ISSN 0381-9019

A DRIVER FOR RASTER-LIKE PLOTTING DEVICES

BY

UDAY G. GUJAR

AND

J. ANTHONY FITZGERALD

TR76-013, DECEMBER 1976

A DRIVER FOR RASTER-LIKE PLOTTING DEVICES

BY

Uday G. Gujar

Computing Centre
and
School of Computer Science
University of New Brunswick
Fredericton, N.B.

J. Anthony Fitzgerald

Computing Centre
University of New Brunswick
Fredericton, N.B.

TR76-013, December 1976

ABSTRACT

Electrostatic plotter type devices, CRT or hardcopy, are basically printers which allow the "paper" movement only in one direction. As a result, the entire display file has to be created before the process of generating the display can begin. This arrangement differs significantly from the X-Y plotters where the "plotting pencil" and/or paper may be moved in any direction. An algorithm for generating a display on the electrostatic plotter type devices is described in this paper. This algorithm is based upon creating and maintaining a vector file consisting of end points of visible vectors, sorting these vectors, and then generating the plot - a strip at a time. Only the points that lie within the strip are calculated. The details of implementation and the data structure used are discussed. There is no restriction on the size of the display that can be generated; in fact, plots of size 100" by 20" have been produced using this algorithm. No a-priori knowledge of the extent of the plot is required. The algorithm has been designed to operate in a device independent computer plotting system and has worked very satisfactorily without imposing any restrictions on the users.

Key Words and Phrases: graphics, electrostatic plotters, raster displays, vector generation, data structure, sorting.

CR Categories: 8.2, 4.41

1.0 INTRODUCTION:

A device independent computer plotting system [1,2] has been in use at the authors' installation for a number of years. This system consists of a set of FORTRAN callable subroutines and provides the programmer the convenience of switching from one plotting device to another without making any changes to his program.

The logic of the plotting system is such that the device driver is called once for each action of a "plotting pencil" (see Fig. 1). The call indicates the X-Y coordinates of the point to which the plotting pencil should be moved and whether it be "raised" or "lowered" before and/or after the motion. After the display is defined, the end of the plotting frame is indicated by calling the end of plot routine.

The subject of this paper is the interface to electrostatic plotter type devices (shown in dotted line in Fig. 1). It was desired to include the electrostatic plotter in the package in such a way as to be absolutely transparent to the users; this has been achieved without any exceptions.

Though the programs developed are designed to work under the framework defined above, the concepts and methods used are totally environment independent.

2.0 GENERAL COMMENTS:

There is a fundamental difference between an X-Y plotter and an electrostatic plotter. In the case of an X-Y plotter, the plotting pencil may be moved anywhere in the entire display area. The result is that a vector can be drawn on an X-Y plotter as it is created by moving the plotting pencil and/or paper.

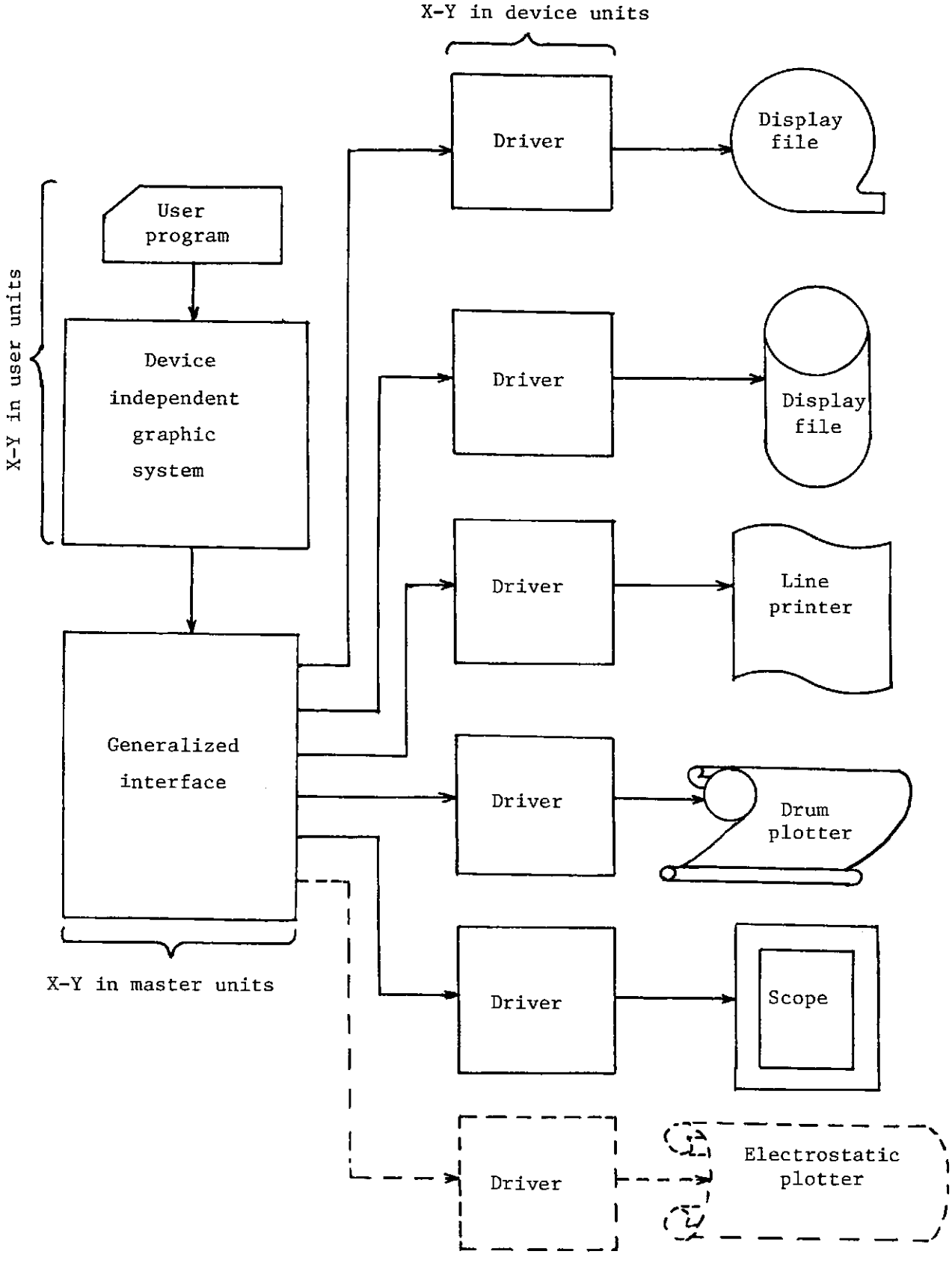


Fig. 1: Computer plotting system

However, electrostatic plotter type devices, CRT or hardcopy, are basically printers. As such they print a line of information and allow the "paper" movement only in one direction. The implication is that all the vectors crossing a line must be known before that line can be printed. This further suggests that the entire display be available before the process of generating a display can begin.

A second basic difference is that an X-Y plotter is a device specifically designed to draw a vector while an electrostatic plotter represents vectors by individual points arranged in horizontal (or vertical) line segments which criss-cross the actual vectors. Thus, each vector has to be translated into these line segments.

An obvious method is to generate the entire display in memory in a bit pattern form and at the end send it to the electrostatic plotter in the required raster format. This procedure is used in [3]; while a slight modification of this scheme where a rotating drum is used for the memory is reported in [4]. However, the amount of memory required in this technique is extremely large. For example, if one wants to produce a 20" by 20" picture, it would require, assuming 1/100" resolution in X and Y direction, 4 million bits. Large plots of size, say 100" by 20", would require 20 million bits!

An improvement over the above method is found in [5] where only one tenth of the picture is stored in a memory buffer. The complete image is generated from the X-Y file ten times and only those parts corresponding to that portion of the image to be plotted are stored in the buffer area.

Jordon Jr. and Barrett report an ingeneous algorithm [5] which removes the above drawbacks. The algorithm suggested in this paper is very similar to that reported in [5]. However, the implementation details are significantly different and the technique employed does not impose any restrictions whatsoever on the size of the plot that can be generated. In fact, the number of horizontal line segments in a plot does not have to be known a-priori.

3.0 OVERVIEW OF THE ALGORITHM:

The conventions for the X and Y axes are shown in Fig. 2. These conventions are chosen so as to be compatible with the ones used for the drum plotter at the authors' installation. As can be seen from Fig. 2, the limit on the Y-axis is the width of the paper while the X-axis is limited by the length of the roll of the paper.

There are two phases to the algorithm. These are shown in Fig. 3. The first phase involves generation of the vector file. In this phase, the endpoints of the visible lines are stored in a file called the vector file. A simple initial pre-sort is performed on these endpoints so that the starting or the first point has the larger X coordinate value. It should be noted that if a point is to be plotted, it is stored as a vector whose endpoints are the same.

Most of the work is done in the second phase which is initiated by the command to end the current plot frame. In this phase the vectors in the vector file are first sorted in the descending order of the starting X coordinate value. After the sort is complete the display generation begins as a series of strips. These strips are as long as the width of the paper and of width Δx which may

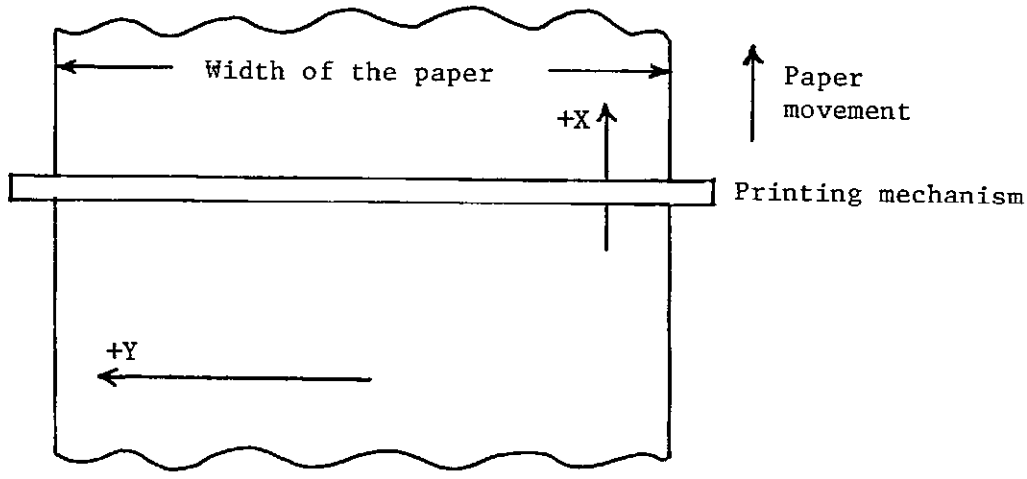


Fig. 2: Convention for X-Y axes

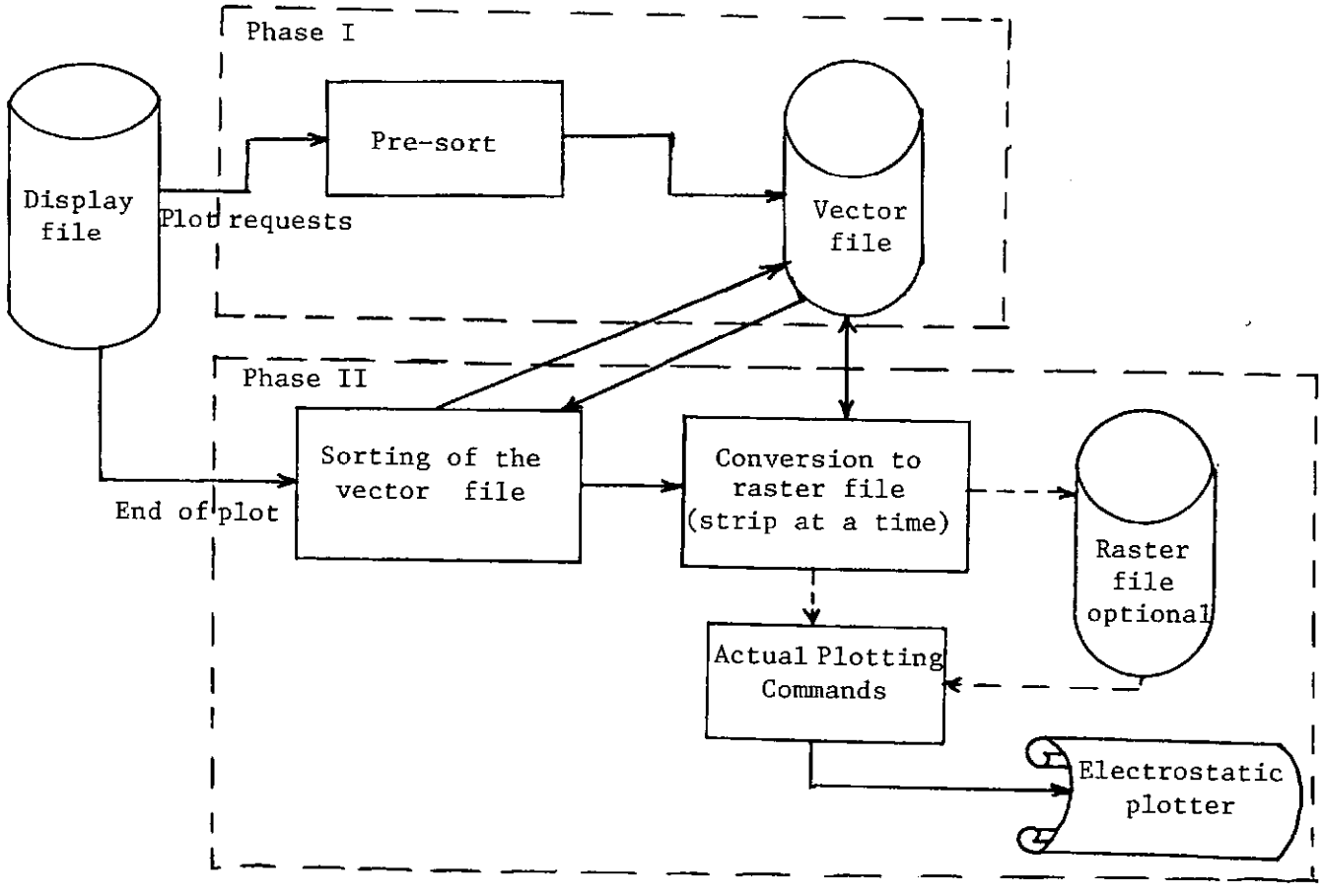


Fig. 3: Overview of the algorithm

be typically chosen as one inch (see Fig. 4). The number of strips, m, that make the display is given by

$$m = \lceil (n \div \Delta x)$$

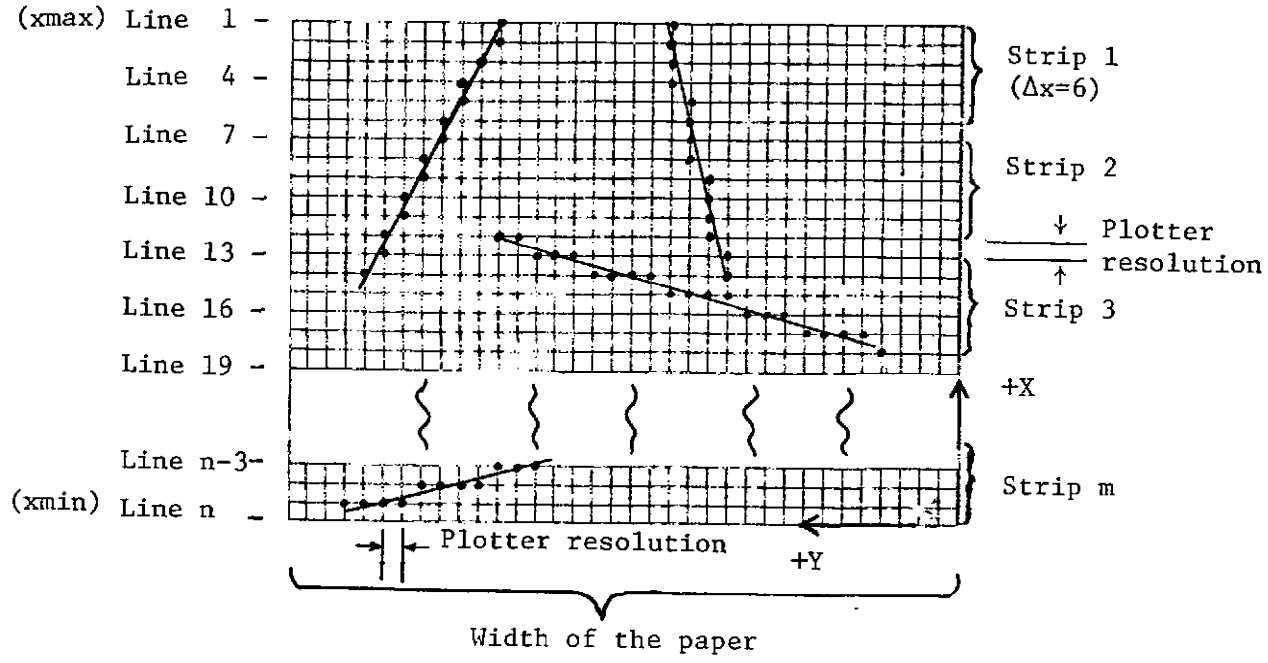
where, $\lceil$ denotes the ceiling operation and the number of horizontal lines in the display, n; is given by

$$n = xmax - xmin + 1$$

where xmax and xmin are the maximum and minimum excursions, in terms of the resolution of the plotter, of the X coordinate of the display. Note that the number of lines in the m^{th} strip may vary from 1 to Δx.

The raster plot is produced a strip at a time. Any dot generating algorithm [6,7] may be used at this stage to translate the portions of the vectors that lie within the strip under consideration into a series of dots. The strip is then sent to the plotting device; optionally, or in addition, it may be stored on an auxiliary device for future use.

Production of a raster file strip proceeds by processing from the start of the vector file and translating vectors until a vector which starts after the current strip is found. Because the vector file is sorted in the descending values of the starting X coordinates, this signifies the completion of the generation of the entire strip. If any vector before this does not cross the boundary of the strip, then it is removed from the vector file. Generation of the raster file strips continues in this fashion until the vector file becomes empty at which point the complete display will have been created. If a vector originates in one strip and crosses the boundary into the next strip, it is retained in the vector file. When the next strip is processed, the point of



Note: The number of lines in the m^{th} strip may vary from 1 to Δx

Fig. 4: Generation of a plot as strips

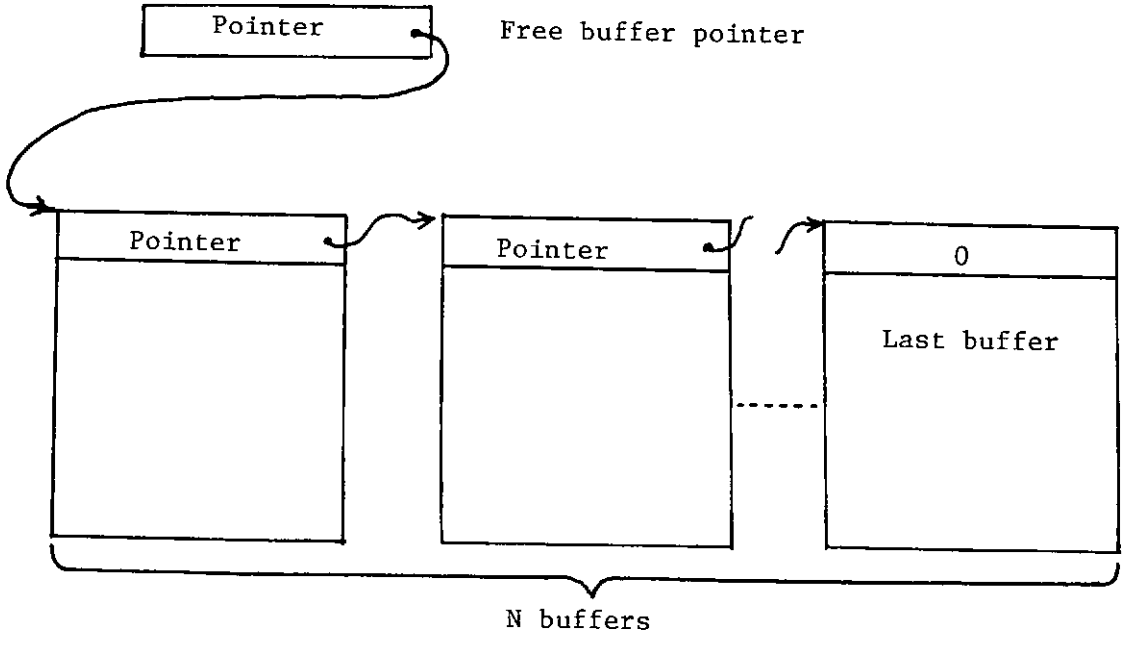


Fig. 5: Free buffer chain

intersection between the starting line of the strip and this vector is calculated and the generation proceeds as described above. The extra step of calculating the point of intersection is necessitated by the fact that the vector file contains integer numbers and the dot generating algorithm used [6] requires the fractional information for initial rounding purposes to obtain the smooth appearance. Alternately, one could retain the required information for continuation of the vector beyond the strip [5]; however, this implies an increase in the amount of auxiliary storage for the vector file. It should be noted here that relatively little time is spent in initialization compared to the dot generating mechanism.

4.0 IMPLEMENTATION:

The hardware, pertinent to the subject matter of this paper, at the authors' installation consists of a Gould 5100 electrostatic plotter connected to an IBM System 370 model 158 via an IBM 1827 data control unit.

The routines to drive the electrostatic plotter are written in the IBM OS/VS assembler language [8] making liberal use of conditional and symbolic assembly, e.g., a single parameter changes the logic for vector generation from bit addressing (suitable for an electrostatic plotter) to byte addressing (suitable for a line printer).

The logic details and the data structures used are described in this section.

4.1 Phase I:

The plotting package [1,2] calls the device driver (routine Pxxxx) each

time a movement of the plotting pencil is involved. This call is of the form:

CALL Pxxxx (IC,IX,IY)

where, IC indicates whether the plotting pencil is to be raised or lowered before and/or after its motion and IX,IY are the X-Y coordinates of the point to which the plotting pencil is to be moved. IX and IY are specified in terms of the resolution of the plotter (which is one hundredths of an inch for the Gould 5100 plotter).

The main aim of this phase is to create the vector file. In its simplest form, this is achieved by collecting as many of the vectors as possible in a single buffer in main memory and, when the buffer is filled, writing its contents to an auxiliary storage device (referred to as a "disk" hereafter); the buffer can then be reused.

When two or more buffers are specified through the job control specification statements, a somewhat more elaborate scheme is used in order to overlap the processing and the input/output to the "disk". The principle elements of this scheme are:

(a) free buffer chain

(b) active buffer control chain

(c) vector file.

The "free buffer chain" (see Fig. 5) is a forward linked list whose head is kept in a "free buffer pointer" word. At least one free buffer is maintained on this chain all the time.

The buffers are removed from the "free buffer chain" as required and used as active buffers to store the vectors as they are generated. When filled,

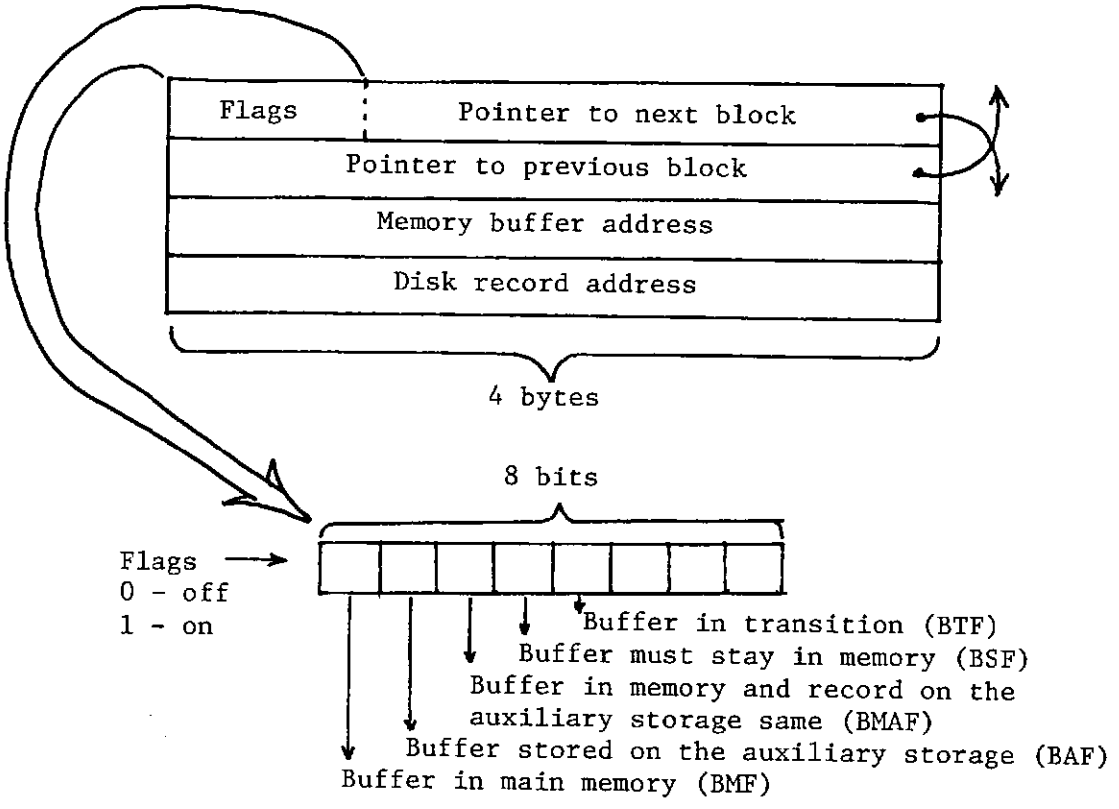


Fig. 6: Format of a "control block"

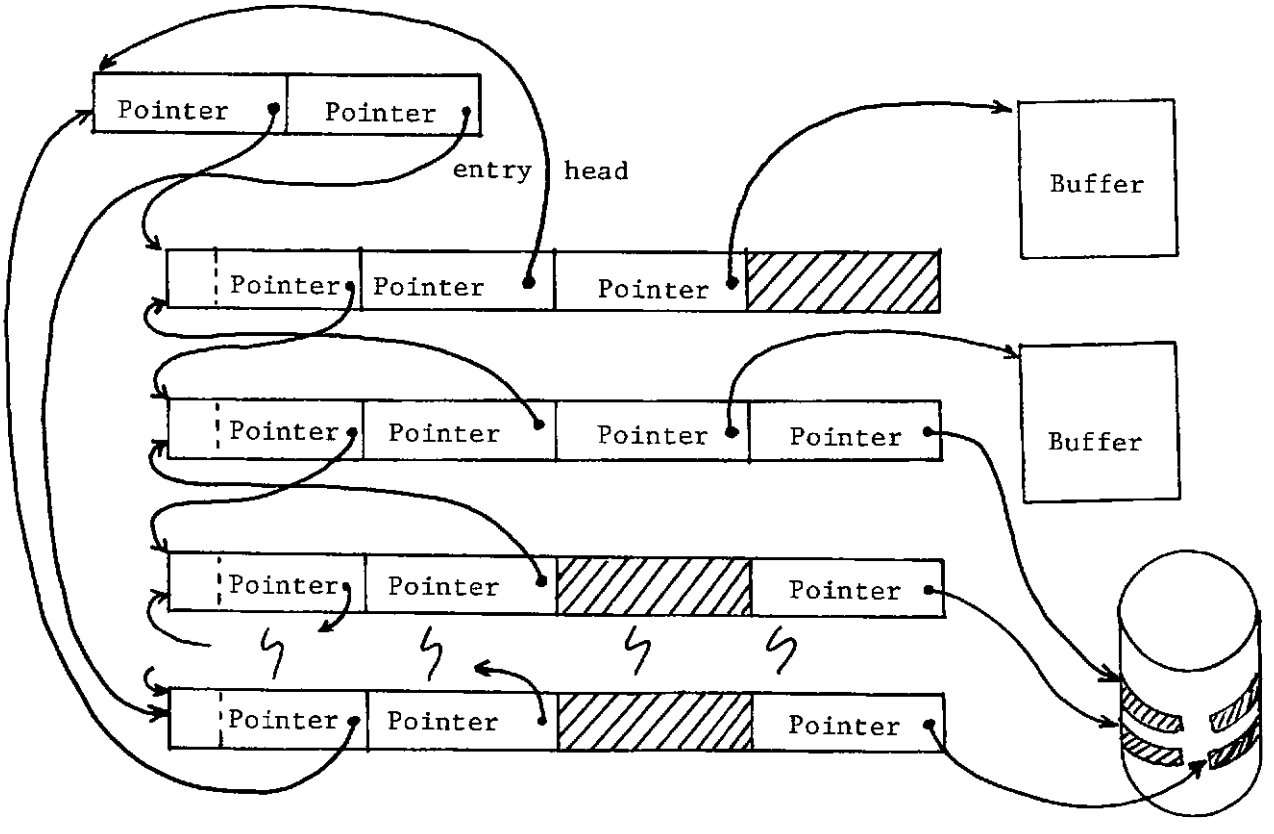
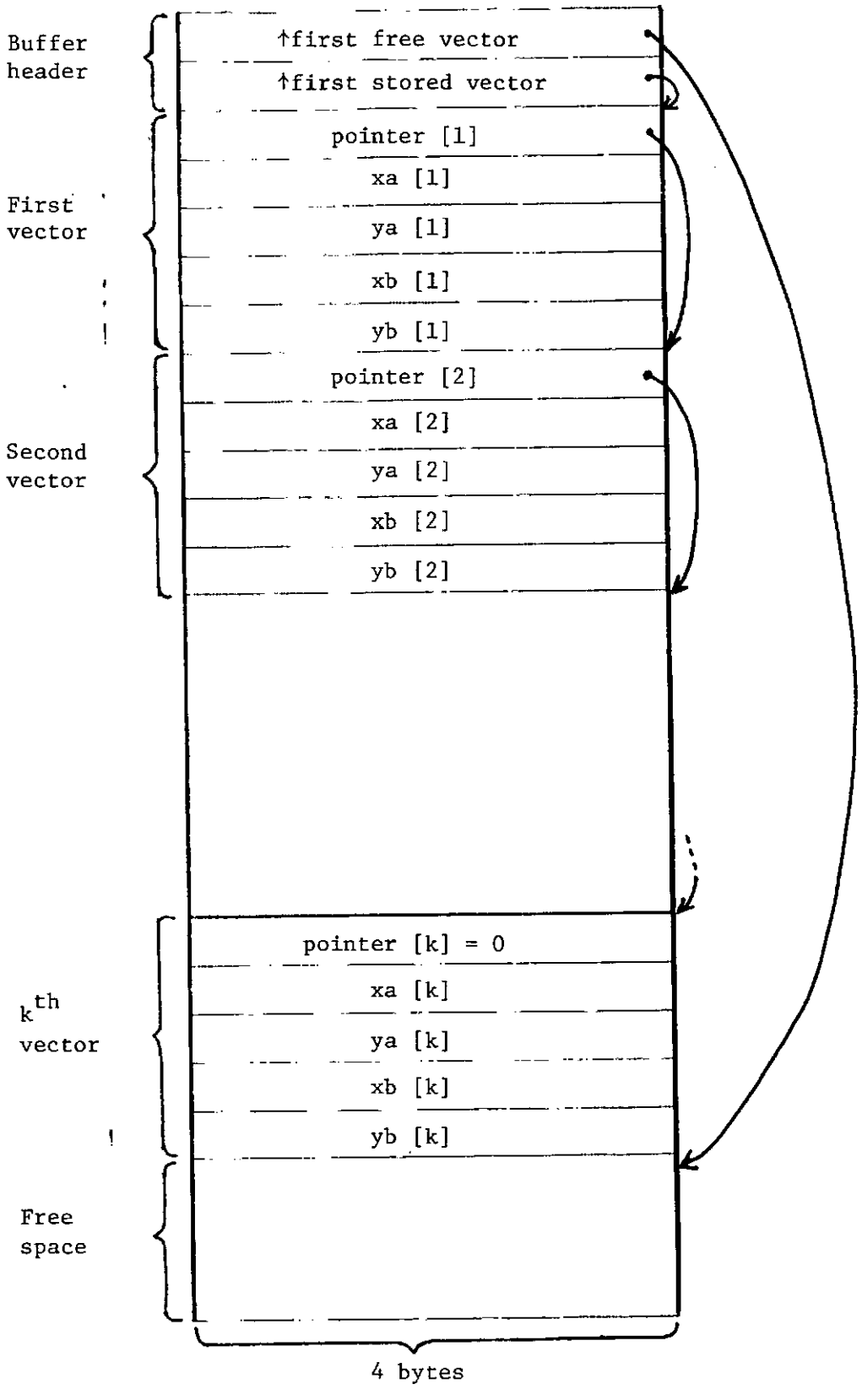


Fig. 7: Active buffer control chain



Note:

1. The number of vectors, v, that could be accommodated in a buffer is given by:
$$v = \lfloor \frac{B-8}{20}$$
where, B is the number of bytes in a buffer and $\lfloor$ denotes the floor operation.

2. $xa[i] \geq xb[i]$

Fig. 8: Structure of an active buffer

these buffers are emptied onto the vector file and they once again join the "free buffer chain". The management of these active buffers and the records created with them on the "disk" is achieved by means of the "active buffer control chain". Each entry, called the "control block", in this chain contains the information about the status of the buffers and the records and their addresses in main memory and/or on "disk". The format of a "control block" is given in Fig. 6. The "active buffer control chain" is pictorially depicted in Fig. 7, from which it can be seen that this chain is a doubly linked circular list with the "entry head" kept separate in a double word. This "entry head" contains the addresses of the first and last block on the chain.

The structure of the active buffer, when it is partially filled, is shown in Fig. 8. Once again one can see that the buffer management is achieved through a linked list.

The third element, namely the vector file, is in a way optional. For relatively small plots, the entire vector file may be maintained within the buffers in main memory, thus not requiring any input/out to the "disk". It is possible to specify a large number of buffers to achieve this.

The various aspects involved in the first phase are summarized in the algorithm given in Fig. 9. This algorithm calls the procedure "ACQBUF" (see Fig. 10) which acquires a buffer from the "free buffer chain".

The following quantities are used in "ACQBUF":

```
procedure  Pxxxx;
   if  first time  then begin
      allocate buffers;
      construct "free buffer chain";
      count ← 0;
      end;
   if  a vector is to be produced  then begin
      get end points (xa,ya),(xb,yb);
      arrange end points such that xa ≥ xb;
      if  count = 0  then call  ACQBUF;
      store vector in buffer;
      count ← count-1;
      end;
   return;
end;
```

Fig. 9: Algorithm "Pxxxx" for phase I

```
procedure  ACQBUF;
   if  N=1  then  initiate buffer write request;
   if  buffer write request outstanding  then begin
      wait for completion of write;
      turn on BAF in "control block";
      turn off BTF in "control block";
      store disk record address in "control block";
      transfer buffer from "control block" to "free buffer chain"
      end;
   allocate a new "control block";
   transfer the first buffer from "free buffer chain" into
      "active buffer control chain";
   turn on BMF in "control block";
   count ← v;
   if  "free buffer pointer" = 0  and  N > 1  then begin
      turn on BTF in "control block" for the "oldest"
         buffer in main memory;
      initiate a write for this buffer;
      end;
   return;
end;
```

Fig. 10: Procedure "ACQBUF" to acquire a buffer from the "free buffer chain"

N - number of buffers in main memory

v - maximum number of vectors that could be stored in a buffer

BAF - Buffer stored in Auxiliary storage Flag

BTF - Buffer in Transition Flag

BMF - Buffer in main Memory Flag.

4.2 Phase II:

This phase is initiated when the end of plot is indicated by invoking the routine Exxxx. The major aspects of this routine are:

(a) completion of vector file

(b) sorting

(c) strip production

(d) vector buffer management.

The logic of the routine, in general form, is given in Fig. 11.

When the end of plot routine receives control, it determines if a request to write a buffer is outstanding. Absence of such a request may imply that the vector file can be maintained in main memory which in turn would result in no input/output activity to the "disk".

If a buffer write request is found to be outstanding, it is completed. All the active buffers in memory are then written onto the "disk" and the "buffer in memory and record on the auxiliary storage same" flag (BMAF) in the "control blocks" owning these buffers are turned on. This completes the creation of the vector file.

The host operating system SORT program [9] is then invoked to sort all the

```
procedure  Exxxx;
   if  buffer write outstanding  then begin
         wait for write to complete;
         for  all active buffers in memory
              write buffer to "disk"
              turn on BMAF in "control block";
              end;
         end;
   call  SORT (vectors);
   allocate strip buffers in main memory;
   k ← number of "control blocks" -N
   do while  sufficient memory  or  k > 0
      allocate a vector buffer from main memory;
      k ← k-1;
      end;
   n ← xa[1]-(minimum xb)+1
   for  i=n  step  -Δx
      do until  j="end of list"  or  xa[j] ≤ i-Δx
         convert portion of vector within strip to dot form;
         if  xb[j] > i-Δx  then begin
              remove vector [j] from buffer;
              if  buffer is empty  then begin
                  remove associated "control block";
                  if  "active buffer control chain" is empty  then begin
                          write last strip to raster file;
                          return;
                          end;
                  end;
              end;
         j ← pointer [j];
         end;
      write strip to raster file;
      end;
end;
```

Fig. 11: End of plot routine

vectors. (Any SORT program, e.g. [10], may be invoked at this stage). SORT returns the vectors in descending order of xa and these are stored back into the vector buffers which are processed in much the same way as when the vector file is created. When this process completes, the first "control block" owns a buffer with the highest xa values which in turn are arranged within the buffer as a linked list in descending order of xa values, i.e.

$$xa[i] \geq xa[pointer[i]]$$

for all values of i.

The algorithm then advances to the strip production stage. First, a sufficient number of strip buffers are allocated from main memory. The number of bits, b, required for a strip buffer is given by,

$$b = (w \times r + c) \times \Delta x$$

where, w is the width of paper in inches

r is the plotter resolution

c is the length of plotter control information

Δx is the number of lines in a strip

Additional vector buffers are created until there is no more main memory available or there is a buffer for each "control block" in the "active buffer control chain"; the latter is the ideal case.

To produce a strip from x to $x-\Delta x$, the vector buffer file is scanned from the beginning. The portions of vectors that lie within the strip boundary are converted to dot forms until a value $xa \leq x-\Delta x$ is encountered. At this point, the strip buffer is written to the plotter or a raster file. If a value $xb > x-\Delta x$ is encountered, then the corresponding vector is removed from the buffer.

When all the vectors have been removed from a vector buffer, the associated "control block" is removed from the chain and when the "active buffer control chain" becomes empty, the plot is finished.

Finally, the raster file, if it has been created, is copied to the plotter. This step may be required to avoid lengthy delays between strip productions. These delays cause toner stains on electrostatic plotters. In addition, as a by product of this step, multiple copies of a plot can be produced; provisions to achieve this are made in the implementation.

A significant part of the implementation in Phase II is involved with managing the vector buffers and in trying to overlap processing with input/output to "disk". This management essentially involves "software paging" with provisions for garbage collection for reclaimation of buffers when input/output activity to the disk becomes too high. The detailed description of this process is beyond the length of this paper.

5.0 CONCLUDING REMARKS:

An algorithm suitable for producing plots on electrostatic plotter type devices has been designed and implemented. It has worked very satisfactorily serving a number of users. There is no restriction on the size of the plot that can be generated and, in fact, no a-priori knowledge of the size is required. Complete passes of the vector file are not required during production of each strip and only the points that lie within the strip are calculated.

Typically, a vector file (even for a large complicated plot) is found to be somewhat more manageable than the corresponding raster file. For close to two years that the algorithm has been in production, we have yet to have a plot

fail because of insufficient auxiliary storage space for the vector file; however, allocation for the raster file had to be increased on several occasions during the same period.

It might be interesting to note that, at the authors' installation where IBM 3330 disk drives are used, a track of disk can hold 651 vectors in the vector file. The production version is setup to handle up to 1,171,800 vectors. As far as the raster file is concerned, a track of an IBM 3330 disk can hold a strip 0.49" wide and about 21" long; the production version is set up to handle up to over 3000" along the X axis.

The program requires less than 12000 bytes of main memory.

Originally, a simple algorithm for sorting was built into the end-of-plot routine rather than invoking the IBM SORT program; however, the performance was found to be intolerable, especially for large displays.

References:

1. Gujar, U.G. Computer Plotting. Computing Centre, University of New Brunswick, Fredericton, N.B., Canada (Oct. 1972, fourth printing Oct. 1976), 212 pages.

2. Gujar, U.G. A device independent computer plotting system. Proceedings of ACM SIGPLAN/SIGGRAPH Symposium on Graphic Languages (April 1976), 85-100.

3. Noll, A.M. Scanned display computer graphics. Comm. ACM 14,3 (Mar. 1971), 143-150.

4. Ophir, D., Rankowitz, S., Shepherd, B.J., and Spinrad, R.J. BRAD: The Brookhaven raster display. Comm. ACM 11,6 (June 1968), 415-416.

5. Jordon, Jr., B.W., and Barrett, R.C. A scan conversion algorithm with reduced storage requirements. Comm. ACM 16,11 (Nov. 1973), 676-682.

6. Newman, W.M., and Sproull, R.F. Principles of Interactive Computer Graphics. McGraw-Hill, New York, 1973, 41-46.

7. Bresenham, J.E. Algorithm for computer control of a digital plotter. IBM Systems Journal 4, 1(1965), 25-30.

8. ----. OS/VS and DOS/VS Assembler Language. International Business Machines Corp., Lidingo, Form No. GC33-4010-0, 1972.

9. ----. OS/VS Sort/Merg Programmers Guide. International Business Machines Corp., Lidingo, Form No. SC33-4035-3, Mar. 1976.

10. ----. SyncSort Programmer's Guide. Whitlow Computer Systems, Englewood Cliffs, 1976.