A note on compiling display file from a data structure

By N. E. Wiseman*

A display file compiler developed for a particular project in computer graphics is described. It is designed to operate on a small computer and attempts to compile display files of minimum length. To this end a mechanism for automatic cataloguing has been implemented which recognises common substructures and compiles them into display file subroutines.

(First received March 1968)

The use of data structures to represent pictorial information began with Sketchpad (Sutherland 1963). As a graphical communication experiment Sketchpad demonstrated many of the facilities required for use at a general purpose interactive terminal, but it required most of the resources of a very big computer (TX2 at Lincoln Laboratory) for even a quite simple task, and the program was in general unable to manipulate application oriented data. Many experiments, making use of techniques first exposed by Sketchpad, have since been carried out with the object of developing graphical interaction methods as an economical approach to computer aided design.

In some applications graphical interaction is appropriate for only a small (though important) phase of the design problem and it may be possible to implement the entire mechanism for this phase on a relatively small and cheap computer. Other phases of the design problem may be run efficiently on a larger machine to which the small computer is connected as a satellite. This approach is being tried in an experimental computer-aided circuit design project at Cambridge. Graphical interaction is for the purpose of inputting and editing data structures which represent schematic diagrams of electronic circuits. Building and editing a data structure is carried out wholly in a small computer (a PDP-7 with Type 340 display) which is attached as a satellite to the Cambridge Multiple Access System (operating on the Titan, a prototype Atlas 2). Circuit analysis programs operate in the Titan on data sent from the PDP-7 and no interaction is permitted (or desired!) with the running of the analysis phase. It is hoped to show that this approach results in cheap and efficient operation with, nevertheless, all (or nearly all) of the desired graphical interactive facilities being present. Although the primary goal is computer-aided circuit design, it is proposed to attempt other design exercises in a similar manner, and a suite of programs known as PIXIE (standing for nothing in particular) of reasonably general usefulness is being constructed for the PDP-7.

Data structures in the PDP-7 are generated with RSP (Wiseman and Hiles, 1968), a ring structure processor specially designed to operate in a computer with a small core store. Structures may be built cooperatively by user programs and by structure generating routines provided in the PIXIE system. Parts of a data structure may represent pictorial data and it is the task of the compiler described in this note to generate a sequence of display commands (a display file) from such pictorial data which will produce a corresponding image on the screen of the CRT display. The data paths are shown in **Fig. 1.** The traversing routine scans the source structure, performs a simple syntax analysis on what it finds and issues a description of the picture in terms of points and lines. A name parameter N starts the traverse from a selected spot. The transformation routine is a user-supplied subroutine which generates a 'view' of the picture by scaling, rotating and translating it according to the set of T-parameters. Finally the display code generator computes how the screen window defined by the W-parameters intersects this view and generates a sequence of commands to cause the hardware to display the result.

Data structures presented to the compiler will generally comprise pictorial data, built by PIXIE, and user data, built by the user. Although arbitrarily many element

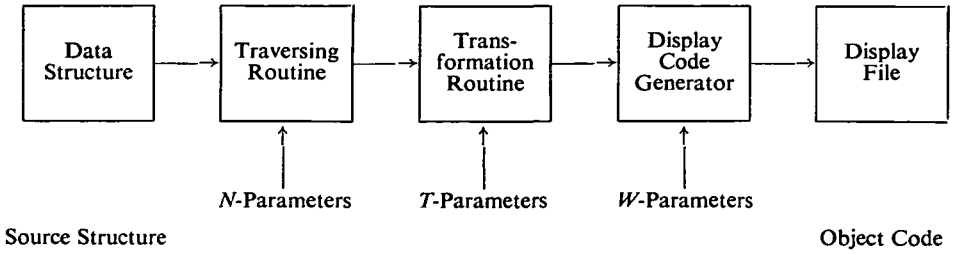

Fig. 1. Data paths through the compiler

* *University Mathematical Laboratory, Corn Exchange St., Cambridge.*

Table 1

Extract of RSP syntax

⟨relation⟩ ::= **son** | **father**
⟨element⟩ ::= ⟨pixie element⟩|⟨user element⟩
⟨pixie element⟩ ::= **point** | **line** | **instance** | **subpicture**
⟨element relation⟩ ::= ⟨relation⟩⟨element⟩|⟨element relation⟩⟨relation⟩⟨element⟩
⟨path⟩ ::= ⟨element⟩⟨element relation⟩
⟨data structure⟩ ::= ⟨path⟩|⟨data structure⟩⟨path⟩
⟨point relation⟩ ::= **isolate**|⟨point relation⟩**father line**|⟨point relation⟩**father instance**
⟨point path⟩ ::= **point**⟨point relation⟩
⟨line relation⟩ ::= **son point son point**|⟨line relation⟩**father subpicture**
⟨line path⟩ ::= **line**⟨line relation⟩
⟨instance relation⟩ ::= **son subpicture son point**|⟨instance relation⟩**father subpicture**
⟨instance path⟩ ::= **instance**⟨instance relation⟩
⟨subpicture relation⟩ ::= **isolate**|⟨subpicture relation⟩⟨relation⟩**instance**|⟨subpicture relation⟩**son line**
⟨subpicture path⟩ ::= **subpicture**⟨subpicture relation⟩
⟨pixie path⟩ ::= ⟨point path⟩|⟨line path⟩|⟨instance path⟩|⟨subpicture path⟩
⟨pixie data structure⟩ ::= ⟨pixie path⟩|⟨pixie data structure⟩⟨pixie path⟩

types are permitted, only the four types issued by PIXIE routines are noticed by the compiler, and only certain relations between these four types cause the compiler to generate any object code.

The four PIXIE element types are point, line, instance and subpicture and the extract of RSP data structure syntax given in **Table 1** will indicate the way in which pictorial data are represented. The definition of a user element is not necessary in order to understand picture structures and is omitted in Table 1. User elements are invisible to the display file compiler.

In RSP the relational features are represented by rings which associate elements with one another and the values (properties) are stored in the data areas of their respective elements. For example, the coordinates of a point might be regarded as the value of the point and would then be recorded in its data area, while the fact that the point was the endpoint of a line would be shown by a ring connecting it to the element which represented the line. A ring in RSP may pass through any number of elements, one (and only one) of which is the 'owner' of the ring. The owner is known as the **father** of all the other elements on the ring. Each such element is the **son** of the owner. The vocabulary word **isolate** indicates that the corresponding element can exist without relational features at all (i.e. no rings pass through it).

As a simple example of a structure obeying the syntax above refer to **Fig. 2.** The notation follows that proposed for the ASP data structure system (Gray, 1967) in which boxes represent values of elements, and triangles, circles and lines show the relations between elements. For our purpose it is sufficient to define the father-son relation as a path ⟨element⟩⟨triangle⟩⟨circle⟩⟨element⟩.

Thus P1 is the father of L1, P2 is the father of L1 and L2 and so on.

The structure represents two occurrences of a pair of lines L1 and L2. It is interpreted as follows: Two lines

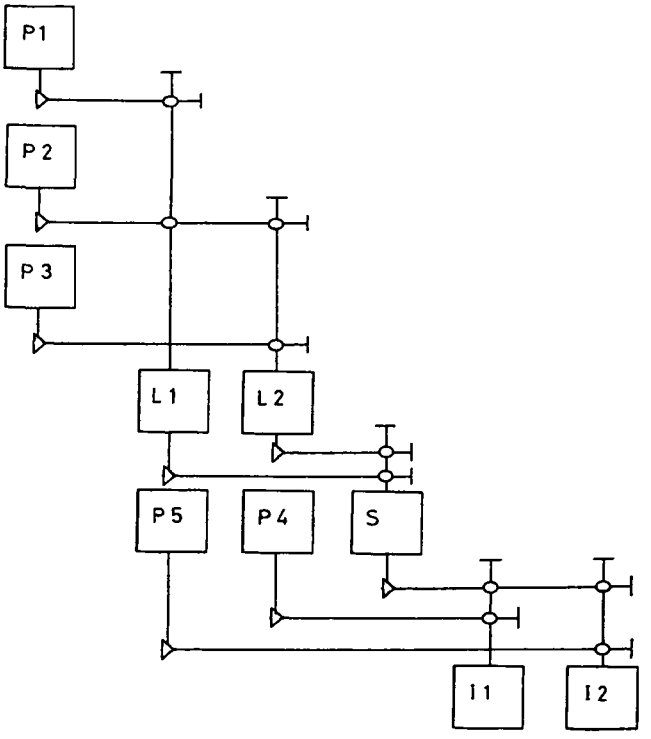

Fig. 2. Example of a data structure

L1 and L2 are defined by their end points P1, P2, P3. The lines are collected together as a single entity into subpicture S. An occurrence of S is represented by the instance I1 which relativises the internal points of S (i.e. P1, P2, P3) to the point P4. A second instance of S relative to another point, P5, is represented by I2.

It will be clear how additional structure could be set up to represent, say, a subpicture S2 comprising a new line L3 together with the instances I1 and I2. Any instances of S2 would then carry further point elements

to which all points inside S2 would be relativised. In this way subpictures can be nested to any depth and any number of occurrences of each subpicture are permitted. An additional feature is that the position in space of an instance of a subpicture of arbitrarily many constituent parts is determined by the single point element associated with that instance. Moving a picture part (for example during tracking) is thus accomplished simply by altering the value of the appropriate point element and then recompiling the picture.

The traversing algorithm

The compiler is entered with an RSP name parameter specifying a selected instance element in the data structure. This parameter identifies a pointer to the selected instance which is moved over the structure by the traversing routine. We refer to this pointer as an *M*-pointer and to subsidiary pointers which take part in the traverse as *F*-pointers.* As the *M*-pointer moves over the structure it sends the *F*-pointers (in a manner of speaking) on errands to gather information about the picture. At a point of recursion we can think of the *M*-pointer as replicating itself and sending its progeny on in its place. Different replication schemes yield different routes through a given structure and hence different orders of retrieving items of information about the picture. It has been a particular challenge to discover a scheme for which the order of retrieval is in some sense optimal. The design of the display hardware determines, to a large degree, what constitutes an optimal sequence and the relevant features of the 340 display, which are given in the appendix, provide the background to what follows.

A natural unit of display code is a display file subroutine, and since the object is to generate compact display files the compiler must detect repeated sequences of display commands and write them as display subroutines in consecutive registers of core (no gaps).

Candidate structures for compiling into display subroutines are suggested by the picture syntax given earlier. For example, a particular subpicture element may have several relationships of the sort

⟨subpicture relation⟩::=⟨subpicture relation⟩**father instance**

representing the fact that this subpicture is repeated several times in the picture. Similarly (although less obviously) an instance element may have several relationships of the sort

⟨instance relation⟩::=⟨instance relation⟩**father subpicture**

meaning that several subpictures share the use of this instance. A subroutine which is used by other routines has an analogy with a father being 'used' by his sons (to bring them into existence) and thus it seems reasonable to associate display subroutines both with subpictures and with instances (points and lines have been excluded from consideration because of the relative simplicity of their compiled form). The formats chosen for these two subroutines are shown below:

```
instance,      DDS end instance
               DJS instance1
               ⟨name of instance⟩
instance1,     DDS ⟨save address⟩
               ⟨invisible vectors to position beam⟩
               ⟨parameter word for scale, intensity &c.⟩
               DJS ⟨subpicture⟩
               ⟨parameter word to set subroutine mode⟩
end instance,  DJP
subpicture,    DDS end subpicture
               ⟨parameter word to set subroutine mode⟩
               DJS ⟨instance⟩
               ⟨parameter word to set subroutine mode⟩
               DJS ⟨instance⟩
                 .
                 .
                 .
               ⟨parameter word to set vector mode⟩
               vector
               vector
                 .
                 .
                 .
               ⟨parameter word to set subroutine mode⟩
```

end subpicture, DJP

The 2nd, 3rd and 4th words in the instance subroutine are for the purpose of identifying a lightpen hit in the display file.† The address of the word carrying the name is written to core by the DDS and may be retrieved by the program if a pen hit stops the display anywhere in the subpicture. The name is a duplicate of the name of the instance element in the data structure (known as the *atname* in RSP). Different save addresses are used for different levels of instance so that a hierarchy of names is available to the pen hit routine. The purpose of the remaining words in the instance and subpicture routines will be evident. Recall, however (see Appendix), that the number of vector words required to draw a given line varies, according to the length of the line, from 1 to 8. The space required for a display subroutine can therefore only be ascertained when the compiler has retrieved enough information from the structure to define fully the length of every line in the subroutine as well as just the number of lines and calls to other routines. The compiler is required to write display file directly to its place in core and it must determine the length, if not the content, of each subroutine before starting the next in order to ensure that no gaps are left (positive or negative) between routines.

Fig. 3 shows the traversing algorithm. An explanation of its action will now be given.

The routine enters itself (recurses) each time it

* *M* and *F* stand for Mouse and Flea respectively. Ross (1964) uses the terms in roughly the same way.

† This method of identifying a pen hit was first suggested by J. V. Oldfield of Edinburgh University (private communication).

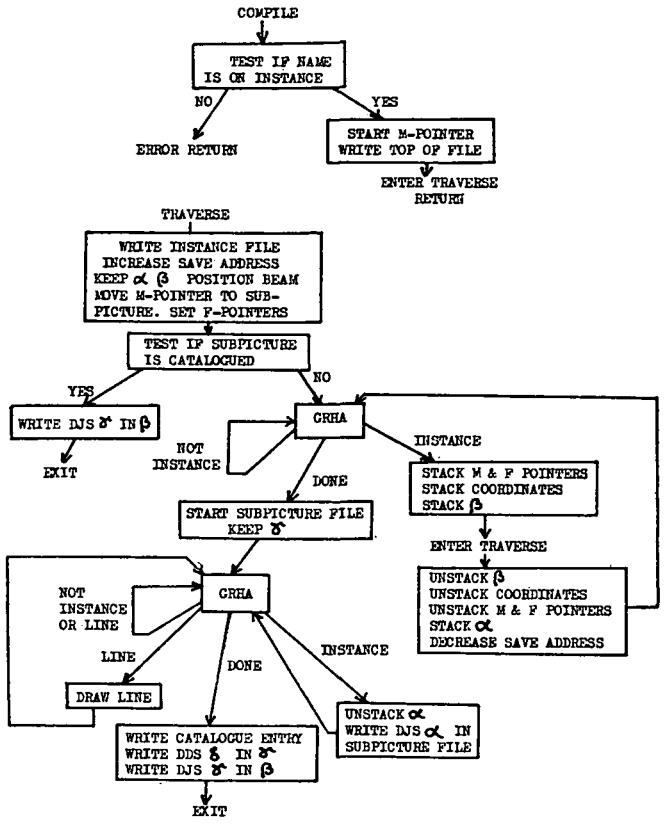

Fig. 3. Traversing algorithm

encounters an instance element, and exits each time a subpicture file is completed. An instance subroutine is written at each entry but the recursions delay the writing of a subpicture subroutine until all the relevant data has been collected. The variables α, β, γ and δ refer to the start of an instance subroutine, the address in an instance routine where a subpicture is called, and the start and end of a subpicture routine, respectively. The two boxes labelled GRHA are GOROUND routines (Sutherland, 1963) operating on the relational features of a subpicture element. Every element which fathers the subpicture is visited and its type and value are collected by a number of *F*-pointers. The type is used to steer the *M*-pointer as shown in the figure. A system of cataloguing is provided to shorten the amount of display file required for a given picture. It makes use of the fact that some subpicture elements in the data structure may represent entities within which no structural detail is needed. In the circuit analysis project, for example, there is no need to refer at each occurrence of a resistor symbol to the detailed arrangement of points and lines which make it up. In such cases a precompiled display subroutine can be provided, to be referenced from the display file each time the subpicture is required. Such a subpicture is said to have been catalogued and three cataloguing actions are distinguished:

(*a*) Permanently catalogued. A subpicture element is permanently catalogued if the structure from which the catalogued subroutine was compiled is no longer connected to the subpicture element. The resistor symbol mentioned above would be permanently catalogued.

(*b*) Temporarily catalogued. Any subpicture which is used in more than one instance should in principle profit from the cataloguing action and compile into a subroutine which is called by the several instance routines. This is in fact done, but since the length of the display file may alter from one compilation to the next (if, for example, something is tracking or the window is moving), entries in the temporary catalogue are deleted each time the compiler is entered. A temporarily catalogued subpicture thus has a short lifetime, but is otherwise similar to a permanently catalogued subpicture.

(*c*) Uncatalogued. An instance which is being tracked may cause an edge violation and then have to be recompiled as a consequence. When this happens the particular instance is disregarded by the cataloguing mechanism and compiled independently. When zooming or rolling the window through the picture this uncataloguing action is applied to everything, except permanently catalogued subpictures. Two data words in each subpicture element in the data structure are used to hold, respectively, the catalogue status of the subpicture CST, and the entry point for any corresponding display subroutine, CEP. The user initially sets CST = 10000 for a permanently catalogued item and CST = 0 otherwise. An indicator word, CMK, is maintained in the compiler to validate the cataloguing mechanism. A digit in one of the data words in an instance element causes the compiler to disregard the catalogue status until the particular instance is completed.

The catalogue is operated as follows. Each time a subpicture is encountered by the compiler, its CST is compared with both CMK and a constant (10000). If equal to either, the subpicture is already catalogued and CEP is used to give a value for γ (refer to Fig. 3). If not equal, compiling continues and when the subpicture file is completed γ is written to CEP and a masked version (mask = 7777) of CMK is copied to CST. The subpicture is then temporarily catalogued. Before returning control to the user CMK is increased by 1 and masked with a constant (7777) to unset the temporary catalogue in preparation for the next entry to the compiler. The uncataloguing digit in an instance causes the compiler to increment CMK until the instance is completed (the uncataloguing digit is also reset by the compiler to restore the normal cataloguing action on the next entry). Global uncataloguing is done by writing the constant 10000 to CMK before an entry to the compiler. CST can then only be equal to CMK for permanently catalogued items. Note that, owing to the method of updating CMK, normal cataloguing action is restored automatically for the next entry.

The transformation routine

The repertoire of transformations required in this routine depends on the particular application for which

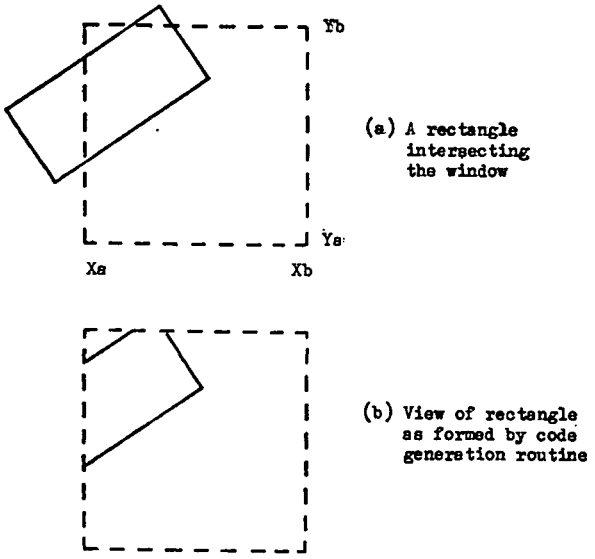

Fig. 4. View through the window

PIXIE is being used. For this reason it is at present not part of the compiler but is supplied by the user as required. It is entered once for each point element which specifies the position of an instance and once for each line element in a subpicture. On each entry *F*-pointers are suitably stationed on the data structure to specify the operation required.

No assumptions are made about whether the picture information is in 2 dimensions or 3 or whether the coordinates are given in parametric form or not. The user writes what words he likes in the data area of picture elements and interprets them as he likes with the transformation routine. However, the output from this routine, representing a view of the picture, is used by the code generator to write words in the display file and it is assumed that this view is represented as Cartesian points and lines in 2-space.

The display code generator

The view generated by the transformation routine is positioned so that the required part falls over a window formed by the CRT screen. In some applications not all of the screen is available for this purpose (for example, if several non-overlapping views are displayed simultaneously or if an area of the screen is reserved for the use of light-button controls) and thus we introduce four parameters to specify the window edges, *Xa*, *Xb*, *Ya* and *Yb*. These define two vertical lines $X = Xa$ and $X = Xb$, and two horizontal lines $Y = Ya$ and $Y = Yb$ and are chosen so that $0 \leqslant Xa < Xb \leqslant 1023$ and $0 \leqslant Ya < Yb \leqslant 1023$. The window is the rectangle bounded by these lines. The code generation routine writes display commands for that part of the view which is inside the window. A rectangle intersecting the window edges as shown in **Fig. 4a**, for example, would be displayed as the sequence of visible and invisible lines shown in **Fig. 4b**. A flow chart for the code generation

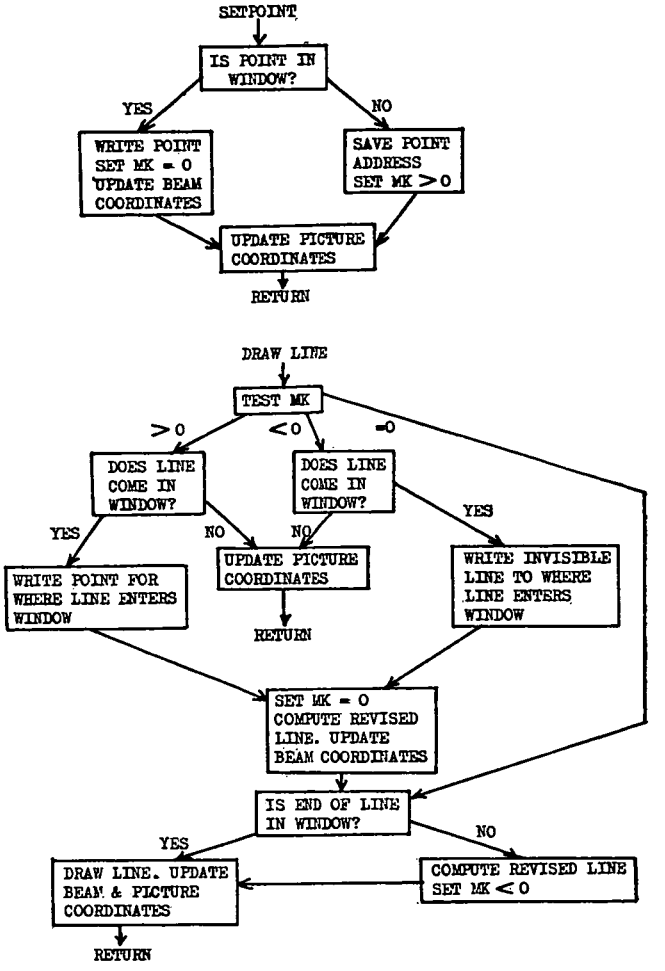

Fig. 5. Display code generator

routine is given in **Fig. 5**. The determination of whether a given line intersects the edges of the window is remarkably tedious, and no alternative was found to solving explicitly for the points of intersection. An automatic scissoring capability such as is found on one or two more modern displays would have been a most useful facility.

Prior to writing any display subroutines the code generation routine sets up a small segment of display file known as the 'top of file'. This consists of a few commands to position the beam for the first instance (SETPOINT in Fig. 5), a subroutine jump to the first instance, and a stopcode. The compiled picture is displayed by starting the display at the first word in the top of the file. A stopcode interrupt occurs at the end of the picture and further action is taken by PIXIE as necessary.

Miscellaneous comments

The compiler is a part, in fact a rather small part, of the PIXIE system and a few words about the operation of the display package in PIXIE will not seem out of

place. Successive entries to the compiler with different *N*-parameters can be used to generate a sequence of display files for different pictures. These pictures can be superimposed on the screen by displaying the individual display files in round robin fashion.

A pen hit from some picture part gives immediately information about its identity from the list of names saved by DDS instructions in instance subroutines. A level marker in the name list is provided to indicate the 'scope' of the hit (i.e. whether a window, or a house of which the window is a part, is intended when the window is pointed at by the user). The picture is then recompiled with the selected instance set to produce a blinking effect on the screen. Thus the scope of the hit is fed back to the user and he can, for example, alter the scope (by moving the level marker) and point again, or track the selected instance about on the screen, or take some other action. Tracking is done by copying the relative pen coordinates into the appropriate point element and then recompiling the picture. Compiling is normally done with interrupts enabled so that tracking (for example) can continue even though the tracked instance may not keep up with the motion of the user's hand. If uncataloguing mode is selected (CMK = 10000) the compiling is slow and the display file is long, but the picture remains accurately registered in the window. If normal mode is selected an instance may be tracked outside of the window and the fact will not be detected until an edge of the screen is violated. The interrupt arising from the edge violation, however, can be used to uncatalogue the instance being tracked so that, after the next compilation, a correctly registered picture is restored. In applications where this action can be tolerated substantial savings in compiling time and length of resulting display file are normally obtained. These are the applications to which PIXIE is oriented.

Appendix

The display hardware

The Digital Equipment Corporation type 340 is a digital incremental CRT display employing a 16-inch tube with a useful area $9\frac{3}{8}$ in. $\times$ $9\frac{3}{8}$ in. Deflection currents which direct the beam are controlled through digital-to-analogue converters from two 10-bit registers (x and y).

The display is set in motion by an instruction from the processor, which loads an address into the Display Address Counter (DAC) and issues a start signal. Once started, however, the display obtains its own commands from the PDP-7 core store through a direct-access channel, without disturbing the processor, until it requests processor attention by setting one of its four device flags.

Commands are normally obtained sequentially from the display file under the control of the DAC, which is incremented after each access, but the Cambridge machine has a Type 347 Subroutine Interface, which enables the display to break sequence, and call 'subroutines', without processor intervention (see 'subroutine mode' below).

Commands are interpreted according to the current state of a 3-bit mode register, which is cleared to zero on starting and may be set, according to rules described below, so as to select the mode for the *next* command. Six modes are available on the Cambridge machine. Numbers in brackets after each mode are the contents of the mode register when in this mode.

Parameter mode (0)

This mode in which the display necessarily starts, is used to set the scale and brightness of the display, to enable or disable the lightpen and to set the mode for the next command. It can also be used to stop the display and set the STOP FLAG to request processor intervention. Apart from mode, none of the parameters is altered unless an associated 'permit' digit is set, so that scale, brightness and lightpen status need not be specified in every parameter word. There are eight brightness levels and four scales (X1, X2, X4, X8); change of scale affects only Increment, Vector, and Vector Continue modes, where it alters the spacing between consecutive points, up to 0·07 in. at the largest scale.

Point or x–y mode (1)

This mode enables either the x or y coordinate register, according to bit 1, to be loaded with the value given in bits 8 to 17. If bit 7 = 1 a spot is 'intensified' at the resultant coordinates. The mode register is set in this as in parameter mode, and the lightpen status may also be altered. A pair of point mode words will, in general, be used to set the starting position for a picture-part drawn in the other, more powerful modes.

Vector mode (4)

In this mode, the hardware for drawing straight lines is activated. A single vector word can produce a line segment (vector) with Cartesian components of up to 127 in each direction, the two components being given (as sign + magnitude) in bits 2–17 of the word. Thus from 1 to 8 words may be required to draw a given line, depending on the length of the line. If bit 1 = 1, the line is intensified at the present brightness; otherwise the beam is suppressed completely. Bit 0 is called the 'escape' digit, and if set to 1 causes the mode register to be cleared (forcing parameter mode) at the end of this vector. The mode otherwise remains set to vector. 'Escape' also occurs if either coordinate overflows while the vector is being drawn ('edge violation'); this also

stops the display and sets either the HORIZONTAL or the VERTICAL OVERFLOW (EDGE) FLAG, as appropriate.

Vector continue mode (5)

The format of the command is identical to that for vector mode, but on completing the vector specified by the given components the display starts again, drawing the same vector repeatedly. When the line reaches an edge, the display escapes to parameter mode and demands a new command. No edge flag is signalled to the processor. The escape digit (bit 0) is ignored.

Increment mode (6)

This mode can be used to display four points per word, each point being specified by x and y steps from the previous point. The step is a single increment at the current scale. Greater control is obtained over precisely which points are intensified than in vector mode, and increment mode is particularly useful for highly curved parts of a drawing.

Edge violation may occur in this mode as in vector mode, with similar effect.

Subroutine mode (7)

Commands obeyed in subroutine mode enable the display to execute jumps and enter subroutines and also set the mode for the next word as in parameter and point modes. Bits 5 to 17 hold an address. Bits 0 and 1 comprise the 'op-code': code 0 is spare and the other three have effect as follows:

Display Jump (*DJP*) (*Code* 2)

The display jumps unconditionally to the given address, setting the given mode.

Display Jump and Save (*DJS*) (*Code* 3)

The current contents of the DAC is preserved in the 'Address Save Register' (ASR), and the display then jumps to the address given. At the same time a flip-flop called 'SAVE' is set; while this remains set, a return jump occurs automatically when the display next 'escapes' from vector, vector continue, or increment mode. This instruction thus provides a single-level closed subroutine facility.

Display Deposit and Save (*DDS*) (*Code* 1)

This instruction does not cause a jump but transforms the content of the ASR into a DJP instruction selecting parameter mode, which is then written to the address given in the DDS instruction. It also clears 'SAVE', preventing automatic exit on 'escape'. Having written a DDS at the head of a subroutine, the programmer is free to use all the display facilities *within* the subroutine—he may cause escape, or call other subroutines without error. Final exit is achieved by obeying in subroutine mode the DJP written away by the DDS instruction. Nesting of subroutines to any depth is possible by use of DDS.

The lightpen

This takes the usual form of a tube with shuttered aperture, connected by a flexible fibre-optic pipe to a photo-multiplier. When it has been 'enabled', a sufficiently bright source of light within the field of view will stop the display and set the LIGHTPEN FLAG. The display stops in such a way that it can 'resume' precisely where it left off (e.g. in the middle of a vector) if the processor issues the appropriate IOT instruction.

Interaction with the processor

By means of IOT instructions the processor can obtain the following information at any time, though it will usually do so when the display is stopped requesting attention:

The content of the DAC.
The content of the ASR and 'SAVE' flip-flop.
The contents of the coordinate registers, except bit 9 of each.

The processor can also discriminate between the four flags and issue commands as follows:

Load DAC and start (in parameter mode), clear flags.
Start (in parameter mode), clear flags.
Resume (in current mode), clear flags, disable lightpen.
Clear flags.

The resume command is normally useful only after a pen flag, since the display is necessarily in parameter mode after a stop or edge flag. It disables the lightpen, so as to prevent multiple interrupts from the same object.

References

SUTHERLAND, I. E. (1963). Sketchpad, a man-machine graphical communication system, *AFIPS Proc. Spring Joint Computer Conference*, p. 329.

WISEMAN, N. E., and HILES, J. O. (1968). A ring structure processor for a small computer, *Computer Journal*, Vol. 10, p. 338.

ROSS, D. T. (1964) AED JR: An experimental language processor, *M.I.T. Electronics Systems Laboratory Technical Memorandum ESL-TN-211*.

GRAY, J. C. (1967). Compound data structure for computer aided design; a survey, Proc. ACM 22nd Nat. Conf., p. 355.

CROSS, P. (1967). A logic drawing board for the PDP7/340, *Computer Bulletin*, Vol. 11, p. 237.