May 1983

Report No. STAN-CS-83-966

A Formal Approach to Lettershape Description for Type Design

by

Pijush K. Ghosh and Charles A. Bigclow

Department of Computer Science

Stanford University
Stanford, CA 94305

Computer Science Department
Report No.STAN-CS-83-966

A Formal Approach to Lettershape Description For Type Design

PIJUSH K. GHOSH

CHARLES A. BIGELOW

Research sponsored by

National Science Foundation IST -8201926
United Nations Development Program
Systems Development Foundation

COMPUTER SCIENCE DEPARTMENT
Stanford University

Contents

1 Introduction

To an octopus, a triangle is visually equal to a diamond but not to a square[41]. Very often we fail to distinguish x (times) from x (letter), ∪ (union symbol in set theory) from u (letter), v (logical or symbol for binary operation) from v (letter), or the Greek ϕ (phi) from the slashed zero $\emptyset$ (empty set symbol). There are thousands of such simple geometrical shapes which are visually equivalent and it is hard to discriminate them from each other. We, therefore, use *adjectival descriptors* of shape. For example, we describe u as the 'union symbol' and u as the 'small letter u'. However, this adjectival descriptor method is often misleading, since almost no word has a precise meaning in any language. Consider, for example, the word *tree*. The word *tree* should convey to your mind some sort of woody structure with a trunk, limbs, twigs, leaves etc. Now let us imagine this tree as it undergoes a series of changes: First a young man comes along and cuts its initials in the trunk. Is the object, our "tree", still a tree? Second, a strong wind blows of some of its main branches. Do we still have a "tree"? Finally, the owner starts to remove it by whittling away at the twigs, sawing off the limbs, chopping down the trunk, digging up the stump, and converting the entire original body of material into firewood and trash. Now the question is, at what precise instant did the "tree" cease to be a tree? The point is that if the word *tree* has a precise meaning, then there must be an answer to this last question. One can carry out a similar process by gradually extending the leftmost stem of the letter n and finally generates the letter h in one hundred steps. Again a similar question is, at what precise instant did the letter n become the letter h? Hence the words used to refer to geometrical shapes, pictures or physical objects can not be exact in their meaning.

In this report, we are primarily concerned with lettershapes which are sufficiently regular, deviating only in small ways from truly 2-D geometric shapes, and which have only two grey levels — black and white. The main question is, how can we specify a lettershape. Can a lettershape be specified, or can it be only shown? By "specify" we mean, describe characteristics in such a way that the important pictorial aspects of the original can be recovered in some resonably short amount of time. To put it another way, is it possible to have a notational system for lettershapes, a way to write them symbolically to avoid both ambiguity and obscurity, in the way that we have notations to write mathematical relations, electronic circuits, or music? The problem has become acute now that number-crunching machines have started producing thousands of lettershapes, and seem to give artists an infinite freedom in

lettering design.

The answer is by no means obvious, and a complex hybrid investigation is necessary for this purpose. This should include psychologists concerned with determining how humans extract information from visual stimuli, as well as mathematicians and computer scientists concerned with developing mathematical models and algorithms to supply to the machines similar information for generating visual images.

Most of the time, scientists approach phenomena analytically. A traditional analytical approach would dissect a form and measure its components, on the assumption that synthesizing the measure would reconstitute the form. The Gestalt psychologists, however, resisted this analytical quantification. The Gestaltists argued that the form itself, and not its parts, is the proper unit of observation. The whole form in the mind is little more than a mental compound made up of its constituent elements. But the Gestalt emphasis upon wholeness as figural unity had at least one undesirable consequence: their philosophy tended to promote the mistaken idea that because our perceptual worlds are made up of objects, our perceptual apparatus deals with objects only as whole things. On the contrary, we may now know that one of the major constructive operations our visual apparatus performs is to build up a "picture" of a visual scene out of a series of glances at different parts or aspects of the scene[7]. In the recent past, many perceptual psychologists have turned to analyzing the constituents of form perception, but with a vocabulary that tries to incorporate Gestalt discoveries.

This report is designed to explore some analytic means of specifying lettershapes. Computer representation and analysis of lettershape have made use of two diametrically different approaches, one representing a shape by its boundary, the other by its skeleton or medial axis. Generally speaking, the boundary representation is conceptually simpler to the designer, but the skeletal representation provides more insight into the "piecedness" of the shape. Donald Knuth's METAFONT is one of the sophisticated lettering design system which has basically adopted the *medial* axis approach. Moreover, METAFONT system has introduced the idea of metafont-description of a letter, i.e., to give a rigorous definition of the shape of a letter in such a way that many styles are obtained from a single definition by changing only a few user-defined parameters. That is why we have considered METAFONT system as our starting point and have shown how we can arrive at the definition of a formal language for specifying lettershapes. We have also introduced a simple mathematical model for decomposing a letter into its constituent elements.

This work is a pre-preliminary step to *specify* a letterform. The unkind fact

is that, every shape is *context-sensitive.* If any type design system allows too much freedom, it creates more problems. Each alphabet has characteristic rules that limit the formal possibilities, though not, of course, the creativity of the artist working within the tradition. The creative possibilities of, say, English poetry are on the one hand bounded by the structure of the language, but on the other hand appear infinite for the speakers of the language. Something similar seems to be true for any formal language for type design.

Our hope is that a factor analytic approach may enable us to identify a small set of geometric attributes that predict how a designer will judge, and, by inference, perceive the shapes. These would then be taken to be the "**visual** alphabet" of letterform generation.

2
Albrecht Durer and A Disobedient Student

In his treatise *Underweysung der Messung mit dem Zirckel und Richtscheyt* Albrecht Durer gives instructions for the geometric design of the Roman capital letters[13]. He states each step of the constructions clearly and unambiguously. We would like to present his instruction steps for designing the letter 'A '.

'Draw for each a square of uniform size, in which the letter is to contained./ But when you draw in it the heavier limb of the letter, make this of the width of a tenth part of the square,/ and the lighter a third as wide as the heavier: and follow this rule for all letters of the Alphabet.

First make an A after this fashion : Indicate the angles of the square by the letters a, b, c, d : then divide the square by two lines bisecting one another at right angles- the vertical e,f and the horizontal g,h :/ then, in the lower line, take two points i and k, distant respectively one tenth of the space c,d from the points c and d:/ then, from the point i draw upwords to the top of the square the lighter limb; and thence downwards the heavier limb, so that the outer edge of both may touch, respectively the points i and k:/ then let the triangle be left between the limbs, and the point e be fixed at top in the middle of the letter, and next join both limbs beneath the horizontal line, and let this limb be a third as broad as the heavier limb./

Now let the arc of a circle, applied to the top of the outside edge of the heavier limb, project beyond the square. Then cut off the top of the letter with a serpentine or curving line, so that the concavity decline towards the lighter limb,/ and prolong acutely either limb of the letter at the bottom of either side, so as to meet the angles of the square at c and d: this you shall make with the arc of a circle, whose semi-diameter is one seventh of the side of the square./

..Moreover, this same letter A you may cut off at top with the side of the square, and then produce to a fine point in either direction, as you did the feet below.../

Likewise the same A you may draw in yet another manner- that is, pointed at top./ . . . .

And note likewise that in exactly the same fashion in which this letter is

acutely prolonged at top and bottom, are the other letters to be so prolonged which are drawn with **oblique** lines, as V, X, Y, although a few changes may be necessary.. .

We have here subjoined an engraving of this letter.' **(Fig.1)**

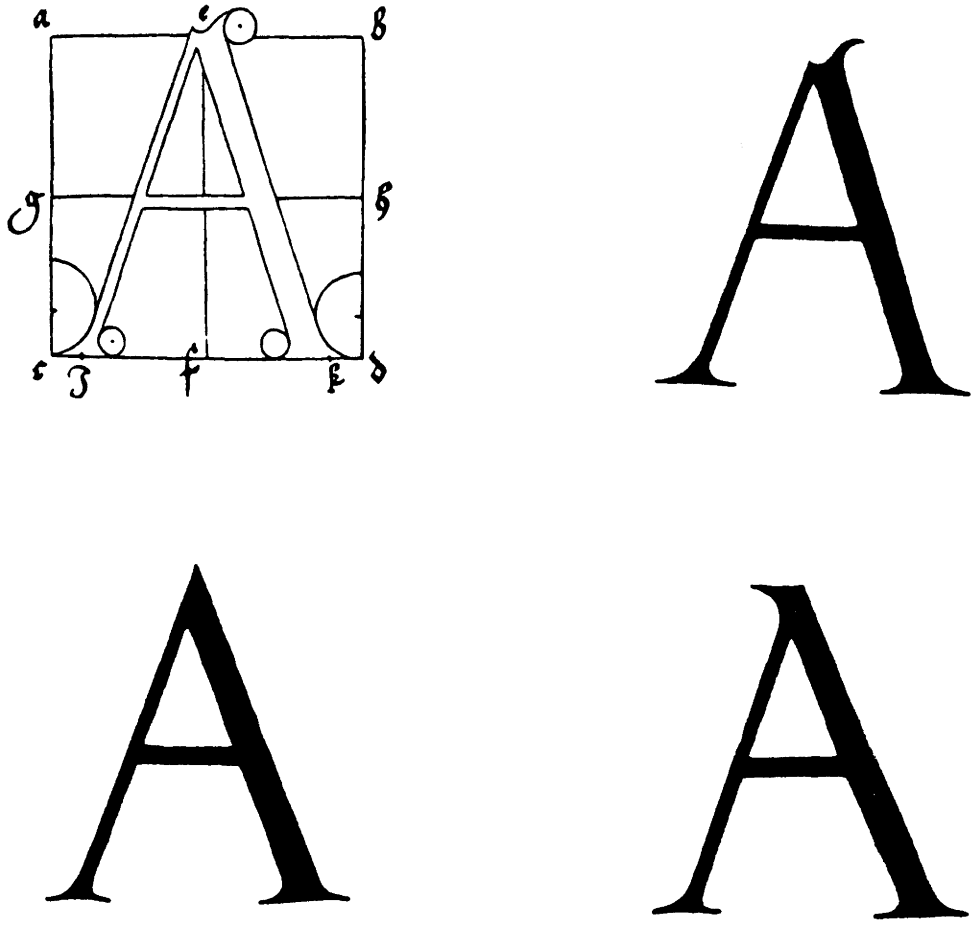


FIGURE **1**

Many lettering artists regard this treatise (as well as other treatises of the same kind) as of little practical value, since the rules of constructions are constrained by artistically worthless and inflexible geometrical calculations. To make the constructions

analytically more flexible and hopefully artistically more useful, we may modify each and every instruction step (separated by /) of Durer and observe the consequences.

The new instruction steps could look like this:

1. The basic letter need not be square; it could be a rectangle, or a rhombus or even a parallelogram. When we design letters with the help of a computer, it is better to *parameterize* the grid, so that by choosing appropriate values of the parameters we are able to produce all kinds of grids, as described above.

2. The width of the heavier stem need not be exactly 1/10th of the side of the square. This relationship may also be a variable parameter.

3. The lighter stem may also be treated as a variable parameter in the same way.

We can proceeed in this fashion, and ultimately find that we have described a letter-shape in terms of a large number of parameters whose values can be varied at will to produce different forms of the letter. This is equivalent to a METAFONT description of a letter.

Donald Knuth in his article *The Concept of Metafont* [28] describes Meta-font as:

"a schematic description of how to draw a family of fonts, not simply the drawings themselves".

He also adds:

"such descriptions give more or less precise rules about how to produce drawings of letters, and the rules will ideally be expressed in terms of variable parameters so that a single description will actually specify many different drawings".

3
Parameterization: A Serious Problem

Parameterization, however, creates a new problem for the designer. As soon as he or she starts work, there is the problem of HOW to paramerize. Basically there are two ways to choose the parameters:

1. Each and every possible parameter is an independant variable, or
2. Some of the parameters are functions of other parameters.

The first method gives rise to too many parameters, which overwhelm the designer with potential choices. This method is potentially too flexible. Arbitrarily choosing the values of the parameters may generate peculiar graphical patterns rather a series of well-formed letters. Thus, the problem is: how to choose the proper values of the parameters.

The second method poses a different questions- how to parameterize, that is, which parameters should be kept as independent variables and which of the rest of the parameters should be related to them, and in what ways. One should not also forget the fact that the range of shapes is likely to be limited if there is a small set of control parameters.

It may be senseless to search for a unique best algorithm for parameter-choice, since there is no evidence for the existence of such an algorithm. The method of choosing parameters mainly depends on what type of variations the artist would like to produce from the basic description of a letterform. Thus, the proper choice of parameters is at least in part subjective.

Yet characterizing the problem of parameterization as a subjective issue does not solve the basic problem. Douglas Hofstadter has used an appropriate term: '*Knobbifying*' the alphabet[23].

The idea is as follows: once the description of a letter is made, it can be viewed as a ***black box*** with a number of parameters as as external knobs. The designer can 'then twiddle the knobs to produce different forms of the same letter.

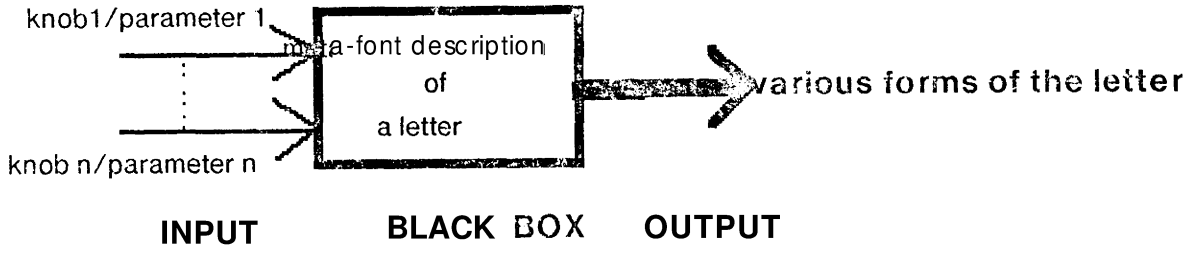
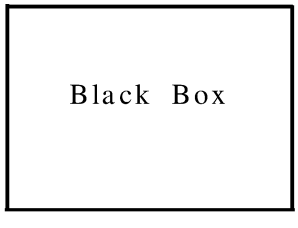


FIGURE 2

These ideas of *knobbifying* and *twiddling* occurred to us (though under different names) while we were exploring the fluidity of METAFONT letter programs by seeing how far the letters can be stretched and still retain their identities. One such set of experiments was based on a very simple METAFONT program for the letter 'b'.

```
EXPERIMENTAL LETTER b
---------------------
%Knobs of the experimental letter 'b"
new o,p,q,r,s,t;
    n=0.25;
            o=1.30;
            p=-0.05;
            q=0.65;
    r=0.70;
            s=0.30;
            t=0.05;
```



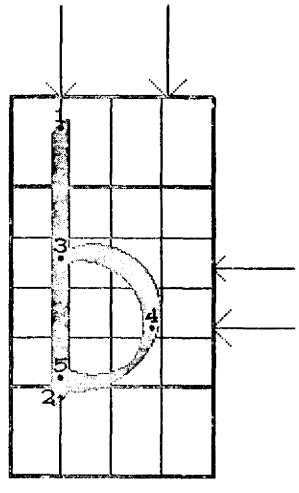


FIGURE 3

We had seven knobs (variables n, o, p, q, r, s, t) for twiddling and started our game by changing the value of 'r'-knob from positive to negative value. The result was this.

```
new o,p,q,r,s,t;
    n=0.25;
            o=1.30;
            p=-0.05;
            q=0.65;
     r=-0.70;
            s=0.30;
            t=0.05;
```

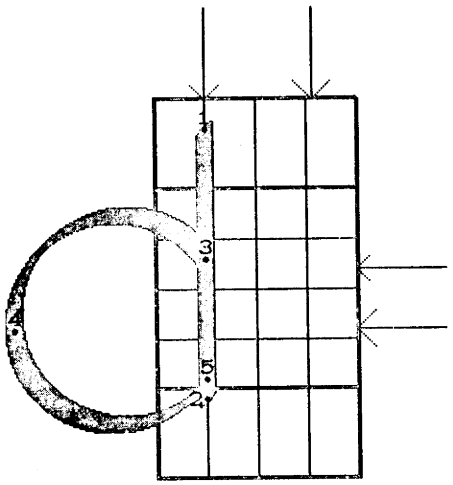


FIGURE 4

Naturally we were disheartened to get the letter **'d'** since we were expecting-a **'b'** instead. We rcalised that r-kncb is not the appropriate knob to fiddle with, therefore we changed the values of o- and p-knobs. To our surprise we got some picture which looks like the letter **'q'**.

```
new o,p,q,r,s,t;
    n=0.25;
            o=0.70;
            p=-0.65;
            q=0.65;
    r=-0.70;
            s=0.30;
            t=0.05;
```

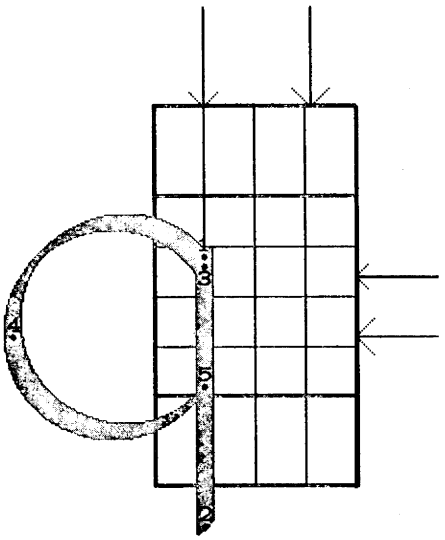


FIGURE 5

We put the r-knob back in its 'original position and this time we produced the letter 'p', but still not a new form of **'b'.**

```
new o,p,q,r,s,t;
    n=0.25;
            o=0.70;
            p=—0.65;
            q=0.65;
     r=0.70;
            s=0.30;
            t=0.05;
```

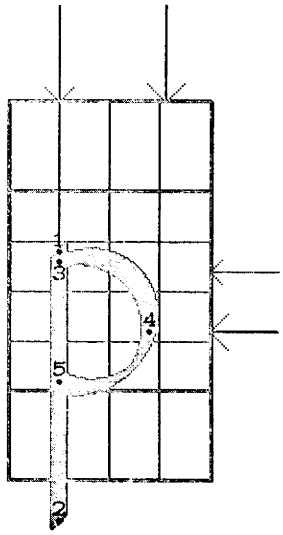


FIGURE 6

Then we went back to the previous value of r and also changed the value of the parameter p to its initial value. The result was a graphical symbol which resembles the letter 'a'.

```
new o,p,q,r,s,t;
    n=0.25;
            o=0.70;
             p=-0.05 ;
            q=0.65;
     r=--0.70';
            s=0.30;
            t=0.05;
```

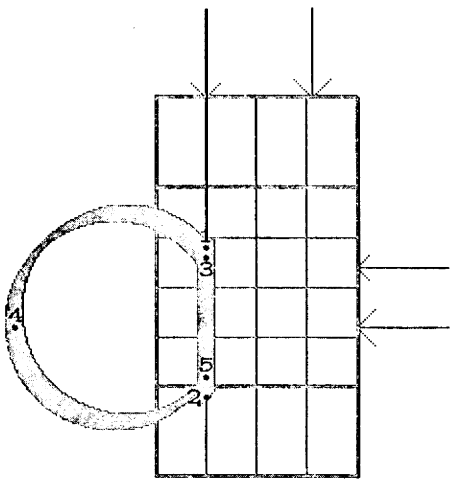


FIGURE 7

At this point, without thinking much (since thinking was not helping at all), we made the value of 'p' and 'q' equal to the value of 'o' and we got the letter 'C'.

```
new o,p,q,r,s,t;
    n=0.25;
            o=0.70;
            p=0.70;
            q=0.70;
    r= -0.70;
            s=0.30;
            t=0.05;
```

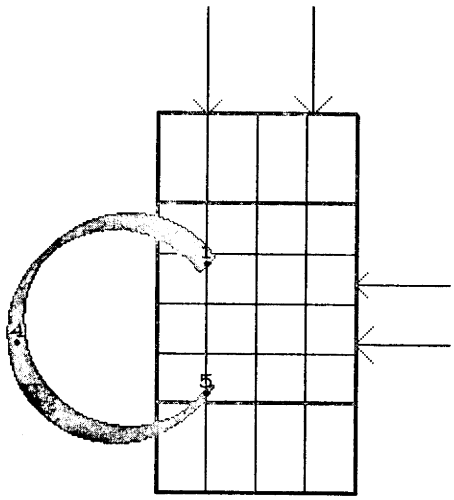


FIGURE 8

Still we had some hope (being optmists) of generating an exotic 'b', so we changed the value of 'r' to equal that of 'n'. The resulting drawing was a vertical line which resembles the letter 'I'.

```
new o,p,q,r,s,t;
    n=0.25;
            o=0.70;
            p=0.70;
            q=0.70;
    r=0.25;
            s=0.30;
            t=0.05;
```

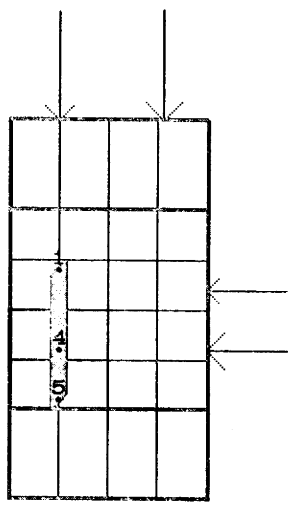


FIGURE 9

We then arbitrarily changed the values of o, p and r and we next, had the letter **'G'** in our possesion.

```
new o,p,q,r,s,t;
    n=0.25;
            o=0.35;
            p=0.05;
           q=0.70;
     r=-0.70 ;
           s=0.30;
           t=0.05;
```

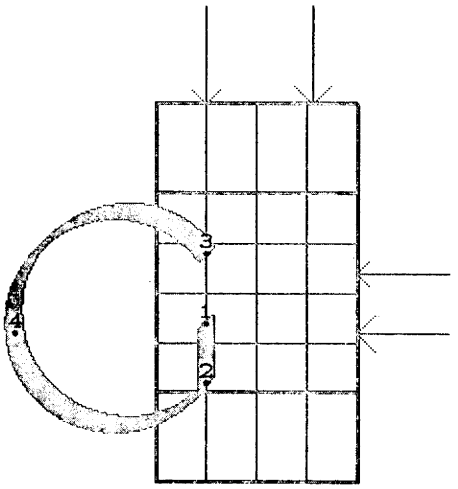


FIGURE 10

Finally we gave up our unsuccesful experiment when we produced an inclined straight line which doesn't resemble any letter shape.

```
new o,p,q,r,s,t;
    n=0.25;
            o=0.05;
           p=0.05;
            q=0.05;
    r=-0.70;
           s=0.30;
           t=0.05;
```

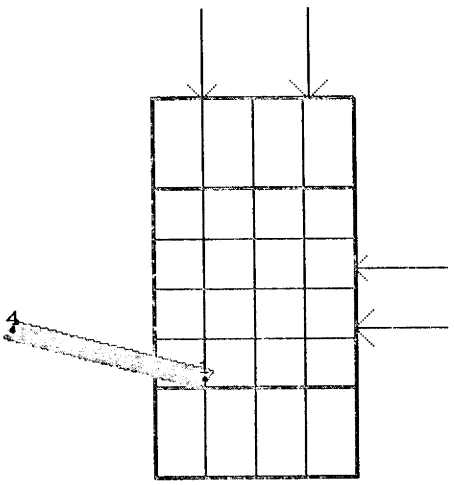


FIGURE 11

One might get the idea at this point that all the twenty six letters of the Roman lowercase alphabet could be produced by carefully choosing the parameter-values of this METAFONT program. We are sorry to state that it is not possible. I-Iowever, we have written another simple METAFONT program which did produce all the twenty six

lower case letters. The letters are added at the end of this report. An interested designer may also produce numerals, punctuation marks and other letter-forms using the program. Let us now open the black box and examine the program to see why

1. arbitrary change of the parameter values produce other letter forms and graphical patterns, instead of generating new forms of the letter 'b';
2. the program is unable to generate all the letters, no matter how distorted they may be.

```
          BLACK BOX
%Here is the inside circuitry of our BLACK BOX
%Drawing the straight vertical line
x1=n*l;  y1=o*h;
x2=x1;   y2=p*h;
draw 1..2;
%Drawing the arc segment
x3=x5=x1;  y3=q*h;
x4=r*l;    y4=s*h;
           y5=t*h;
draw (5..)3..4..5(..3);
fi.
```

There are two 'draw' statements; (1) therefore, the picture has atmost two segments. (2) In the first appearance, it seems that the first 'draw' command always generates a vertical line, but that is not the case. If the parameters 'o' and 'p' get same values, the picture transforms to a single point. (3) Similarly the second 'draw' command can generate either a point, a straight line or an arc depending on the parameter value, although we might have expected to get an arc all the time. (4) Moreover even if it is an arc, we don't know what would be the positional relationship between the vertical line and the circular arc. the program allows many (not 'all' due to the constraint x3=x5=x1) possibilities.

An improved program looks like this.

```
"New Program for the 'letter b'';
%
subroutine  error:
no  proofmode;
```

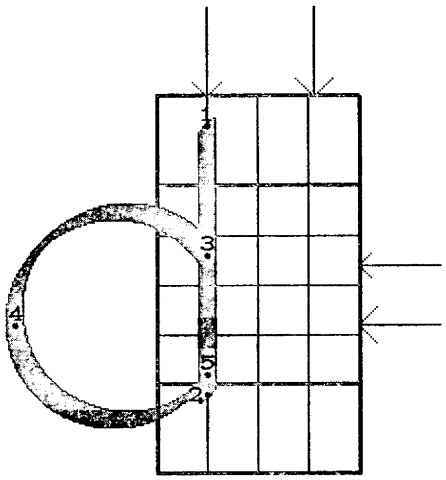
```
x50=x52=0.8l;   x51=x53=1.0l;
y50=y53=1.2h;   y52=y51=1.4h;
cpen; 3
draw 50..51;
draw 52..53;
new ef; ef=0;
fi.

%
%The external. Knobs
new  diRi,diIi,diIii,dIi,diiRii,diiIii,diiIiii;
diRi=0.25; %Parameters for 1st draw command
           diIi=1.30;
           diIii= -0.05;
           dIi=0.65; %Parameters for 2nd draw command
diiRii=0.70;
           diiIii=0.30;
           diiIiii=0.05;

%
%The Main Program
ef=0;
%
%Drawing the straight vertical line
x1=diRi*l;  yl=diIi*h;
x2=x1;   y2=diIii*h;
if y2y1:  new ef; ef=1;
else:     draw 1..2;
new ef; ef=0;
fi;

%
%Drawing the arc segment
x3=x5=x1;        y3=dIi*h;
x4=diiRii*l;     y4=diiIii*h;
                 y5=diiIiii*h;
if y5y3:  new ef; ef=1; fi;
if x4x3:  new ef; ef=1; fi;
if(y5-y2)(y1-y3): new ef; ef=1; fi;

if ef=1:call error;
else:   draw (5..)3..4..5(..3);
fi;
fi.
```

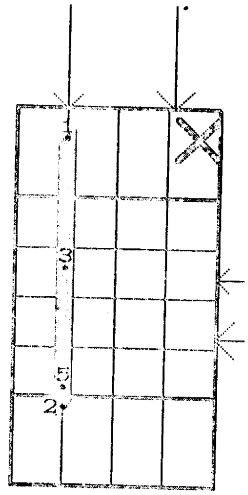
The second program differs from the first one in the following respects:

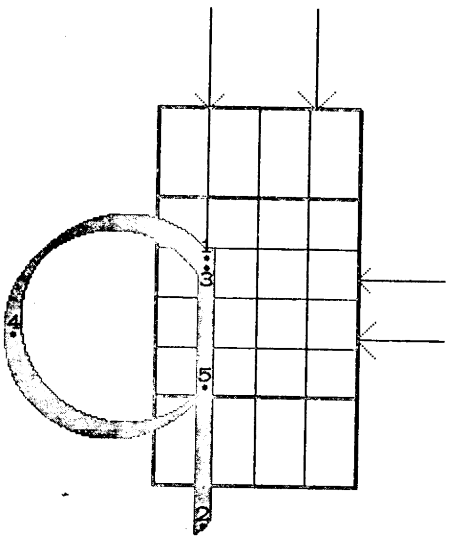
1. Each knob has an appropriate label on it. This enables the user to know which parameter is being twidelled. (It is like the labels VOLTJME, TONE, BALANCE on the knobs of a stereo cassette recorder). 'd' in the parameter name signifies draw, 'R' means x-value, 'I' means imaginary quantity or y-value, 'i' specifies first, 'ii' means second and so on. For example diIii means- 'in the draw command 1, parameter for the y-value of the second point'.

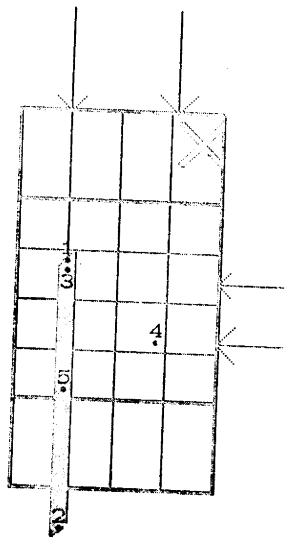
2. Four conditional branching have been added to check the type of strokes that each 'draw' command generates and their positional relationships. If the appropriate conditions do not hold true for improper choice of parameter values, the program raises an error flag and notifies the user which 'draw' command is causing problem.

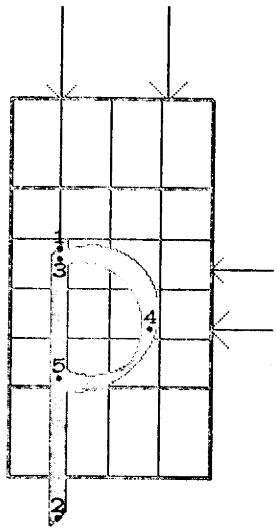
The result of changing the parameters in this new program with the old set of values is shown in the following pages. The error flag is the 'X' in the upper right–hand corner.

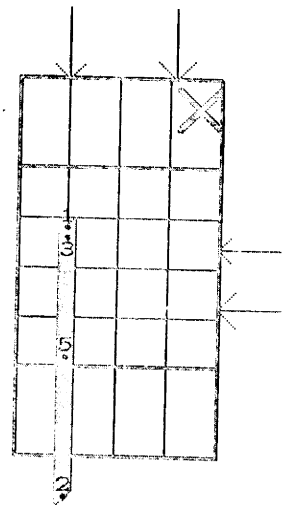

1
3
4
5
2

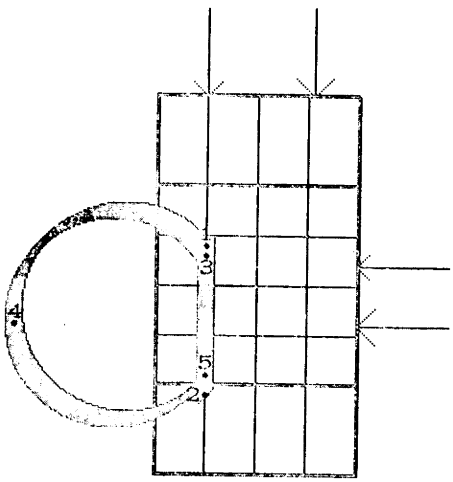

1
3
5
2

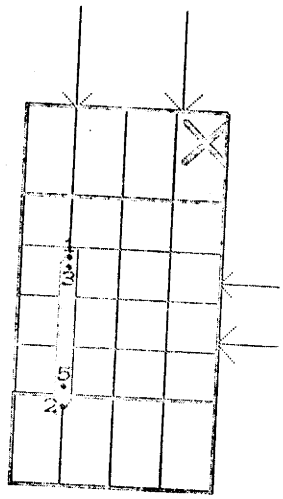

1
3
4
5
2

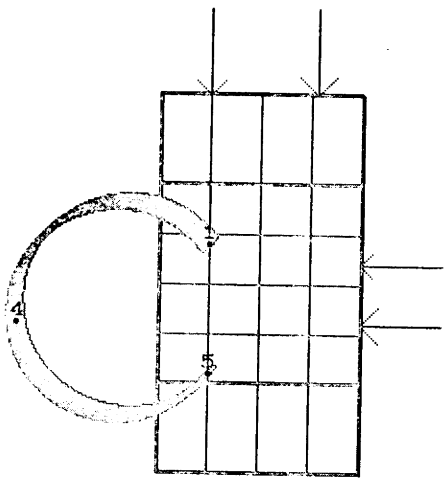

1
3
4
5
2

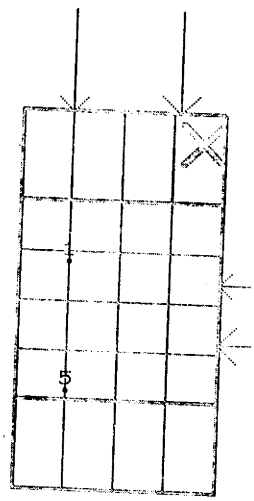

3
4
5
2


3
5
2


3
4
5
2


3
5
2


1
4
5


5

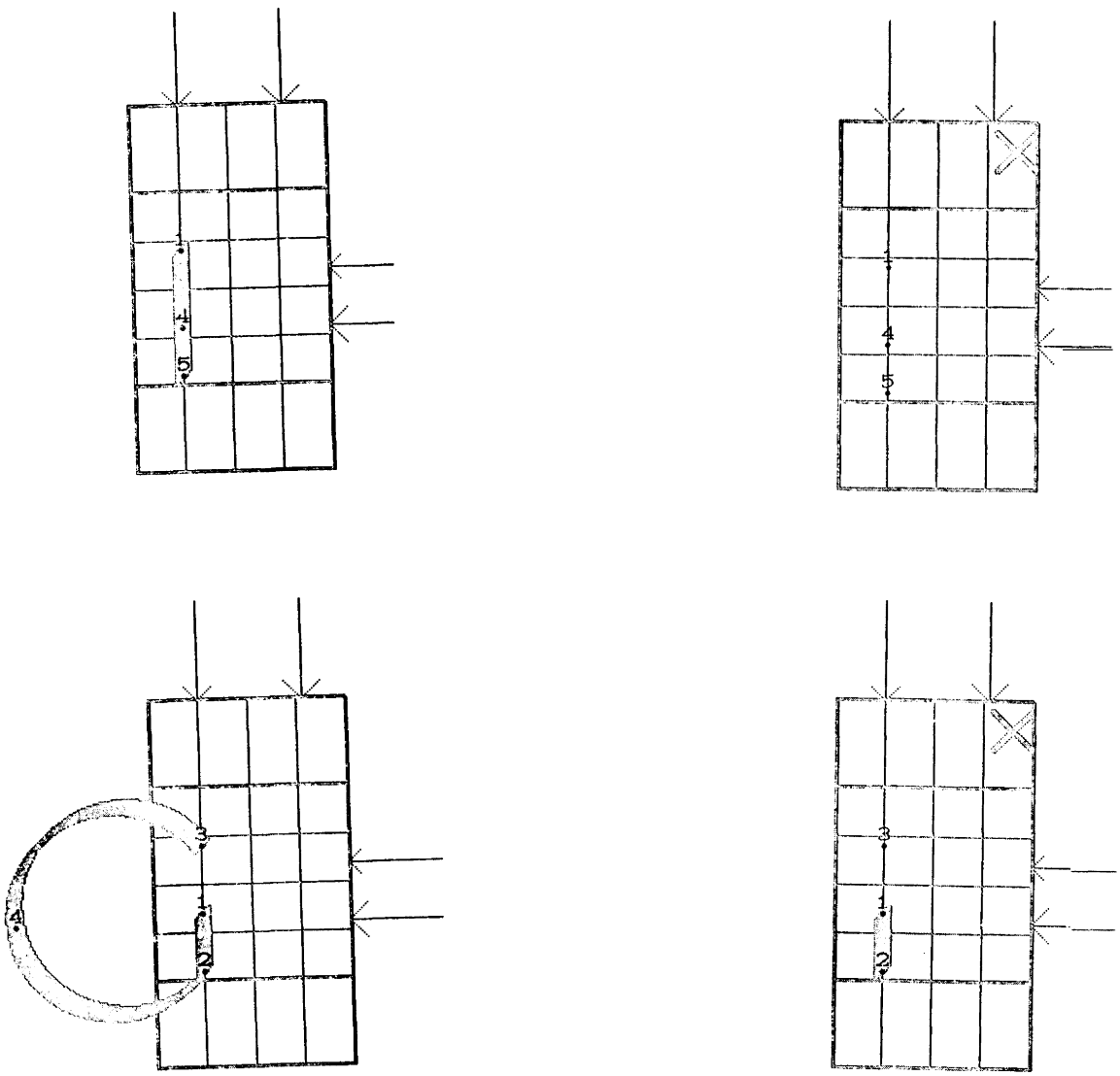


FIGURE 12

4
Syntactic Description of Letterforms

We have already cited Knuth's description of the meaning of a meta-font. It is a schematic description of how to draw a family of different forms of a letter-- not simply the drawing of the letter itself.

In this context Bigelow has pointed out that, 'Letters have structure which the system must comprehend'[3][4][5]. More precisely,

> *Any metafont system should provide a descriptive scheme in terms of which the structural features of individual letter shape can be efficiently described and talked about.*

The word '*efficiently*' used in the sense that the description should be understood both by the designer and the machine.

A convenient way to describe a picture is to use a two-level data structure. This is reasonable since we are trying to represent a 2-D image by a I-D language. More precisely, this is to describe a picture in terms of its subparts/subpictures and adjacency relationships among the parts. For example, Grimsdale et al. (cited in *Pattern recognition Techniques,* Ullman J.R., 1973) divided the letter 'R' into six subpictures and expressed the relational information with the help of an adjacency graph[42]. Their work is shown in the following figure. (Fig.13)

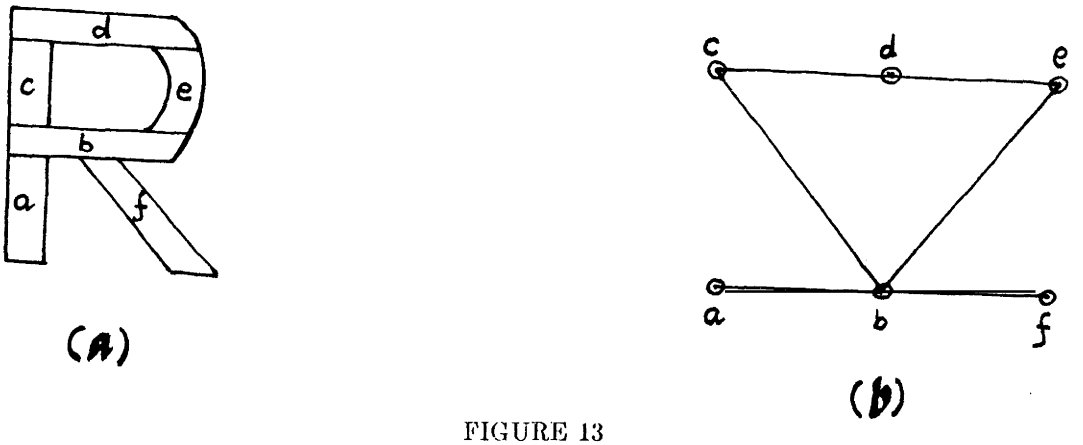


FIGURE 13

This concept of describing pictures in terms of primitive elements or subpictures and their relationship is analogous to the syntactic structure of languages. For example,

<sentence > → <subject phrase> <verb> <object phrase>

describes a relationship between the subparts of a sentence.

The letter shapes in any script are some specific *class* of pictures (by 'specific class' we mean to exclude the very general class of any complex image, which is difficult to handle). It is possible to set up a 'syntactic model' for description of letter forms.

Before proceeding further, let us take a brief look of the 'real world of letters' and examine HOW (or, whether any) syntactic structures are embedded within them.

5
Stroke Analysis for Lettershapes

Most lettering artists have described (directly or indirectly) how they make letters from simple brush or pen strokes. For drawn letters also, similar parts of different letters can be related. We observe that the letters h, n, m etc., or v, w, y etc., are grouped together at the time of design. Here is not the place to give a detailed account of the typographers' work, yet we offer one or two examples in this context.

Michel Harvey in his book *Lettering Design* has shown some of the characteristic brush strokes which are important for designing Roman Letters[22] (Fig.14)

FIGURE 14

He has also shown the sequence of brush strokes in a built-up letter,as shown in the following figure. (Fig.15)

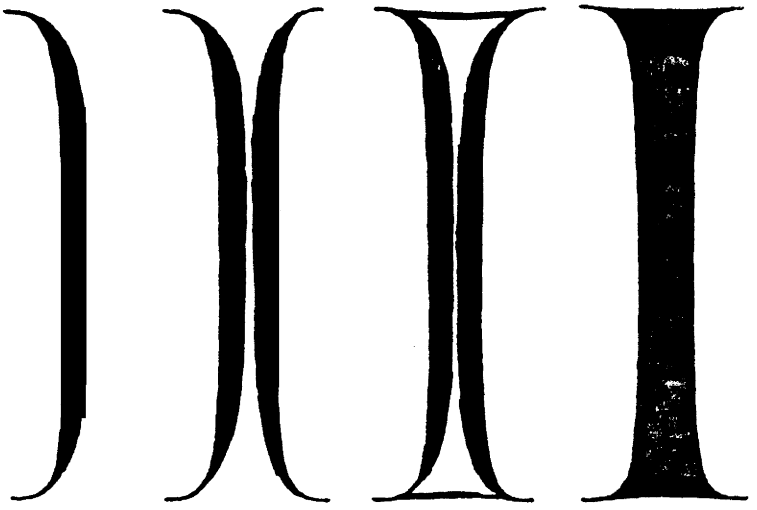
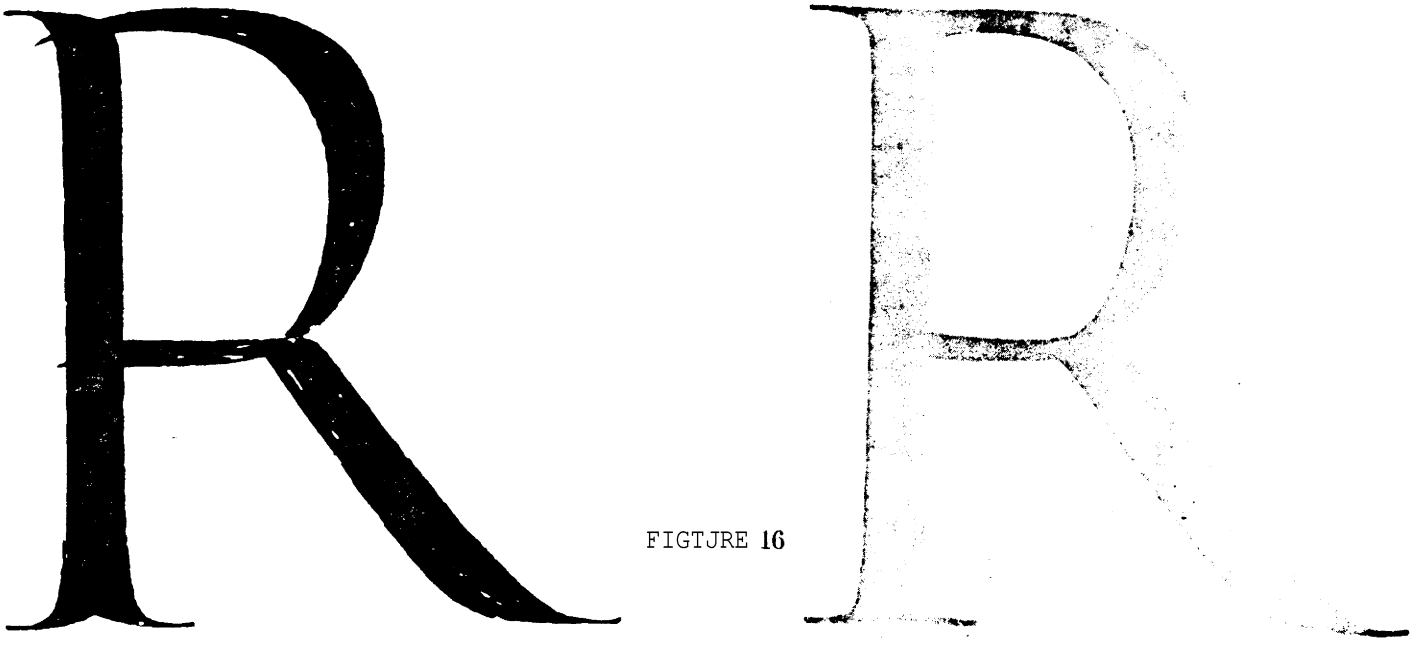


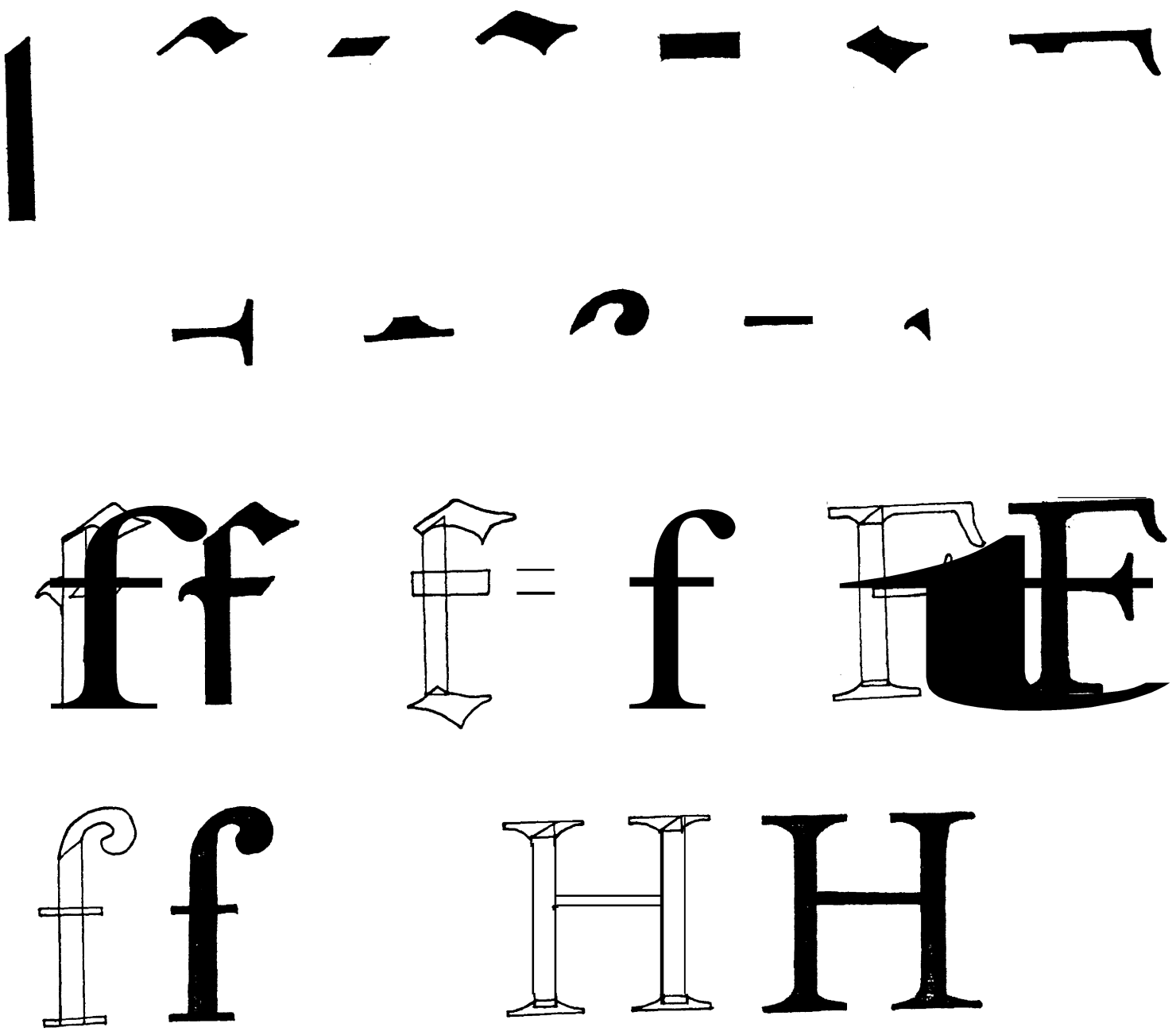
FIGURE 15

There is also the analysis of a roman 'R' made by E.M.Catich, in which economical brush strokes produce a letter of great elegance. (Fig.16)



FIGTJRE 16

We also did an experiment on some existing letter forms. Goudy in his book *The Alphabet and Elements of Lettering* presented his work on letters of different forms[19]. After carefully examining his alphabet designs, we found that all the letters of all the alphabets are actually formed out of some (not too many) basic primitive strokes. It appears to us that he designed his letters by the '*method of cutting and pasting*' and sometimes '*rubber-banding*'. Of course he didn't do that manually, but he appears to have adopted that conceptual technique. By experimenting with *cutting and pasting*, we were able to produce some of Goudy's letters. Sketches of our work is shown in the next page. One should note here how nicely different forms of a letter can be produced from a few primitive subparts. One should also note that *rubber-banding* technique has been adopted to produce the horizontal stroke of the letter 'H'.



FIGIJRE 17

6
Lettershape Description Language

A syntactic model for lettershape description can be roughly viewed like this:

1. This is a description model in terms of subparts or strokes present in the letter, and properties of and relations among these subparts.

2. The descriptive statements which generate a letter constitute a hierarchic system that can be represented by a multi-level graph.

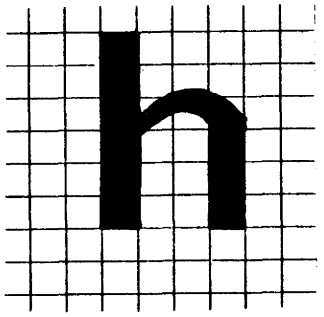
3. Labels are assigned to the different levels; Each label consists of two parts: i) the *NAME* of the subpart type (in other words, phrase name) and ii) the *ATTRIBUTE* values of the subpart. The NAME is actually a convenient way of referring abstractly to sets of invarient properties and property relationships; an *ATTRIBUTE* list is a set of modifiers or variable properties.

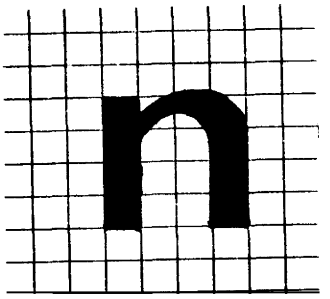
4] The *CONCATENATION OPERATORS*, which relates the subparts, have also two parts- the class name and the attribute lists.

To illustrate how the proposed scheme would work in an actual application, let us consider an example.

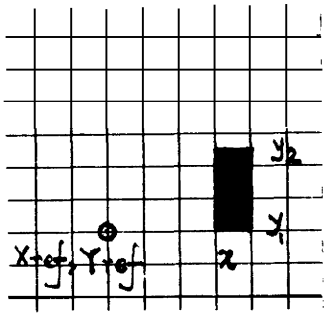
Example: We have to generate two letters 'h' and 'n' which are shown in Fig.18a.

We can figure out two subparts VI, (Vertical Line) and CS1 (Curve Segment 1) necessary to design the letters. They are shown in Fig.18b.

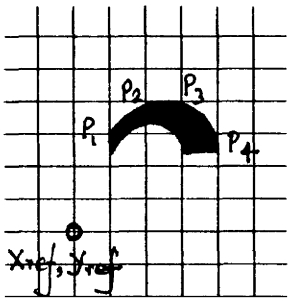




(a)

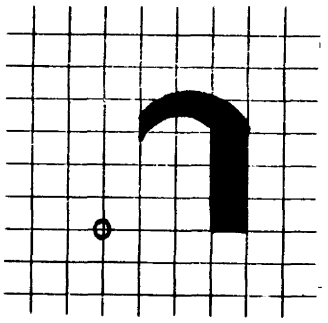


VERTICAL-LINE



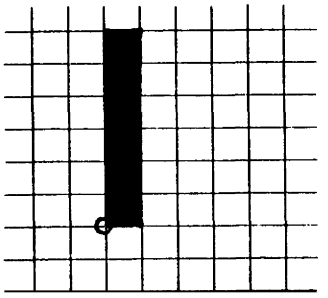
C-STROKE1

(b)



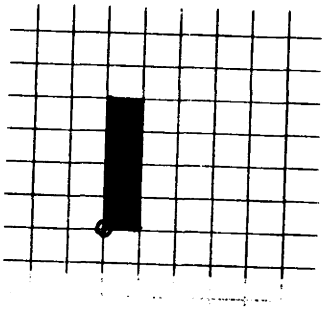
COM_STROKE1

(c)



V-STROKE2

(d)



V-STROKES

(e)

FIGURE 18

For the time being let us consider these subparts as the "first level" primitives. These subparts are represented in our model like this:

Label Class: SIMPLE STROKE

NAME	ATTRIBUTES (Pen type, dimension; Ref pt; Cordinates of pts)
1. VERTICAL-LINE	hflat, w1, w2; Xrcf, Yref; X, Y1, Y2
2. CURVE-STROKE	hflat, w1, w2; Xrcf, Yref; P1, P2, P3, P4

NOTE: For each subpart at every level, the reference point is necessary to exactly specify the adjacency relationship between two subparts.

Let the concatenation operators be as folows:

CONCATENATION OPERATORS

NAME	ATTRIBUTES (Distance— a positive value)
1. Left (lt)	dist
2. Right (rt)	dist
3. Above (ab)	dist
4. Below (bl)	dist
5. Adjacent (ad)	(no attribute)

The descriptive statements :

V-STROKE1 := VERTICAL_LINE (v1) ;

C-STROKE 1 := CURVE_STROKE (v2);

COM_STROKE1 := V_STROKE1 right by 1u above by Ou of C_STROKE1;

form a complex stroke COM_STROKE1 which is shown in Fig.18c.

NOTE: Unless cxplicitely specified, the reference point of COM_STROKE1 coincides with the reference point of C_STROKE 1, since conceptually we cut the V_STROKE1 from the graph sheet and pasted it on top of C_STROKE1; we didn't touch the C_STROKE1 at all.

COM_STROKE1 is one level higher than VL and CS1, and according to our model should take new attribute values. For example, we can write

Label Class: COMPLEX STROKE

NAME	ATTRIBUTES (Ref pt; X-stretch, Y-stretch; Rotation etc.)
1. COM_STROKE1	Xref, Yref; XX, YY; ROT; . . . .

It is not necessary to state the attribute value for the higher levels explicitely at all times. Unless specified, it assumes the default values of the system.

The next step is to generate the long vertical bar for the letter **'h'.** This can be easily done by changing the attribute values of VL. The statements for generating the letter **'h'** are:

V-STROKE2 := VERTICAL_LINE (v3);

LET-h := V-STROKE2 adjacent of COM_STROKE1;

Note that LET-h has higher level than COMP_STROKE1 and its attribute sets may be as follows:

Label Class: LETTER

NAME	ATTRIBUTES (Ref pt; X-stretch, Y-stretch; Rotation; L-sp, R-sp;...)
1. LET-h	Xref, Yref; XX, YY; ROT; lsp,rsp;....

Assigning a set of attributes to the elements of different class or category serves many useful purposes. For example, we can define one special stroke, say NULL-STROKE. This stroke can combine with any other stroke in a usual manner and does nothing but to put the second stroke in a higher level. Thus a simple stroke may be converted to a complex stroke without any visible change. By varying the attribute values of the complex stroke thus formed, we can rotate, shrink or expand it according to our need.

To be clearer, let us generate V-STROKE2 in the following manner:

COM_STROKE2 := V-STROKE1 adjacent of NULL-STROKE;
COM_STROKE3 := COM_STROKE2 (v4);

LET_h := COM_STROKE3 adjacent of COM_STROKE1;

We have simply changed the 'y-stretch' parameter in v4 to generate a stroke as shown in Fig.18d.

The letter 'n' can be generated in one of the above ways.

The multi-level graphs for the letter 'h' is shown in Fig.19.

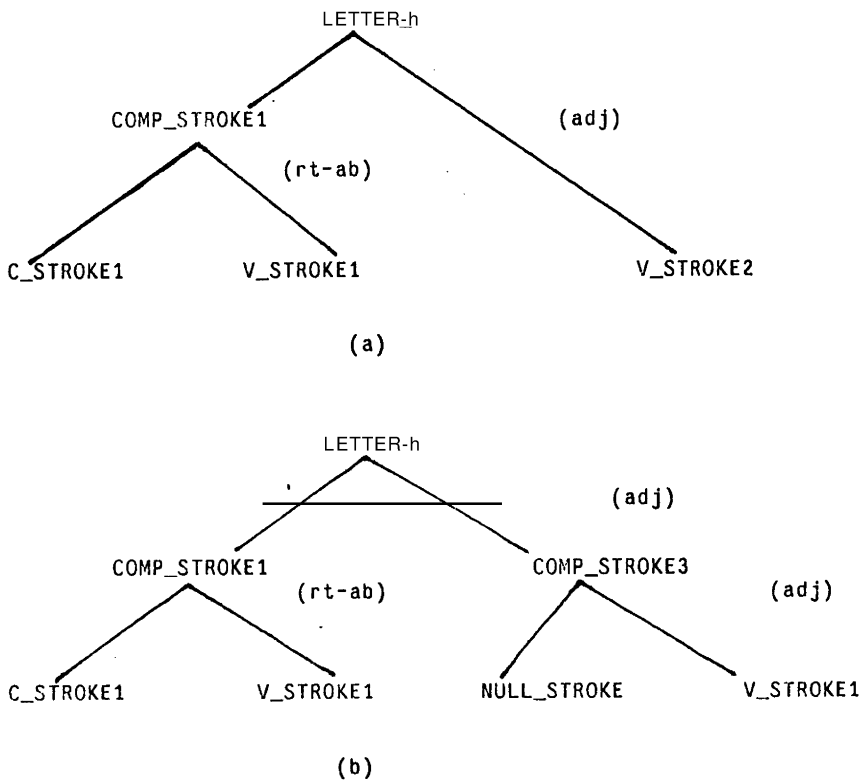


FIGURE 19

We have not discuss of how to generate the primitive strokes like VL or CS1. This can be done in the same way that Knuth's METAFONT generates them, or by some other similar systems. However, we can think of a model different from METAFONT , to treat the whole structure within our proposed formalism. Roughly this can be viewed like this:

1. The lowest level object type is the POINT. Its two permissible attributes are gray scale and Cartesian coordinates, i.e., POINT(x,y,g).

2. Few more primitive elements also can be thought of.
   VECTOR_DIRECTION(gradient , sense-of-movement),
   CURVATURE(val), etc.

3. The next higher level element types are simple strokes like
   STRAIGHT_LINE(p1,p2), VERTICAL_LINE(x,y1,y2),
   HOROZONTAL_LINE(x1,x2,y),
   ARC(p1,p2,p3) / ARC(p1,p2,curvature),
   SMOOTH_CURVE(p1,p2,...,pn) / SMOOTH_CURVE(p1,d1,...) etc.

7
A Phrase–Structured Grammar for LDL

Our proposed model of heirarchical structure of subpatterns is analogous to the syntactic structure of a language.

The central idea of formal language theory is the generation and/or analysis of the strings (sentences) of languages in terms of grammars, typically *Phrase–Structure Grammars*. The structural description of the language in terms of a grammar is called a *syntax* of the language. Analysis in terms of this structure is called *syntactic analysis*, or parsing.

Accordingly, a *Phrase–structure grammar* is a four-tuple

$G = (V_n, V_t, P, S)$.

V_n: nonterminals/variables
V_t: terminals
$S \in V_n$: Start/sentence symbol
P: productions/rewriting rules

$V_n \cap V_t = 0$

Productions have the general form
$\alpha \mapsto \beta$

where α, β arc strings over $V_n \cup V_t$, and '$\mapsto$' is read *"is replaced by"*.

In formal language theory, the only relation between the elements in a string is *concatenation*, i.e., the juxtaposition of adjacent elements. The most crucial point involved in adapting the techniques of formal language theory to letter shape design is the generalisation of this simple notion to include the other relationships.

With the help of the old example, we shall show what the phrase-structure grammar will look like for our proposed model.

Our Old Example of Generating Letter h

We shall denote the *concatenation operators* by the symbol '*'.

$*$ = { lt-ab, lt-bl, rt-ab, rt-bl }

V_n = { COM_STROKE1, COM_STROKE2, COM_STROKE3, LET-h }

V_t = { C_STROKE1, V_STROKE1, V_STROKE2, NULL-STROKE}

Writing p-name for non-terminals or phrase-name and w-name for word or terminal-name, we can easily state the productions for our proposed model as follows:

P :

$S \mapsto$ p-name

p-name $\mapsto$ p_name*p_name

p-name $\mapsto$ p_name*w_name

p-name $\mapsto$ w_name*w_name.

Therefore,

case A:

$S \mapsto$ LET-h

$\mapsto$ COM_STROKE1*V_STROKE2

$\mapsto$ (C_STROKE1*V_STROKE1)*V_STROKE2.

case B:

S $\mapsto$ LET-h

$\mapsto$ COM_STROKE1*COM_STROKE3

$\mapsto$ (C_STROKE1*V_STROKE1)*(V_STROKE1*NULL_STROKE).

Extention of the idea of Hierarchic Structure

The idea of this hierarchic structure, and assigning different attribute sets at different levels, can be extended from the meta-font description of a letter to a system of typesetting a book or journal.

For example, the letters form a WORD whose attributes may be spacing, position, etc. The words form a LINE-OF-TEXT whose attributes are *left/right margin, indentation, position in a page, projection* and so on. Similarly, a PAGE is made up of lines (non-empty or empty) with attributes like *shape, position in the white space* and so on. Selection of attribute sets at different levels depends on the type of requirements of the artist, the ease of setting the values for different attributes, and finally on their efficient implementation in the machines.

8
Mathematics for Decomposition

Basic Postulates

The basic postulates of our *Decomposition Mathematics* can be stated as follows:

1. A collection $f_1, f_2, \ldots . f_m$ of figures is a *decomposition* of a pattern P in m pieces iff

   $$P = \cup_i f_i,$$

   any two pieces f_i and f_j are *distinct,* but do not necessarily have *disjoint* interiors. By the term figure we mean a bounded region in a 2-D plane, hereafter which will be mentioned as 'region'.

2. For any allowed transformation, or concatenation of' transformation g, there may be some f_i and f_j, where

   $$f_j = g f_i.$$

3. For any g (as described in Postulate 2),

   $$gP = \cup_i g f_i.$$

4. The union operation is defined as *commutative,* as well as *associative,* i.e.,

   $$f_i \cup f_j = f_j \cup f_i ;$$

   $$f_i \cup (f_j \cup f_k) = (f_i \cup f_j) \cup f_k .$$

Due to the Postulate 2, there arise two types of regions-- pure regions and *transformed regions.* The second type is generated by applying some allowed transformation(s) on the pure regions.

If F denotes the set $\{f_1, f_2, \ldots, f_m\}$ for figure P (P is any character, i.e., letter shape or symbol in an alphabet), then the set E which is the *union* of all F's may be termed as *Element Set* for a particular font of an alphabet. The elements of E are either *pure* or *transformed regions.* There then exists a subset (most of the time, a proper subset) B of E, whose elements are the *pure regions. B* is evidently the *Pattern Primitive Set* for the alphabet.

A Few Important Considerations

The following considerations may serve as a guide-line for selecting the *pattern primitives*:

1. The number of elements in B should be as few as possible. The set B can be defined as 'good' set if the number of elements are less than the number of characters in the alphabet.

2. Each element in B should be conceptually clear to the designer.

3. The representation of the elements should be compact, but computationally convenient for the machine.

Graph Representation Techniques

It is often helpful to represent each character by a graph. This graph representation partially enables us to comprehend how a decomposed letter will be stored inside the machine in an abstract way. The data structure for picture representation becomes easier from the graph of the picture. In fact, it is a type of notational system for representing shapes.

The graph formation technique is as follows:

1.. Represent each region as a point.

2. Draw a line between two points if the corresponding regions are adjacent.

For example, let /, \ , — are three regions. Following the above procedure the graph of the letter A can be drawn as follows—

FIGURE 20

All graphs thus formed are planar graphs, and in a strict sense they are all *connected* graphs as well. This second property at first seems to be puzzling for the

letters **i, j,** or for the punctuation marks !, ? etc., but one should reveal the fact that the disconnected dot in each of the characters has a fixed adjacency relationship with the rest of the figure and cannot be placed arbitrarily in the plane.

In a connected graph, a vertex is said to be an *articulation point* if the -vertex can be split to yield an unconnected graph. One very interesting characteristic is that, if any region which represents an articulation point in a adjacency graph is removed, the letter shape loses its identity; however, the converse is not true. (The experiment has been done with. simplex pattern primitives). For example, let us consider the graph of the letters H, G and **m.** The following component regions are chosen—

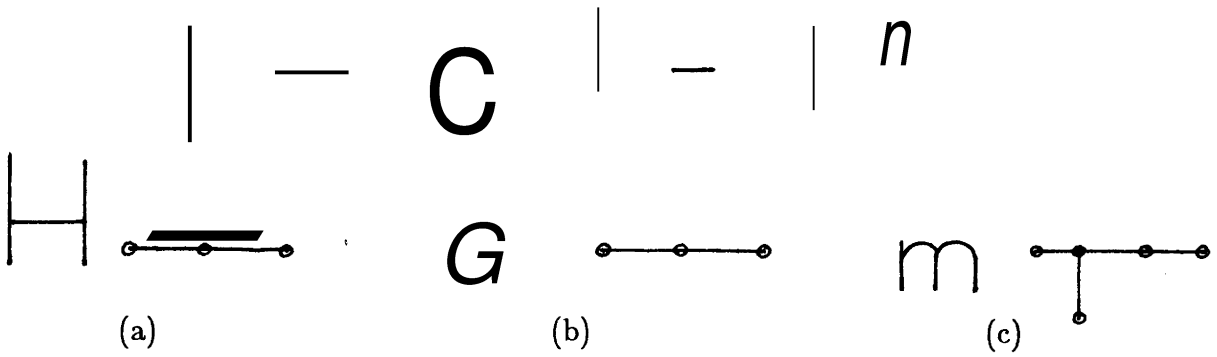


FIGURE 21

If WC remove the middle articulation point the corresponding patterns would look like this—

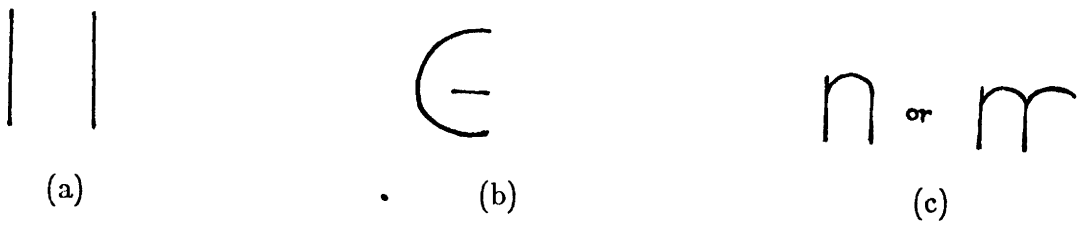


FIGURE 22

The adjancy graph technique as described above is one method of checking the recognizability of characters under the changes of parameter values. However, the graphs of many chracters may be *isomorphic* (as it is evident from Fig.21(a) and (b) above) and isomorphism is an unnecessarily complicated property for distinguishing

between characters from their respective graphs.

This problem can be partly solved by assigning names to each distinct regions; the graphs thus formed is a *Labeled Graph,* where the points are distinguished from each other by names such as v_1, v_2 etc. For example, the graphs for H and G may be redrawn in the following way.

(a)graph for H (b)graph for G

FIGURE 23

Still we cannot avoid the problem completely. Let us consider the example of the letters p and q. Suppose we have two regions

FIGURE 24

The graph for both letters will look alike.

FIGURE 25

To avoid such situations, we propose the notion of *links with attributes.* Our graph representation technique can be described very briefly as follows:

1. Rules are prescribed for constructing a graph from any given pattern and regions.
2. Further rules are prescribed for assigning label to each point.
3. Further more, rules are prescribed for assigning attributes to each line of the graph.

For example, the graphs for p and q can be differenciated easily if WC have one link-attribute called 'left *of'*. If this attribute is represented by the following symbol

FIGURE 26

the resulting graphs would be as follows:

(a)giaph for p

(b)graph for q

FIGURE 27

Obviously we need more than one attribute: *left, top-left, middle,* etc are few possible at tributes.

It is interesting to note that Frutiger (*Die Kapitalform und die Minuskelform,* pp 55 - 59 of *Der Mensch und Seine Zeichen,* Vol. II: *Die Zeichen der Sprachfixierung,* A. Frutigcr, II. Heiderhoff, D. Stempel AG, Frankfurt, 1979) has also used a method similar to *link with attributes* to show the movements of strokes in constructing roman majuscules.

For example, he has categorised seven different type of strokes where the No.4 and No.6 are shown as follows-

FIGURE 28

However, one cannot adopt his method since it frequently produces *pseudo-graphs*. His seventh category is a loop. Loop-multigraphs are considered as *pseudo-graphs* since we have the restriction "irreflexive" in the definition of graphs.

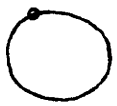


FIGURE 29

Allowed Transformations

This section would remain incomplete without saying something about allowed trans-forrnations. To be very brief', all the transformations applied on the pure regions should be conceptually clear to the designer. The geometrical transformations such as translation, scaling and rotation are easy enough to comprehend. Reflection-transformation also should be considered.

9 References

1. J.K. Aggarawal, R.O. Duda, A. Rosenfeld, (Editors), *Computer Methods in Image Analysis*, IEEE PRESS, New York, 1977.

2. J.H. Benson, A.G. Carey, *The Elements of Lettering*, McGraw-Hill Book Co., Inc., 1950.

3. CA. Bigelow, *Technology and the Aesthetics of Type, Maintaining the Tradition in the Age of Electronics*, The Seybold Report, August 24, 1981.

4. C.A. Bigelow, *The Principles of Digital Type, Quality Type for Low, Medium and High Resolution Printers, Part I*, The Seybold Report, February 8, 1982.

5. C.A. Bigelow, *The Principles of Digital Type, Quality Type for Low, Medium and High Resolution Printers, Part II*, The Seybold Report, February 24, 1982.

6. J.A. Bondy, *The 'graph theory' of the Greek alphabet*, Graph Theory and Applications, Y. Alavi et al., eds , Berlin, Springer-Verlag, 1972.

7. G.T. Buswell, *How People Look at Pictures*, U. of Chicago Press, Chicago, 1935.

8. E.M. Catich, *The Origin of the Roman Serif*, The Catfish Press, Davenport, Iowa, 1968.

9. S.K. Chang, K.S. Fu, (Editors), *Lecture Notes in Computer Science, Pictorial Information System*, Springer-Verlag, 1980.

10. C.H. Chen, (Editor), *Pattern Recognition and Artificial Intelligence*, Academic Press, Inc., 1976.

11. P.J.M. Coueignoux, *Generation of Roman Printed Fonts*, Ph.D.Theses, Dept. of Electrical Engineering, M.I.T., June, 1975.

12. James Craig, *Designing with Type*, Watson-Guptill Publication, New York.

13. Albrecht Durer, *Underweysung der Messung mit dem Zirckel und Richtscheyt*, Nuremberg, 1525. An English translation of the section of the alphabets has been published as Albrecht Durer, *Of the just shaping of letters*, R.T. Nichol, trans., Dover, New York, 1965.

14. Adrian Frutiger, *Type Sign Symbols,* ABC Edition, Zurich, 1980.

15. K.S. Fu, (Editor), *Digital Pattern Recognition,* Springer-Verlag, 1976.

16. P.K. Ghosh, *Introducing Interactive* Computer *Drawing to the Students of Calligraphy and Art with the help of PALATINO system,* CSI Bangalore Report, India, April 1982.

17. P.K. Ghosh, R. Sujata, *A Graphical Approach to Typesetting,* Computer Society of India Annual Convention, January 1982.

18. Frederic W. Goudy, *Typologia: Studies in type design and type making with comments on the invention of typography, the first types, legibility and fine printing,* Berkeley, Calif., Univ. of California Press, 1940.

19. Frederic W. Goudy, *The alphabets and Elements of Lettering,* Dover, New York, 1963.

20. Nicolete Gray, *Lettering as Drawing,* Taplinger, New York, 1982.

21. F. Harary, *Typographs,* Visible Language 7, 1973.

22. Michael Harvey, *Lettering Design: Form and Skill in the Design and Use of Letters,* Bonanza Books, New York, 1980.

23. D. Hofstadter, *Meta Magical Themas,* Scientific American, September, 1982.

24. J.E. Hopcroft, J.D. Ullman, *Formal Languages and Their Relation to Automata,* Addison-Wesley Publ. Co.,Massachusetts, 1969.

25. Edward Johnston, *Writing and Illuminating and Lettering,* Pitman, London, Taplinger, New York, A Pentalic Book, 1971.

26. R.F. Jolly, *Synthetic Geometry,* Holt, Rinehart and Winston, Inc., 1969.

27. Donald E. Knuth, *The Computer Modern Family of Typefaces,* Stanford Computer Science Report, STAN-CS-80-780, January 1980.

28. Donald E. Knuth, *The Concept of a Meta-Font,* Visible Language, Volume XVI, No.1, 1982.

29. Donald E. Knuth, *TeX and METAFONT: New Directions in Typesetting,* Digital Press, Bedford, Massachusetts, 1979.

30. Donald E. Knuth, The *TeX Book*, a pre-preliminary edition for people who can't wait, Stanford Computer Science Department, December 1982.

31. B.S. Lipin, A. Rosenfeld, (Editors), *Picture Processing and Psychopictorics*, Academic Press, New York-London, 1970.

32. M.V. Mathews, Carol Lochbaum and Judith A. Moss, *Three Fonts of* Computer Drawn *Letters*, Communications of the ACM 10, 1967.

33. A.D. McGettrick, *The Definitions of Programming Languages*, Cambridge University Press, 1980.

34. F. Nake, A. Rosenfeld, (Editors), *Graphic languages*, North-Holland Publishing Co., 1972.

35. R. Narasimhan, *Syntax-Directed Interpretation of Classes of Pictures*, Communication of the ACM, 9,3, 1966.

36. Friedrich Neugebauer, *The Mystic Art of Written Forms.* Translation by Bruce Kennett, Neugebauer Press, Boston, 1980.

37. Dan Pedoe, Geometry *and the Liberal* Arts, St. martin's Press, New York, 1976.

38. Emil Ruder, *Typography*, Visual Communication Books, Hastings House, Publ., Inc, New York, 1981.

39. D. Secrest, J. Nievergelt, (Editors), Emerging Concepts in Computer *Graphics*, W.A. Benjamin, Inc., 1968.

40. A.C. Shaw, *The Formal Description and Parsing of Pictures*, SLAC Report No.84, Stanford University, 1968.

41. N.S. Sutherland, *Visual Discrimination in Animals*, Brit. Med. Bull., 20, 1964.

42. J.R. Ullman, *Pattern Recognition Techniques*, Cranes Russak and Co., Inc., New York, 1973.

43. Herman Zapf, *Manuale Typographicum*, MIT Press, 1970.

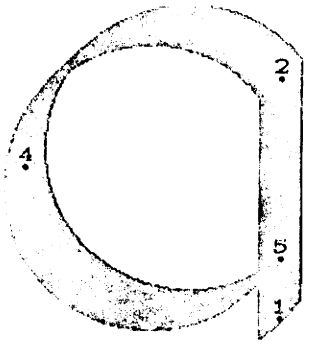
Appendix A

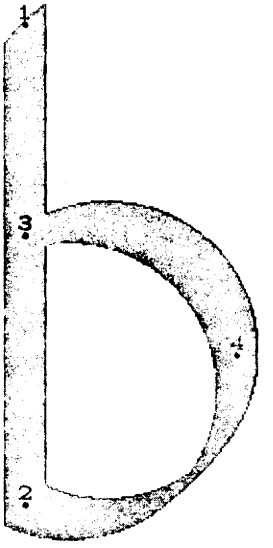
Appendix A contains the METAFONT program that generates all the twenty six lower case letters of Roman alphabet. An interested designer may also produce numerals, punctuation marks and better-looking letterforms using the program. This is particularly important, since it demonstrates how flexible a computer program can be, so that it can generate not only many different forms of a single letter, but can also produce the whole alphabet From a single rigorous definition. This METAFONT program is a very simple one which has sixteen variable parameters or knobs For twiddling. It is possible to write a more complex program that can produce beautiful letters according to traditional notions of aesthetics.

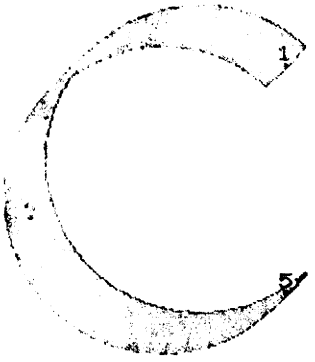
Appendix A

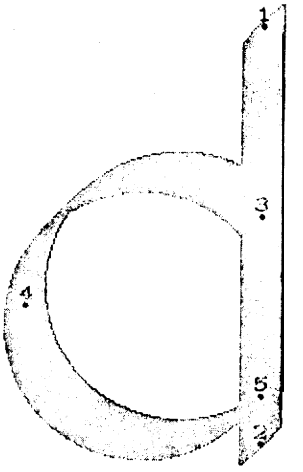
```
" An experimental program for all letters ";

drawdisplay; proofmode;
u=300;                  % Setting the unit

% To choose a flat pen for drawing
call ellipticalpen(0.10u,0.03u,45);
spen(ellipsa,ellipsb,ellipsc,0,0,0,0);

% Variable parameters/knobs
new a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p ;


        a=0.30;  b=0.70;
        c=0.30;  d=0.00;

        e=0.19;  f=0.50;
        g=0.30;  h=0.50;
        i=0.50;  j=0.50;

        k=0.30;  l=0.00;
        m=0.30;  n=0.00;
        o=0.30;  p=0.00;


% MAIN PROGRAM (to generate all letters)

% A Flat pen for drawing

% To draw a straight through points 1 and 2

x1=a*u;  y1=b*u;
x2=c*u;  y2=d*u;

          draw 1..2;

% To draw a circular arc through pts 3,4,5

x3=e*u;  y3=f*u;
x4=g*u;  y4=h*u;
x5=i*u;  y5=j*u;

          draw (5..)3..4..5(..3);

% To draw another circular arc through 6,7,8

x6=k*u;  y6=l*u;
x7=m*u;  y7=n*u;
x8=o*u;  y8=p*u;

          draw (8..)6..7..8(..6);

fi.
```

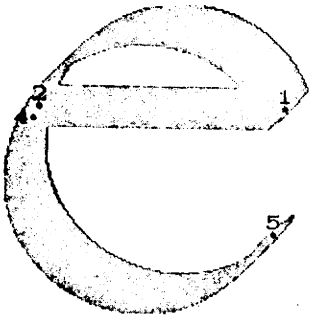
Appendix A

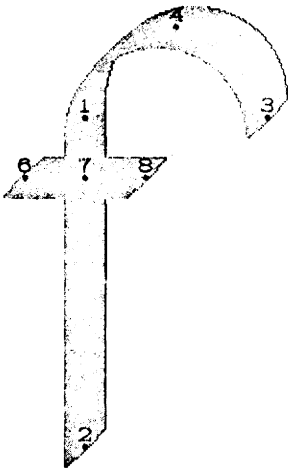




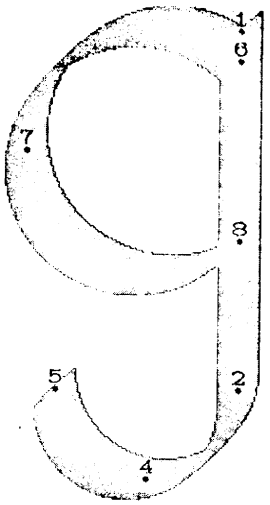


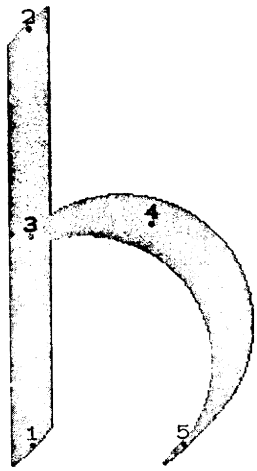


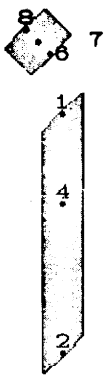


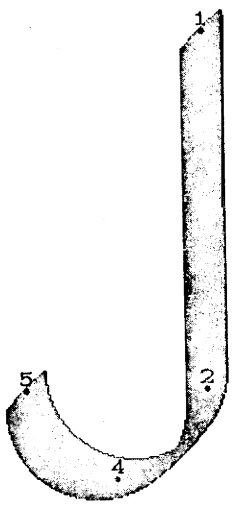


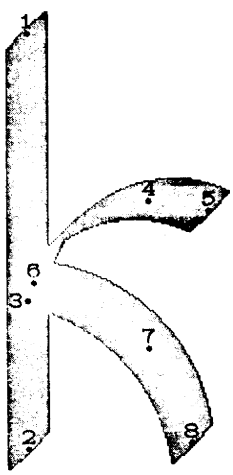
Appendix A



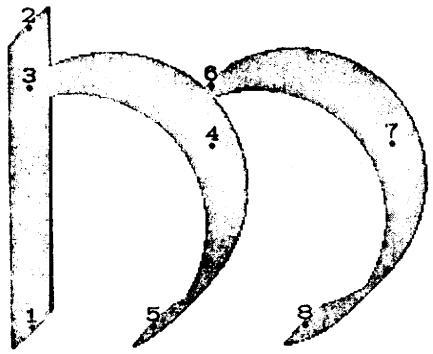


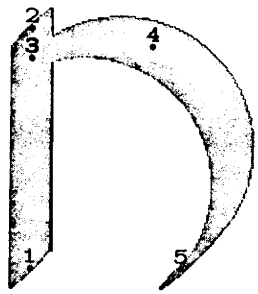


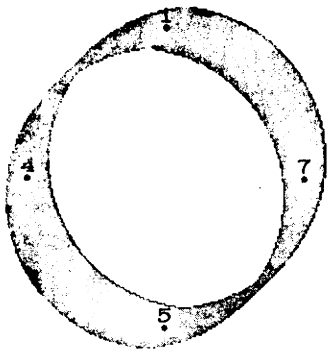


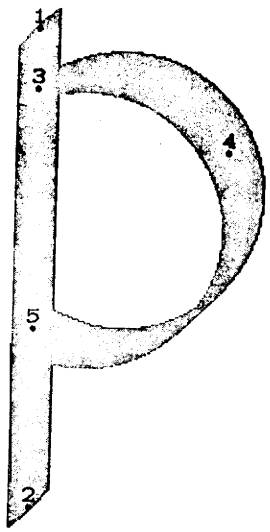


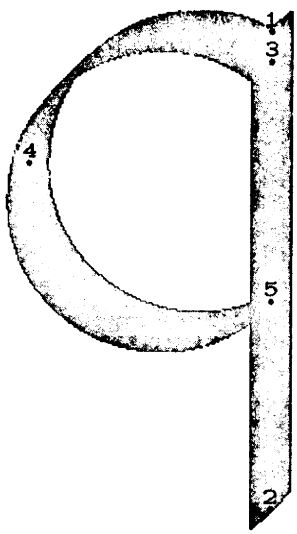
Appendix A

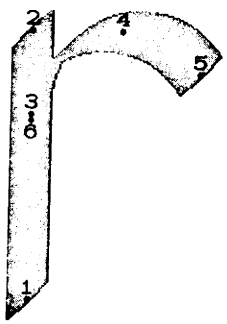


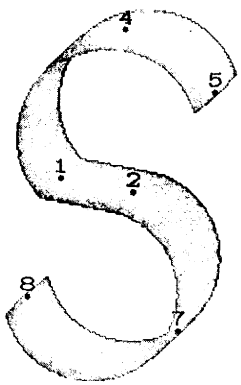


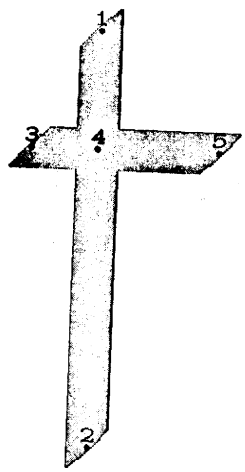


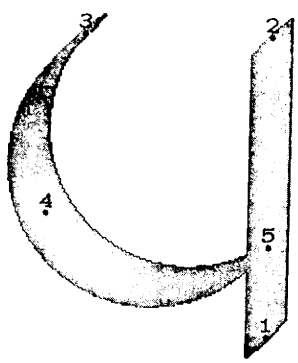


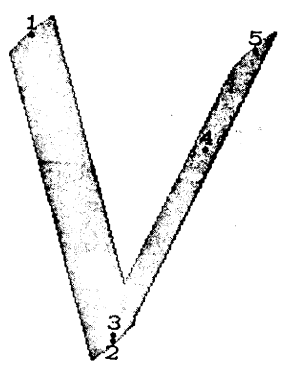


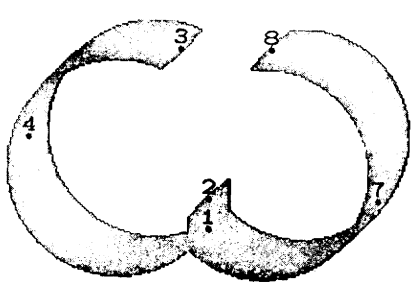


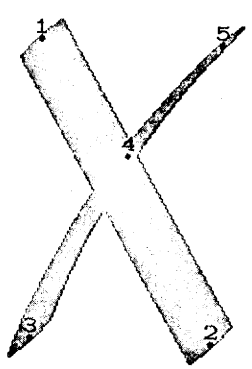

4
5
1
2
8
7


1
3
4
5
2


3
2
4
5
1


1
5
4
3
2


3
8
4
2
1
7


1
5
4
3
2

. Appendix A