriate command on the status monitor. Like many calculators, if too many numbers are entered, they simply "fall out" the left.

Although all the geometric primitives are two-dimensional, they may be moved through three dimensions. All objects are clipped, however, the rectangles and arcs behave anomalously when clipped. Polygons, lines and points behave well.

6. Text Entry

If the artist selects TEXT, he is put into a modest screen editor, that is, the characters appear on the colour monitor in real time. All of the cursor motion keys have the usual meaning, with wraparound. The CLEAR key erases all the text in the current screenful from the screen and the HOME key moves the cursor to the upper left hand corner of the screen. Some of the program function keys have been programmed to erase a single character, erase a single line, delete a line, insert a line, etc. Pressing the ESCAPE key saves the current screenful of text and returns the artist to the bitpad menu.

Before selecting the TEXT command from the bitpad menu, the artist may select the point on the drawing area at which he wishes the cursor to appear. This is useful, for example, in labelling diagrams.

One problem that arose during the implementation of the screen editor was the time delay in producing the blinking cursor. To maintain a blinking cursor, it was necessary to write in the order of 30 characters to the Telidon terminal for every character that the artist typed. Since we were dealing with a 1200 baud line, this was extremely annoying to anyone who typed much faster than 30 words per minute. To eliminate this, one of the program function keys was used to toggle the cursor off and on. When the cursor is toggled off, the speed problem is totally eliminated.

Again borrowing from [3], we made it possible for the user to determine what colour the cursor should be and whether it should blink or not. As Chang points out, it makes no sense to place a blinking cursor on a blinking background.

7. Editing

At any point, the artist may return to any object in the file, by selecting the FIND command from the bitpad menu. He then gives the program a "list" of attributes to search for. He may include the shape and the colour, just the colour, just the shape, blink on or off, and combinations of these. He then selects either of the FIND_BACKWARD or FIND_FORWARD commands from the bitpad menu. As soon as an object with the desired attributes is found, it is outlined with blinking white. (Since this can be annoying, the user may turn the blink off without moving the pointer to the current object.) If the object found is not the one the user wants, he may repeatedly select the search commands, and the program will keep going further forward or backward through the PDI file looking for objects with the desired attributes, which it will highlight after automatically turning off the highlight on the previously found object.

Having found an object in the file, the user may elect to insert new objects before or after the current object, without any side effects to the rest of the file. He may also modify the existing object with full power. At any time, the artist may return to the end of the file. If the artist has selected a text object, he may screen edit it as before.

8. Design Decisions

We began with the *vips* software produced by the University of Athabasca, which at that time was in a rather primitive state. The page creation program, *vips*, provided a menu which was a one-to-one mapping from the bitpad to every PDI defined, plus simple line-by-line text entry. Many of the commands at the PDI level are not of interest to an artist. Others incorporate several commands into a single byte. For instance, the PDI instruction describing a rectangle also describes whether the rectangle is to be filled or just outlined and whether the rectangle is to begin from the last point drawn or whether a new object should be started. This is an annoying array of choices to have to make.

Furthermore, an object was both drawn to the Telidon monitor and written to the file as soon as it was drawn. The artist could not "ponder" over an object before committing it to the file. To edit, a separate program had to be used. The editing program relied totally on the PDI file being encoded with a local convention of control sequences. This is not so bad if all Telidon pages are produced on that system, but it makes it impossible to modify pages produced by other institutions, which destroys the porta-

bility of Telidon.

Another problem with *vips* was the necessity to transfer continually between input devices - certain commands were entered from the bitpad, others required bits of numeric information from the terminal keyboard.

After installing and using the basic system for a few months, and after studying other systems, we came to the decision that it was necessary to build a system that provided a natural drawing capability. The artist should be able to "sketch" or rough out an object without "committing" himself to it. It should be possible to "massage" the object thoroughly before drawing the final object - that is, the artist should be able to change the colour, fill density, etc., as many times as required. Furthermore, he should be able to fine tune any of the coordinates of the object until satisfied with the shape.

At any time, the artist should be able to traverse the PDI file and return to work on any object with as much power as he had at its creation.

It is important to provide an artist at a Telidon page creation station, with a text entry capability that is as close to writing as possible. In an effort to achieve this objective, we decided that a simple screen editor should be written.

Usually a screen editor is not a small task. *Usips*, however, takes advantage of the fact that a Telidon page consists of a *single screenful* of videotex information. Knowing that a text buffer will never get larger than a thousand or so characters made possible a very simple but powerful implementation of a screen editor using a two-dimensional array as the main data structure.

Usips relies on the conception of a geometric object consisting of a set of attributes (colour or gray tone, density, texture) and a set of coordinates, which are treated differently by the system. An artist wishing to locate an object is likely to look for a red object or a rectangle, or a red rectangle, rather than searching for a red object with at least one coordinate at (106, 82). Having found that object, the artist is likely to redraw the same object in different colours or fill patterns or he may alter or fine tune some of the coordinates.

Therefore, on *usips*, the Volker-Craig terminal acts as a status monitor as shown earlier in Fig. 1. Selection of a colour, texture or line pattern is like writing to a register. The most recently made change is displayed in standout mode on the status monitor. Whenever the artist begins to actually draw an object, all the status registers are dumped. This status monitor was written using the "curses.h" screen package for terminals under UNIX.

9. The Data Structure

The main data structure for *usips*, the traversible stack, consists of a twice double linked list of records or structures, each of which points to a "line" of Telidon code, as shown in Fig. 3. A line of Telidon code may consist of either a set-status command or a define-object command.

A set-status command, by the definition of the Telidon technology may only set one attribute on the terminal. It may change the colour

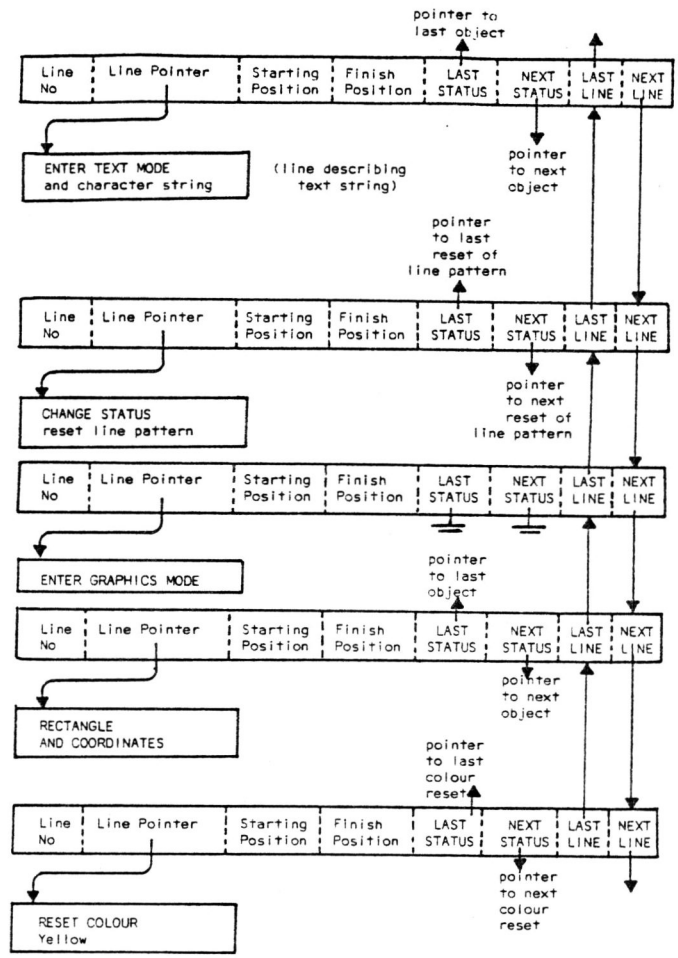

Figure 3 Data Structure for Usips

for example, or it may switch the mode of the terminal from colour to black and white. As it turns out there are about ten different types of status commands. The line contains pointers to the previous line where the same type of status was last set and to the line where it will be next set, besides the usual pointers to the previous line and next line. Each line also carries the coordinates of the logical position of the Telidon terminal cursor before the status command is issued to the decoder and the position the cursor would be in after the command was executed. This is very useful knowledge to have while traversing a PDI file. Knowing the virtual position of the cursor at any position in the file makes the prediction of side effects from insertion, deletion and modification of objects quite simple.

Thus by maintaing a very small block of pointers, it is equally easy to traverse the PDI file backwards or forwards in search of a set of attributes.

A define-object line consists of either 1) one of the 32 ASCII characters which define the geometric primitives available on Telidon and an arbitrarily large set of coordinates encoded according to Telidon conventions, or 2) an ENTER TEXT MODE character, followed by arbitrarily many text characters. It is here that we rely most heavily on the user being able to conceive of a screenful, or partial screenful of text as a type of object.

Note that the diagram shows lines consisting of nothing but a single ENTER GRAPHICS MODE character. These are necessary, of course, so that the geometric objects don't get printed out as ASCII strings. However, it is both trivial for the calling program to insert such lines in the PDI file and for the searching algorithm to ignore them. As well, this type of line is invisible to the Telidon artist.

The traversible stack almost trivializes the writing of a Telidon code optimizer when writing the file although currently, this is not implemented. Nevertheless, optimization is important since the whole videotex philosophy is concerned with storing as much graphic information as cheaply possible.

Support for this data structure results in many run-time requests for storage from the operating system. Space for every record as well as storage for every "line" of code are requested separately.

In the worst cast, our structure requires 30 times as much storage as the information it is maintaining. Despite this, we feel it useful for many reasons. First, it is efficient for searching. From the artist's point of view, it is possible to search a PDI file consisting of a thousand objects backwards or forwards instantly. Secondly, it reflects very naturally both the artist's and the viewer's idea of a Telidon page as a sequence of geometric objects and text. Thirdly, it seems to be reasonable to assume that the average Telidon page will be very small. While users may at first enjoy the colour displays for entertainment, generally the graphics are too slow and very likely to annoy the average user who really wants to get at "fast" information. The same applies to the use of Telidon as a lecturing and teaching aid. In one field trial of videotex technology offering a poison control service, a user whose infant might have just swallowed something dangerous had to watch and wait while a detailed image of a skull and crossbones was drawn on the screen. Such an extended delay is not only annoying but could be tragic.

10. Conclusions

In this paper we have described an interesting and powerful page creation system that runs under the UNIX operating system. Pages can be created and added to a Telidon-like hierarchical database on the same system.

Future research and development efforts include the ability to create, name and later generate complex objects and the use of image analysis to create, from a pixel by pixel representation, pictures that are primarily composed of the high-level graphic primitives of Telidon. Both of these enhancements can significantly improve the quality and quantity of Telidon pages that are produced.

Acknowledgement

The authors acknowledge the support of NSERC under grant number A9290.

References

[1] Sutherland, I.E., Hodgman, G.W., Reentrant Polygon Clipping. Communications of the ACM, Volume 17, Number 1. January 1974.

[2] Bown, H.G., O'Brien, C.D., Picture Description Instructions PDI for the Telidon Videotext System. CRC Technical Note 699-E, Communications Research Centre, Department of Communications, Canada. November 1979.

[3] Chang, Ernest, Design Considerations for the Telidon-APPLE Picture Creation System. Technical Report DCS-16-IR, Department of Science, University of Victoria, Victoria, B.C., Canada. February 1982.