A Soft-edged Character Set and its Derivation

A. J. Wilkes and N. E. Wiseman
University of Cambridge, Computer Laboratory, Corn Exchange Street, Cambridge CB2 3QG, UK

Characters are normally displayed on raster display devices in the form of dot arrays. Humble VDUs often use 6 × 10 arrays spaced on an 8 × 11 grid of positions. Each character is then defined by 60 bits and the entire character set is stored in a read only memory of, say, 96 (characters) × 6 (bits wide) × 10 (bits high). This is 5760 bits all together and accounts for only a tiny proportion of the component cost of the unit electronics. In some of the 'better' terminals more characters may be provided. Some character sets may be held in random access memory and thus be loaded dynamically from a user's definitions, but it remains the case that the character generator logic is relatively small compared with everything else. Indeed, this is increasingly true with more and more upmarket displays—most of the increased function being devoted to picture storage and manipulation. A binary dot array is not in fact the best way to render a letterform on a CRT display. The dottiness produces a pretty crude and unsatisfactory image which is ugly and often hard to read. Raster displays do, it is true, tend always to be seen showing dotty and ugly pictures but diagrams are not intended for 'reading' in quite the same manner as running text. Fortunately, this dottiness is not intrinsic and solutions are known for arbitrary images, including text.[1,2] Quite stunning improvements can be achieved by simple methods which can be cheaply implemented for character displays. This paper discusses the techniques and presents a complete alphabet definition as an example of the method in use.

INTRODUCTION

A picture generated for output on a device of limited resolution ought to be processed to exploit the device characteristics as well as possible so that the distortion and degradation are controlled and minimized. Except for stylized pictures tailored to the display properties, the only freedom we have is in the way the display is controlled to reproduce the given picture. The computation for a raster display is in fact a sampling of the image at a spatial frequency given by the pixel spacing. Consider the edge of some feature in the picture where differently shaded polygons meet (see Fig. 1).

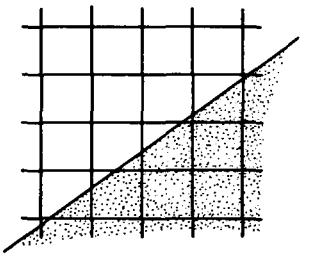


Figure 1. A feature boundary intersecting the pixel grid.

This edge intersects the pixel grid, which is the spatial sampling frequency, and at each sample (pixel) we must compute an appropriate shade. Except when the edge aligns perfectly with the grid, this shade will be neither that of one polygon or the other but an intermediate value fixed by the amount of each which falls within the pixel boundary. Suppose that the edge is black on one side and white on the other, and that 4 bits are available to specify a shade encoded as a binary value between zero and 15. The situation shown above might then give rise to the following pixel values (see Fig. 2).

This use of grey scale to form what is intended as simply a black-white edge works extremely well when the pixel size is reasonably small. The subjective effect is in fact of a hard straight edge rather than of the soft defocussed edge which one might expect. The reason for this has to do with the way in which the brain appears to seek and accentuate line features using global clues as well as local detail. If no grey scale is available the actual pixel values will of course be zero when the wanted value is below eight, and one otherwise which produces the jagged edge which is the familiar and objectionable effect we are trying to rid ourselves of here (see Fig. 3).

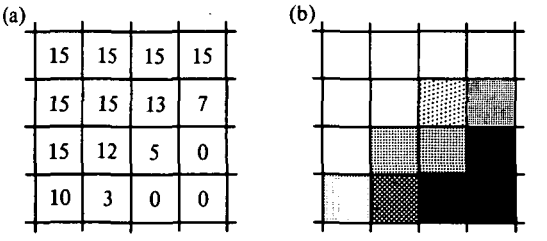


Figure 2. (a) Grey values in vicinity of boundary. (b) Resulting image.

The jaggedness obviously will vary in detail as the edge moves over the pixel grid. A given edge may in fact show itself in many different forms, called aliases, and a slowly moving image produces edges which shimmer as a result of the succession of different aliases which the

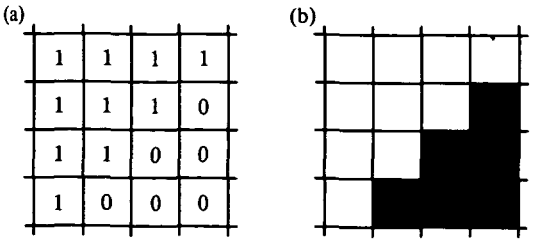


Figure 3. (a) Binary values in vicinity of boundary. (b) Resulting image.

CCC-0010-4620/82/0025-0140$04.00

movement induces. The use of grey scale in the manner suggested above is called anti-aliasing; the grey values may be computed in several different ways. The basic idea comes from sampling theory, which states that with a sampling frequency of f, accurate reconstruction may be achieved for images containing no components above $f/2$. Put simply, this means that if an image is filtered so that such high frequencies are removed, then the jaggies will go away. All methods for anti-aliasing amount to filtering the image in this way, but the methods differ in their accuracy and the amount of computing required. Quite simple, quick algorithms are often good enough, although rendering movement satisfactorily is difficult. For the present purpose a simple method will be used, based on pixel averaging. The letterforms will be specified to a high resolution so that they can be mapped onto the pixel grid by amalgamating squares. The total of black squares per pixel is then counted and used to emit a corresponding grey value for each pixel.

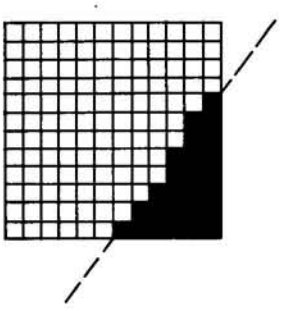


Figure 4. Super-sampling a pixel to determine its grey value.

Shown in Fig. 4 is a single pixel divided into a binary array of dots, presumed fine enough to allow a count of dots to be a sufficiently accurate indicator of the grey value needed. Letterforms when designed will usually be to an extremely high resolution so this condition is easily met.

In summary, the derivation of an alphabet to a low spatial resolution from binary high resolution masters can use grey values to restore some of the lost detail and fool the eye into hardening and smoothing the image contours. Shown in Figs 5–7 are some examples of the sort of results we have obtained starting with a particular alphabet and deriving others from it.

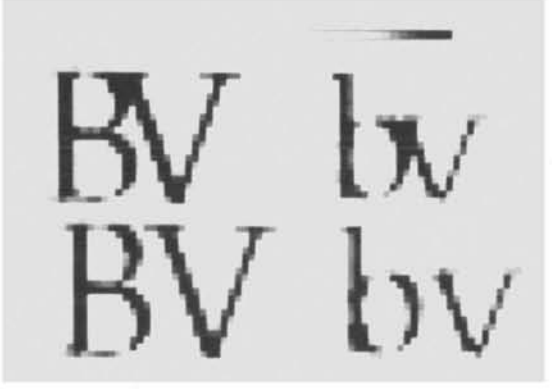


Figure 5. Enlarged letters, 20 pixels high, 2 bits per pixel.

Figure 6. Enlarged letters, 56 pixels high, 4 bits per pixel.

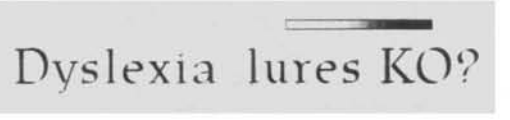


Figure 7. Normal sized letters, 56 pixels high, 4 bits per pixel.

IMPLEMENTATION

Experiments in letter design and text processing for print quality applications have been going on in this department for some years.[3–5] Alphabet definitions associated with this work were available to us in the form of high resolution polygonal outlines. Each letter is formed from the superimposition of a few such polygons, comprising up to a hundred edges or so in a coordinate space delimited by 16 bit integers. A program to scale and scan convert these letterforms into a run-encoded form for writing text to precision raster plotters and film setters was used to preprocess the letters for the use described here. Thus our input was a run-encoded binary array for each letter on a grid roughly 700 × 700. The pixel averaging algorithm we used is shown below.

```
$( LET pixels = VEC width-1
                          // Accumulated values,
                          // which start off at zero
   FOR i = 0 TO width - 1 DO pixels!i := 0

   FOR line = 1 TO N      // Read in N lines of pixel
   DO $(                  // runs and accumulate them
        UNTIL end_of_line
        DO $( LET first = READN ()
                          // Read in a span of blacks
              LET last = READN ()
              FOR x = first TO last
              DO pixels! (x/N) + := 1
           $)
      $)
   // Write out a line
   FOR x = 0 TO width - 1 DO write( pixels!x   )
   newline ()

$) REPEATUNTIL end_of_data
```

The output consists of rows of *width* pixel values, the whole array being terminated by a value of −1. In practice; some compression is used: leading and trailing zeroes are suppressed and each row is preceded by an offset and the number of non zero data values on it.

The pixel values will range between 0 and N^2 and will need to be mapped to the actual set of grey values which the output device can handle. The choice of mapping is up to us. We can make use of the opportunity to tune up the performance of the overall method to compensate for brightness non-linearities in the display, filtering faults and the human vision process. If the output device has more grey values than the filter emits, then the mapping can be done by a table of outvalues indexed by the invalues. If, on the other hand, the output device has fewer grey values than the filter emits—as was the case for our experiments—the mapping can be done by a binary search in a table of invalues. The search code is:

```
outvalue := 0
increment := N²/2
UNTIL (increment = 0)
DO  $(
        IF (invalue > maptable!(outvalue +
          increment))
        THEN  outvalue + := increment
        increment := increment/2
    $)
```

The vector *maptable* records the *invalues* at breakpoints in the mapping. The *outvalue* deduced by the algorithm is the index to the next lower *invalue* to the one given. It evidently requires that the mapping is monotonic increasing. The filtered and remapped data for each letterform can now be stored in a pixel array, say *width* pixels wide, *height* pixels high and *depth* bits deep. An alphabet in this form for *width* = 19, *height* = 21 and *depth* = 4 is reproduced in the Appendix. The mapping chosen for this example is linear—that is the *invalues* were arranged in 16 equal intervals, each corresponding with a particular *outvalue*.

REMAINDERING

In the latter case, where there are more *invalues* than *outvalues*, a cumulative quantization error is introduced, causing the integrated grey level of the resulting picture to be too low by an amount determined by the image

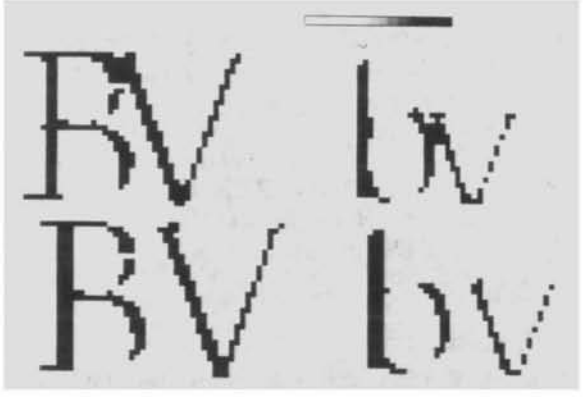


Figure 8. Enlarged letters, 30 pixels high, 1 bit per pixel, no remaindering.

details. Each consultation of *maptable* gives an error (or remainder) of

invalue − maptable!outvalue

which is positive or zero. The cumulative effects can be

Figure 9. Enlarged letters, 30 pixels high, 1 bit per pixel, remaindered.

minimized by distributing the error over neighbouring pixels, rather than simply throwing it away. The simple technique of adding it to the pixel value to its right works quite well, as Figs 8 and 9 show.

Remaindering in both coordinates is theoretically better but probably unnecessary when several levels of grey are available for the output. As might be expected, the effect is most noticeable when the size of the error terms is large: that is, there are few grey levels available in the output. With 16 levels, we could not easily distinguish the effects of remaindering for the alphabets we wished to display on a Sigma 5684 terminal.

USING THE DATA

The use of anti-aliased characters on displays which have a grey scale capability is relatively straightforward. A line of text is scanned *height* times, constructing and emitting on each scan the corresponding pixels for each letter required in turn. The basis of the method is shown below. A vector *txt* holds the line of text being processed, with its first word indicating the number of characters. Each character is represented by the address where its pixel array starts; a pixel array is a vector of *height* rows each held as a vector of *width* pixels.

```
FOR i = 1   TO height
$( scanptr := 1
   FOR j = 1   TO txt!0
   $( letter := text ! j
      FOR k = 1   TO width
      DO   $( scanline!scanptr := (letter!i) ! k
              scanptr + := 1
           $)
   $)
   scanline!0 := scanptr
   emit( scanline )
$)
```

A piece of logic to do the same thing is simple and would be a quite insignificant addition to the hardware of a character based display. The pixel arrays are held in ROM, *depth* bits deep instead of one, and the *depth* bits shifted to the video generator pass through a Digital to Analogue Converter (DAC) en route.

The row counter (*height* bits) and character code address the ROM in the usual way causing the emission of *width* × *depth* bits (instead of *width*). These pass into *depth* shift registers whose outputs are connected to the

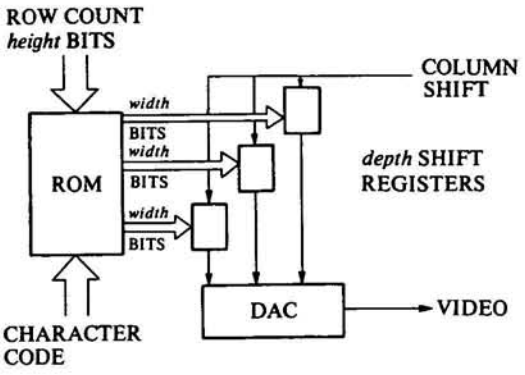


Figure 10. Basic logic for a soft-edged character generator.

DAC which emits video. No timings are altered and the circuits are not expensive.

INTERLACING

With traditional simple VDUs, the use of interlacing does not greatly increase the overall legibility of the character set, which will usually have been designed on a simple dot matrix pattern. Doubling the resolution allows a few refinements, but no major improvements are possible: the image is still locked to the pixel grid. When using a grey level display, however, advantage can be taken of any additional spatial resolution: the increased number of pixels can be immediately taken advantage of (no laborious tweaking by hand is necessary), and will be reflected directly in the quality of the final image. The example character set in the Appendix was sized to allow display of 24 lines of good quality text on a 512 line interlaced screen.

FURTHER USES

This exchange of grey values for spatial resolution in images as perceived is a reversible effect—that is, we can also use binary pictures to create a grey scale effect by losing spatial resolution (this is the means by which continuous tone images are turned into halftone images for printing by ink transfer). It seems very likely that a digital film setter such as is used to make page images for printing could exploit soft edging in the way described in this paper to remove the bumpiness in letters of which the industry constantly complains. The additional apparent spatial resolution which soft edging produces should still be there after the image is thresholded by the film emulsion, provided that image integration prior to exposure is well chosen. We hope to be able to verify this soon.

CONCLUSIONS

We have presented a simple technique for anti-aliasing characters and demonstrated the benefits in picture quality which can be easily achieved. As better displays become available, we think some such technique should be implemented in hardware. Suitable enhancements to handle variable widths and heights are not difficult to devise and, indeed, the example character set given below is intended for use with variable pitch.

By fixing the size in relation to the display resolution and accepting less freedom in the letterforms a soft edged character set could be designed which would reproduce even better than the examples shown here. Graphics arts designers might not find the restrictions of form fully acceptable, but we believe that the raster graphics medium deserves their attention.

REFERENCES

1. F. C. Crow, The aliasing problem in computer generated shaded images. *Communications of the ACM* **20**, (1977), pp. 799–805.
2. J. E. Warnock, The display of characters using grey level sample arrays, *Xerox Palo Alto Research Center Research Note CSL-80-6*, (1980).
3. A. M. Pringle, P. Robinson and N. E. Wiseman, Aspects of quality in the design and production of text. *ACM SIGGRAPH* **13**, (1979), pp. 63–70.
4. N. E. Wiseman, C. I. C. Campbell and J. Harradine, On making graphic arts quality output by computer. *Computer Journal* **21**, (1978), pp. 2–6.
5. N. E. Wiseman, Computer designed letters. *Information Design Journal* **1**, (1980), pp. 218–222.

Received May 1981

APPENDIX

The example is based on an alphabet designed by David Kindersley in digital form using the ELF design system. The data is presented as rows of hexadecimal numbers, each one describing one line of a character. The data for each character is prefixed by $⟨character⟩ ⟨ASCII code in octal⟩. The sizes have been normalized to a total character height of 21 units, and a nominal width of 19. Figure 11 shows a screen reproduction of the alphabet drawn at actual size.

Figure 11. A screen reproduction of the alphabet drawn at actual size.

ALPHABET LISTING IN HEXADECIMAL

$! #41
```
....................
....................
....................
....................
.....1..............
..59F3..............
..FFF2..............
..EFE...............
..CFC...............
..AFB...............
..8F9...............
..6F7...............
..4F5...............
..2F4...............
...3................
....................
...D3...............
....................
....................
....................
```

$" #42
```
....................
....42.52...........
...1F14E............
...77.A5............
..16.26.............
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
```

$# #43
```
....................
....................
....................
....................
....................
....................
....17157...........
....8C1D7...........
....C83F4...........
..13F58F211.........
157CF8FF661.........
...9B1D8............
..1E93F511..........
16AFCDF8861.........
..5F19C.............
..9C.C9.............
..941A3.............
....................
....................
....................
```

$$ #44
```
....................
....................
....................
....................
....................
....15.6............
....27.9............
...27EAF93..........
..9827.BAA1.........
.3F337.9............
.3F937.9............
..7EEE4A............
....5DAFB61.........
....27.A6FB.........
....27.9.AF.........
.2A628.9.A9.........
..7EFE5C35..........
....4A3A............
....25.7............
....................
....................
```

$% #45
```
....................
....................
....................
....................
....................
..........87........
...253....7B........
..61.83..3D1........
..A..44.1C5.........
..6714..99..........
...1...5C1..........
.......1D3..........
......B6............
.....7A.............
....3D1.3358........
....D4..8..83.......
...98...A..72.......
..5B1...4733........
.1E2................
.31.................
....................
....................
```

$+ #53
```
....................
....................
....................
....................
....................
.....33.............
.....D7.............
.....D7.............
.....D7.............
.....D7.............
..4777FE7773........
.....D7.............
.....D7.............
.....D7.............
.....D7.............
.....C5.............
....................
....................
....................
....................
```

$, #54
```
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
..33................
..E5................
.1E1................
.45.................
....................
....................
....................
```

$- #55
```
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
...3566551..........
...32322............
....................
....................
....................
....................
....................
....................
```

$. #56
```
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
....................
..33................
..F3................
..1.................
....................
....................
....................
....................
....................
```

$/ #57
```
....................
....................
....................
....................
....................
....................
....91..............
....79..............
....E2..............
....79..............
....1D2.............
....88..............
....1E1.............
....97..............
....2D1.............
....97..............
....2D1.............
....A5..............
....................
....................
....................
....................
....................
```

$5 #65
```
....................
....................
....................
....................
....................
....................
..1BB8951...........
..4F2...............
..5F1...............
..7D................
..8D4971............
..21.2CD2...........
.....1FA............
.....BE.............
.....9E.............
.....AB.............
...3...E5...........
..CE6366............
...453..............
....................
....................
....................
....................
```

$6 #66
```
....................
....................
....................
....................
....................
....................
.....1353...........
....85..............
....99..............
..2F2...............
..7D25A93...........
..BF5.2CE4..........
..FF...1FB..........
..FF....BE..........
..FC....9E..........
..CD....AB..........
..8F4..1E4..........
..1CE7365...........
....342.............
....................
....................
....................
....................
```

$7 #67
```
....................
....................
....................
....................
....................
....................
.379898BB3..........
.....1EB............
.....8F3............
.....2EA............
.....9F2............
....2F9.............
....AF1.............
....4F7.............
...CD1..............
..5F6...............
..DD................
.6D5................
.2..................
....................
....................
....................
....................
```

$8 #70
```
....................
....................
....................
....................
....................
.....1..............
...288EB1...........
..1F3.6F6...........
..6F1..F9...........
..7F2..E8...........
..4F9.2E2...........
...8DDB3............
...774DE3...........
..6D..2EB...........
..BB...AF...........
..CB...AD...........
..AF1..C8...........
..3FB35A............
...2651.............
....................
....................
....................
....................
```

$9 #71
```
....................
....................
....................
....................
....................
....................
...448EB2...........
..98..3EC...........
.3F2...9F2..........
.7F....4F6..........
.8F1...6F9..........
.6F5...8F6..........
.1ED1..FF3..........
..3CE759F...........
.......AB...........
.......E4...........
......96............
...2343.............
..12................
....................
....................
....................
....................
```

$? #77
```
....................
....................
....................
....................
....................
....................
....36B81...........
..1A1.2DC...........
..99...4F3..........
..C9...3F1..........
..8E1..79...........
...51.39............
.....57.............
....4B..............
....B6..............
....B2..............
.....1..............
...1E3..............
....1...............
....................
....................
....................
....................
```

$@ #100
```
....................
....................
....................
....................
....................
.....4247871........
....34.....3C6.....
...45.......1C5.....
..28.........4D.....
..92..254B7...C4....
..C...B...7...96....
.28..48...74..68....
.3A..77...B1..76....
.4B..5B...D...83....
.1E...C616E4.27.....
..D2..241.274.......
..6B................
...A9...............
....6B6....11.......
.....47542..........
....................
....................
....................
....................
```

$A #101
```
....................
....................
....................
....................
....................
....................
.....11.............
.....FF2............
.....5FF7...........
.....AACC...........
.....F56F3..........
....5E11F8..........
....AA..BD..........
....F5..5F4.........
...5F8788F9.........
...AA....AD.........
...F4....5F4........
..5E1.....E9........
13ED3....3DF4.......
14633....45441......
....................
....................
....................
....................
```

$B #102
```
....................
....................
....................
....................
....................
..112111............
..7DF78ED2..........
...9E..2FB..........
...9E...BD..........
...9E...9D..........
...9E...B9..........
...9F44781..........
...CFCBBF81.........
...9E...7F7.........
...9E....EC.........
...9E....BC.........
...9E....D7.........
..3EF321691.........
..4655752...........
....................
....................
....................
....................
```

$C #103
```
....................
....................
....................
....................
....................
.....21.1...........
.....4369EBC........
...3A1...1BF........
..1E3.....F.........
..7C.....A..........
..C9................
..F7................
.1F7................
..F9................
..CD................
..7F5...............
..1DE2....34........
...2CE7212A7........
.....37643..........
....................
....................
....................
....................
```

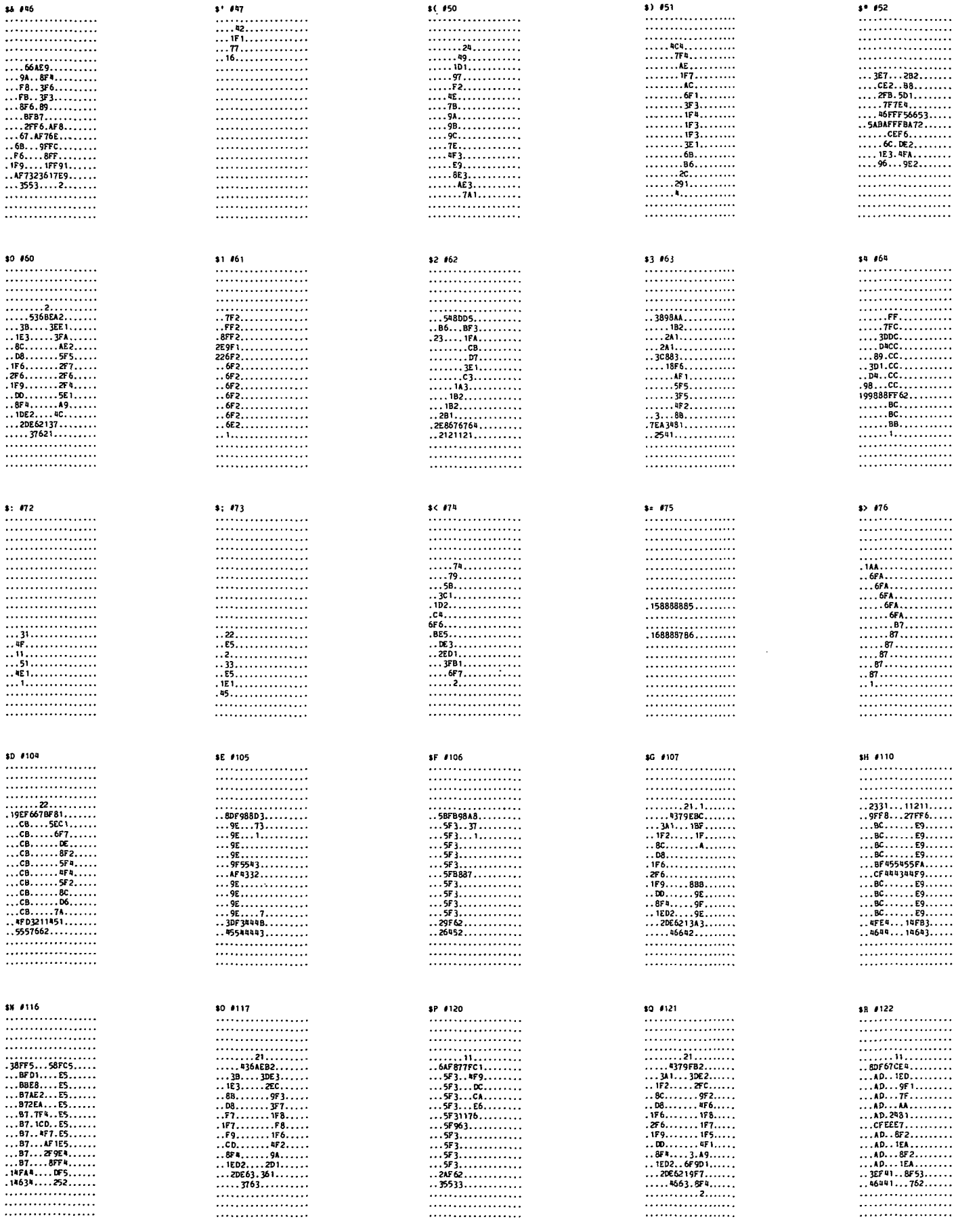
$I #111
```
....................
....................
....................
....................
....................
...2121.............
..49FB5.............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
...3F5..............
..27F73.............
..25543.............
....................
....................
....................
```

$J #112
```
....................
....................
....................
....................
....................
....4AFD6...........
.....2F6............
.....2F6............
.....2F6............
.....2F6............
.....2F6............
.....2F6............
.....2F6............
.....2F6............
.....2F6............
.....2F5............
.7C2.4F2............
.2FC4A5.............
..143...............
....................
....................
....................
....................
```

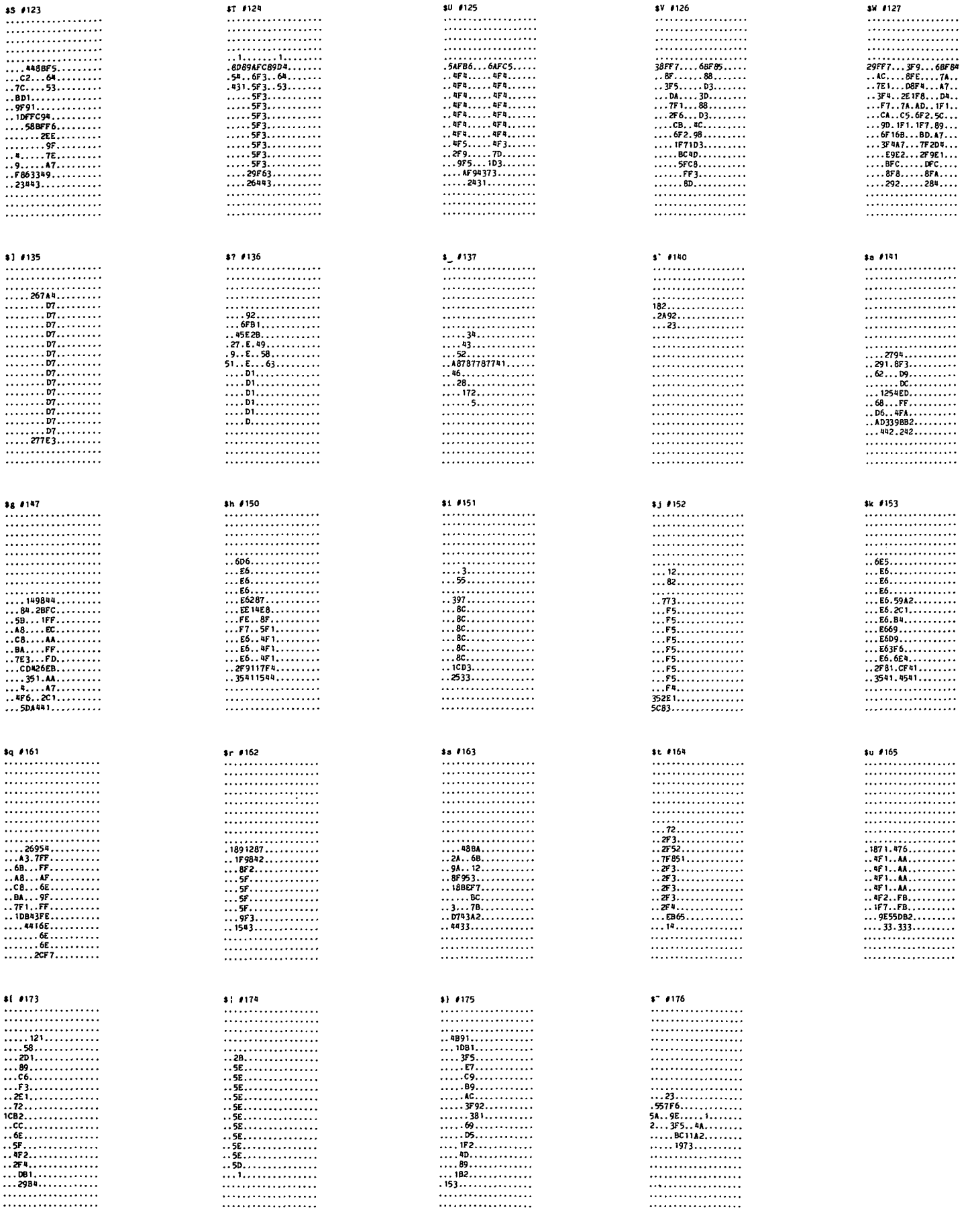
$K #113
```
....................
....................
....................
....................
....................
....................
.18FF81.6CF83.......
...CB...2D2.........
...CB..1C4..........
...CB..A7...........
...CB.7A............
...CB4D1............
...CB6F6............
...CB.8F4...........
...CB..8E2..........
...CB..1CD1.........
...CB...2DC.........
..4FD41..7FC41......
..46441..46441......
....................
....................
....................
....................
```

$L #114
```
....................
....................
....................
....................
....................
....................
..5BFB4.............
...5F3..............
...5F3..............
...5F3..............
...5F3..............
...5F3..............
...5F3..............
...5F3..............
...5F3..............
...5F3..............
...5F3...7..........
..2AF5444C..........
..3545444...........
....................
....................
....................
....................
```

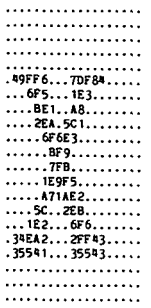
$M #115
```
....................
....................
....................
....................
....................
....................
.19CFC......3FF96...
....8FF4.....7DF3...
....99F9.....C8F3...
...A89E1...3D3F4....
...B74F5...882F5....
...B61DB...D31F6....
...C6.8E2.3D.1F6....
...D4.3F7.97..F7....
...E31.CC.E2..E8....
....F3..6F7C...D9...
....F2..2EF7...D9...
.26F52..BF2.14FD4...
.34542..35...5544...
....................
....................
....................
....................
```



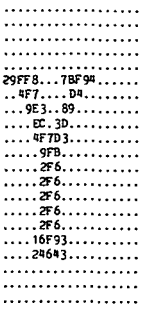
Alphabet Listing in Hexadecimal—*cont.*



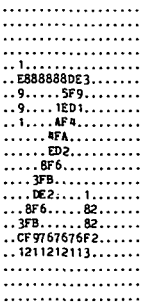
$X #130



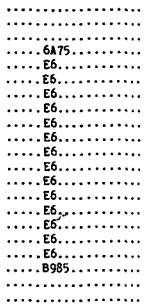
$Y #131



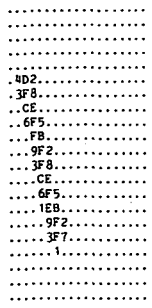
$Z #132



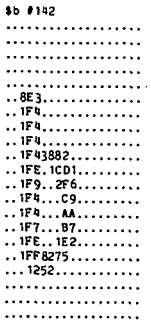
$[#133



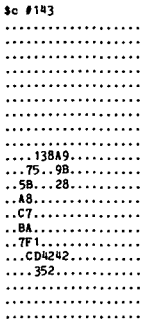
$\ #134



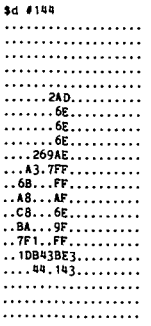
$b #142



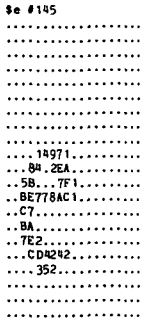
$c #143



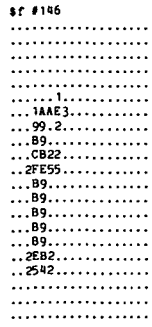
$d #144



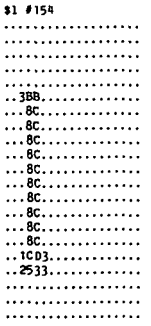
$e #145



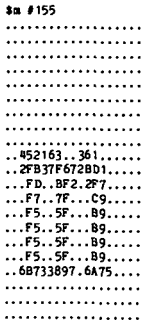
$f #146



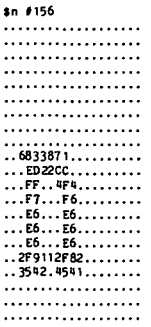
$l #154



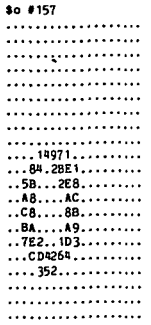
$m #155



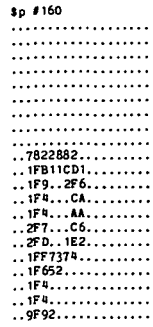
$n #156



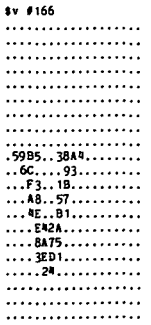
$o #157



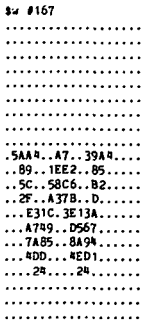
$p #160



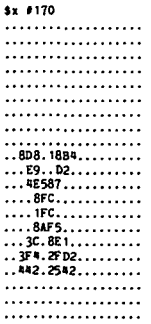
$v #166



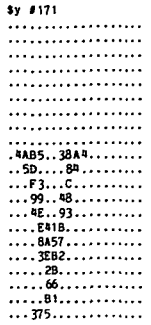
$w #167



$x #170



$y #171



$z #172