PED: A 'DISTRIBUTED' GRAPHICS EDITOR

Theo Pavlidis
Bell Labs (2C-456)
Murray Hill, NJ 07974

ABSTRACT

PED (Point EDitor) is a program intended for creating technical and other schematic illustrations using the bitmap terminal *Blit* designed at Bell Labs. The major new feature of PED is that it has two parts, one running on the Blit and the other on the host computer. In addition to the usual operations that are common in graphics editors, it allows the user to specify splines, fill arbitrary polygons with texture, rotate objects, etc. The files produced by PED resemble display lists and it is easy to write programs for displaying them on other devices, including a phototypesetter.

KEYWORDS: Bitmap terminals, interactive graphics

Introduction

Graphics editors are programs for creating and manipulating files that describe line drawings. Sutherland's *Sketchpad* [1] is one of the earliest and best known. Recent programs include *Grace* [2] developed at UC Berkeley, and *Juno* [3] developed at Xerox PARC. These exemplify two trends. Grace describes objects by their spatial coordinates while Juno describes objects by procedures. PED (Point EDitor) uses descriptions in space like Grace. It is intended for creating illustrations using the bitmap terminal *Blit* designed at Bell Labs [4,5] although its origin goes back to a "device independent" program [6, pp. 239-243]. PED runs under the UNIX† operating system A major feature of PED is that it has two parts, one running on the Blit and the other on the host computer. Most other editors run either on a single work station or on the host computer using only the low levels graphics routines of their graphics devices. However both parts of PED are of about equal importance. (Coincidentally, both have about the same text size.) For simple pictures the host is used only as a link to the file system and all operations are done on the Blit. As the complexity of the picture and the complexity of the type of operations required increases, the use of the host becomes greater. In this way the user has the quick response of a personal workstation combined with the power of the host machine. (Similar division of labor is characteristic of other Blit programs, such as the text editor *jim* [5].) Additional features of PED that are not always found in graphics editors include arbitrary polygons filled with texture, splines, rotation by any angle, 'beautification' procedures, and hierarchical organization of the objects in the drawing. Figures 1 and 2 show some examples. The first two were drawn entirely with the Blit part. Because the Blit allows multiple windows, one can use PED without losing contact with the rest of the computing environment. (Reading mail, making notes using a text editor, etc.)

PED files resemble display lists and as such can be displayed with little effort on any other graphics device. (Certain textures may not be reproduced this way, but it is possible to specify color in their place.) Currently there is a *troff* filter that allows illustrations created by PED to be phototypeset. Figure 1 has been printed in this way. It is also possible to make a copy of the bitmap of the Blit screen and display it on any raster graphics device. The copies of Figures 2 to 4 were made in this fashion.

† UNIX is a Trademark of AT&T Bell Laboratories.

Data Structures and Object Types

The basic data structure of PED is an *object* that contains the following information: (1) A pointer to an array p_0 of x, y coordinates, which is allocated at object creation time. (2) An integer n denoting the number of points in p_0. (3) A *type* flag which indicates how the object will be plotted and what kind of operations are allowed on it. PED allows five basic types of objects: polygons, splines, filled polygons, text, and families. (4) An integer *size* that has different interpretations depending on the *type*. For example, for text it denotes the point size, while for families it counts the number of members. (5) A *status* flag that is set when an object is modified and reset whenever the file is saved. (6) and (7) Two variables *color* and *pen* with the obvious meaning. (8) A pointer *pp* that is used to point either to an array of object pointers *mem* if the object type is family, or to a string of characters *txt* if the object type is text or spline.

The form of an object depends less on the contents of the *object* structure, than on the way these data are interpreted by the plotting routines, depending on the type. If the type is *spline,* the vertices of p_0 are used as control (or guiding) points. The character string associated with a spline denotes whether a vertex of the polygon is multiple or not. The splines used by PED are parabolic splines of the kind described in [6, pp. 262-269]. In particular, they are tangent to the sides of the polygon and pass through the midpoint of the sides. They also pass through all multiple vertices and are still tangent on the sides. In this way multiple vertices can be used to create corners. If the type of an object is *filled,* then the sides of the polygon p_0 are used as input to a scan conversion algorithm that produces a set of horizontal line segments. Each such segment is overlayed on a textured pattern and the resulting texture is displayed on the Blit screen.

If the type of an object is family, the most impor-

tant part of the data structure is the array *mem* which lists the objects belonging to the family. Since the type of some of the latter objects can also be family there is a facility for building recursively complex objects out of simpler ones.

The display of a polygon (or a spline) does not have to be connected: when the plotting routine finds a point with negative coordinates in p_0 it skips that point and does a "move" (rather than "draw line") to the next point. This feature is essential for allowing filled polygons with holes to be single objects.

Objects are drawn or pointed to with the help of the mouse cursor. For pointing the mouse coordinates are checked against the list of all objects. For families the search is done recursively and if a match is made the searching routine does not return the actual object but the one highest in the hierarchy above it. Similar recursion is used with all editing operation so that families can be handled as single objects.

Host-Blit Interaction

The host has two lists of objects: one of inactive objects H and another of active objects A. The Blit has only one list of objects B. PED tries to maintain the lists A and B as close to as possible. There are two major differences in the form of the object data structure between the host and the Blit. One is that coordinates are kept in floating point form on the host and in integer form on the Blit. The second is that the host points have a third coordinate which used for the same purpose that negative coordinates are used on the Blit. (While negative coordinates have no meaning on the Blit, they do on the host.) The host program knows a scale factor S and a shift vector X, Y. Instead of sending the coordinates x,y to the Blit, the host sends the rounded off values of $(x - X)*S \quad (y - Y)*S$. The inverse transformation is applied to coordinates read from the Blit. When objects are created or edited on the Blit the lists A and B differ until an updating of A from B is performed. It as convenient to think of an object as a display list and of plotting an object as a mapping from a display list to the refresh memory of the Blit, R.

When a file is read, it is placed in the inactive list H. Objects can be displayed on the screen of the Blit in two ways. (1) The (scaled) display lists of the objects in H are transmitted one at a time, plotted on R, and then erased. Since R is not affected by the display list erasure, what is seen on the screen on the Blit corresponds to what is the host and there is no real limit on the complexity or size of the displayed image.† Note that in this case both A and B are empty. (2) A set of objects are transferred from H to A, and then a scaled copy of A is made in B. Since these objects now reside in the Blit, they can be displayed, and also edited.

Any new objects drawn are placed in B. Depending on the extent of modifications either parts or the whole list B is copied to A periodically. Any request that might bring more objects from the host to the Blit initiates the following sequence: B is copied to A and then B is erased; all objects of A are transferred to H; a selection procedure is applied on H and a new list A is created; the new A is copied to B. The typical way to edit a file is to display it on the Blit screen, select interactively a part to be transferred from H to A (and then to B) and then work on the latter. Interactive selection (and most other operations) are possible even though the display lists are not on the Blit because positional information from the Blit can be sent to the host with the inverse scaling transformation. Some editing operations are always performed on the host, but this is transparent to the user (except for a slower response). Other operations, such as copying a group of objects, are performed explicitly on the host. This is how Figure 2 was created. First a dragon was drawn locally, then the remote was selected and a copy of the dragon was made and transformed to create the reflected image. Zoom and pan operations are achieved by modifying S, X, and Y. When a zoom operation is requested and B is not empty, A is updated from B (if necessary), and then A is recopied on B with the new transformation.

Summary of Operations

PED has eight menus and at any given time one of them is displayed on the Blit screen. The names of the menus appear in a small menu controlled by one of the mouse buttons. There is a user's manual describing in detail all the operations. The following is a summary of what is available with an explanation about the implementation of the less common operations.

Basic: Controls communications with the host. It includes a selection that activates the keyboard so that file manipulation commands can be passed directly to the host.

Draw: Contains commands for creating objects. Polygons, splines, text, circles, boxes, ovals, and families. Also for establishing a grid where points of the drawn objects are forced to lie. Three grid meshes are possible and these are expressed in unscaled coordinates so they do not change between sessions.

Reshape: Contains modification operations such as move a point, delete a point, change the type of a polygon into a spline, undo an editing operation, and 'formalize' an object. The latter command changes the location of vertices so that lines whose direction is close to a set of chosen directions (vertical, horizontal, 45 degrees, and 135 degrees from the horizontal) are made to have exactly those directions by modifying their endpoints appropriately.

Refine: Similar to 'Reshape' with commands for editing text, make a line dashed, or double, or heavy, etc. The computation for plotting double lines is done in the host because it requires extensive use of floating point arithmetic and trigonometric functions. Double or heavy lines belonging to objects in the Blit part are marked by tags and are plotted single as shown in Figure 3. When a plot using the host facilities is requested, then such

† The only time where there would be a problem is when the size of a single object exceeds the available space on the Blit, an unlikely occurrence.

lines appear in their proper form as shown in Figure 4.

Move: move an object, copy an object, etc. Also delete objects, and move them so they are aligned or centered.

Shade: contains a selection of 10 textures and an 'unfill' option. Polygons are filled with texture as following: The user selects a texture from the menu and then points to a polygon. The only things that change in the data structure are the type and the color. The plotting routine uses a scan conversion algorithm to produce the filled polygons.

Change size: Contains fixed shape and size transformations, such as reducing the size by a fixed percentage. There is also the possibility to apply an arbitrary size and orientation change by selecting the "rotation" command form the menu.

Remote: Operations such as copying or scaling groups of objects performed directly on the host. Also files can be read into specific parts of the screen, or objects may be selectively written into files.

It would seem that the lack of constraints would make it difficult to drawing 'neat' looking pictures but as Figure 1 shows this is not the case. The use of grids and aligning commands results in 'beautification' of drawings. Of more interest is an automatic beautification procedure that modifies a drawn picture as to satisfy certain forms of constraints. As of this time only small parts of this procedure have been implemented and are used with the 'formal' selection in the 'reshape' menu. It is expected that more extensive procedures will be added in the future.

Conclusions

One conclusion from the experience of developing PED is that the "interactive graphics" routines are the least important part of a "picture editor," if one is interested in drawings of any significant complexity or size. Problems of file manipulation, recursive definition of objects, viewport selection, etc. become dominant, in the same way that they are dominant in a text editor. These problems are more difficult in the case of drawings than they are in the case of text because there is no obvious linear ordering in drawings as there is in text. Splitting the program between two machines provided considerable flexibility in the design but also required many decisions about where things are to be done. Time is an ample resource in the Blit, but space is scarce. The opposite is true for the host machine. In addition, the communication channel between the two machines is a resource that must be used carefully. As a rule, operations that are expected to be infrequent are assigned to the host.

Acknowledgments

George Wolberg helped with the implementation of many of the routines, and in particular those dealing with variable size characters. I have also used code provided by Ken Thompson (creating the double lined objects) and Tom Killian (making copies of the Blit screen). Paul Pavlidis was my first user and created the drawing used in Figure 2. I have also received useful comments on the program or the manuscript from Doug McIlroy, Steve Mahaney, Lorinda Cherry, and Mary Bittrich.

References

[1] Sutherland, I. E. "Sketchpad: A Man-machine Graphical Communication System," in *1963 Spring Joint Computer Conference*, reprinted in *Interactive Computer Graphics*, H. Freeman, ed., IEEE Computer Soc., 1980, pp. 1-19.

[2] DeRose, A. D. "Status Report on the Graphical Curve Editor (GRACE)" *Tech. Rept. Univ. of California, Berkeley, Graphics Lab, Sept. 1982* AD-A120664

[3] Nelson, G., personal communication.

[4] Pike, R. "Graphics in Overlapping Bitmap Layers," *ACM TOGS*, **2** (1983), pp. 135-160. Reprinted in *SIGGRAPH'83*, pp. 331-356.

[5] Pike, R. "The Blit: A Multiplexed Graphics Terminal," *Bell System Technical Journal*, (in press).

[6] Pavlidis, T. *Algorithms for Graphics and Image Processing*, Computer Science Press, 1982.

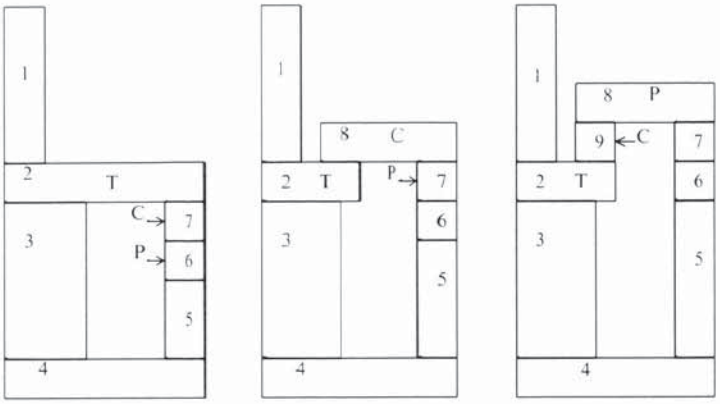

Figure 1

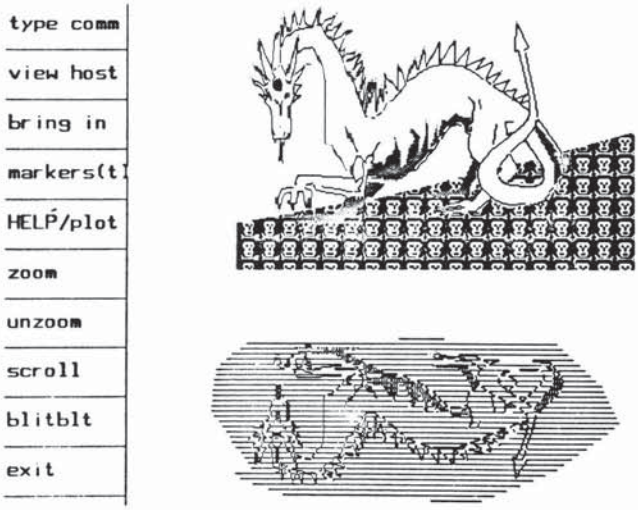

Figure 2

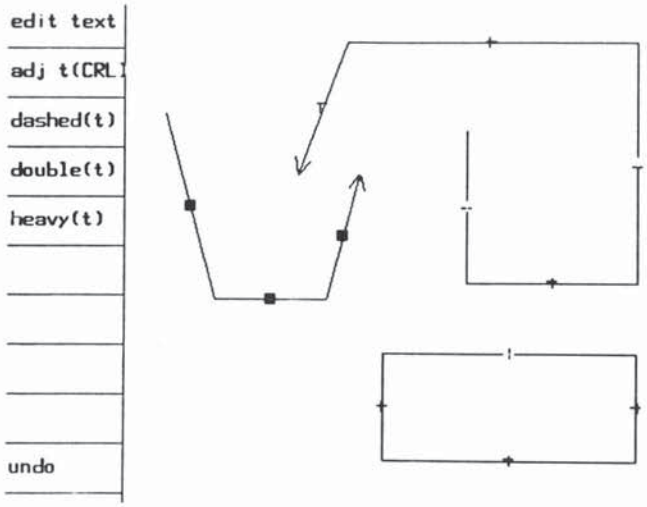

Figure 3

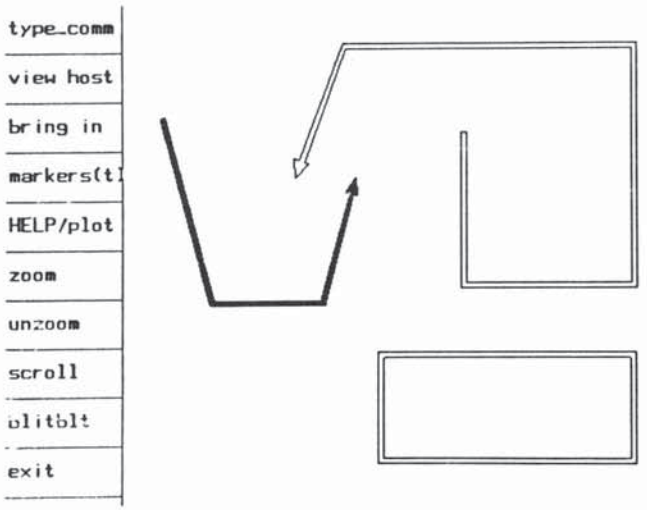

Figure 4