CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

A computerized multi-font character generation system

Assimakis Panoutsopoulos

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

July 1984

ABSTRACT

A computerized multi-font character generation system

Assimakis Panoutsopoulos

In this thesis a character generation system is proposed. The proposed system consists of two main processes: a) the encoding or description process (analysis), and b) the decoding or generation process (synthesis).

In the first process, input digitized character representations in the form of binary (black and white) matrices are transformed into graph representations in the form of vertices and edges. This is achieved by developing a variety of algorithms, including: smoothing, thinning, point labelling and reduction, and graph traversal.

In the second process, different type styles (a total of five) are generated from the code, i.e. graph representations. This is achieved by applying some transformational operations and/or rules on the code depending on the specific type style. Family font

variations in size, boldening and orientation are also achieved. Tests are conducted on standard and non-standard character fonts, and experimental results show the effectiveness of the proposed method.

To the memory

of my beloved father

Acknowledgements

I gratefully acknowledge the guidance, advice, financial and otherwise support of my supervisor Dr. C. Y. Suen, which enabled the successful completion of this thesis.

Special thanks to Mr. Q. N. Ong of the Computer Science Image Processing Laboratory, Concordia University for his efforts to digitize the various character fonts.

The cooperation, assistance and services provided by the faculty members and staff of the Computer Science Department, staff of the Computer Center, and staff of the SEL library are deeply appreciated.

I also wish to express my sincere gratitude to many of my friends and relatives for their companionship, assistance, moral and otherwise support during the course of my studies.

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada and the Department of Education of Quebec.

Last, but not least, thank you mother for your devotion to your children.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Thesis organization

The thesis is organized into six chapters, a glossary and a bibliography.

Chapter 1 begins by giving a brief description of the organization of the thesis (in this section). Then, it briefly describes the necessity of human written communication, the development of the alphabet, the impact of writing into human affairs, the psycological factor and artistic aspects as well as the three eras of the letterform design.

Chapter 2 starts by describing certain aspects of character fonts; it states some criteria that characterize a computer-aided character generator, and discusses some effects and perspectives of the digital evolution into type design. Then, a review of the past computer-based character generation systems is given. Finally, the basic structure of the proposed system will be discussed.

In chapter 3, the first process (i.e. the encoding or generation) of the proposed system is given. A variety of algorithms (i.e. thinning, smoothing, point labelling, graph traversal, etc.) are presented. Examples, possible code inaccuracies adjustments, and the code (in the form of a graph) as maintained by the program (linked list) are also included.

In chapter 4, the second process (i.e. the decoding or generation) of the proposed system is given. The properties and derivation of five different type styles (from a single code) are discussed. The different types of pens and serifs in different sizes are also discussed. Finally, the type style variations in size, boldening and orientation are also included.

Chapter 5 presents some experimental results. The coding size, and the time taken for code derivation and generation are given. Illustrations of the different type styles (demonstrated in chapter 4) and of some standard character fonts (i.e. Orator, Gothic, Courier, Pica) in different variations are presented.

Chapter 6 serves as a recapitulation of the thesis. The contributions and the relative merits of the proposed system are described. The applicability and some suggestions for further developments are also discussed.

1.2 On the written language

Since humans live in socially organized groups, they learn to participate in the behavior patterns of their individual cultures. This behavior exists in the interactions among humans as well as between humans and their environment, and often requires the use of language as the principal means of communication. Language can be expressed in different forms: verbal, written or gesture. Verbal and gesture languages are of momentary duration (i.e. restricted in time) and function adequately only among communicators who are in close proximity (i.e. restricted in relation to space). The written language being a system of human communication by means of visible, conventional, graphic symbols, was originally developed to release the other forms of languages from the mentioned constraints.

In [Hay30] a chronological review is given, and covers the origins of writing as well as the development of the art of printing from historical to modern times. After thousands of years of evolution and refinement, written language is highly developed. Aided by the proper uses of the wedge and soft clay, the reed pen and papyrus, the chisel and stone, written language passed through the following main stages:

a) Pictorial, in which a picture tells the story

b) Ideographic, in which a picture becomes symbolic or represents a thought

c) Phonetic, in which the picture becomes a phonogram or sound representing sign. This stage involved three additional steps:

   1) Signs that stood for words

   2) Signs that stood for syllables

   3) Signs that stood for sounds

Our modern alphabet can be traced back from the Latin to the Greek (the word 'alphabet' being derived from the first two letters of the Greek alphabet, alpha and beta) to the Phoenician alphabets. During the Renaissance, a lot of people devoted their attention to type design. Letters had been defined mathematically and had been constructed by the use of compasses and rulers. In [And71, Mei69] a summary of these developments is given. It wasn't until the introduction of the movable metal type (by J. Gutenberg [Hay30]) that any standard font was achieved. Despite the slight changes that have been made, since then, the basic composition of the alphabet remains the same.

The invention of writing as a means of preserving knowledge on one hand, and the invention of printing to multiply that written knowledge on the other hand, are among the most momentous triumphs of the human mind. They provided a potent weapon for attacking ignorance and

superstition, and made literacy and universal education a practical and economic proposition. Knowledge becomes available to everyone, and as a result the progress of the human race can evolve even better.

What is written (encoding process) is to be read (decoding process) through the eyes. Hence, there exists a communication channel between the writer (sender) and the reader (receiver). For this communication to be effective, it is necessary to discover the reader's true requirements (psycology). Good-looking letter shapes (calligraphy) are to fulfil the aesthetic requirements. The understanding of how the human eye and brain function in reading, will contribute to greater reading efficiency. Furthermore, characters as visual images convey concepts or ideas such as, freedom, smartness, strength, emphasis, sophistication, style, modernity, etc. (typography).

The shape of the letterforms had been influenced by the materials (i.e. clay, stone, wood, paper,etc.) and tools (i.e. wedge, chisel, pen, brush, etc.) used as well as by the printing techniques and methods [Wal69]. It also bears the different periods of typographic design. The development of the letterform design can be marked by the following eras:

a) Scribal (or ductal) era. Ductal letters are handwritten

letters whose basic topology is the result of a smooth series of movements of the writing tool in the plane of the writing surface

b) Glyptal era. Evolution of glyptal letters reflects the fundamental changes in the technology of letter production brought about by the invention of printing from movable type. Each letterform was engraved in relief onto the face of a steel punch; the composition is then inked and printed on paper

c) Digital era. Digital letterforms are made up of digital elements, i.e. pixels, line segments, color, shades of gray or any other graphics unit. The digital type design is discussed in chapter 2.

As a recapitulation of this section the following may be concluded:

a) Written language was the result of the need of communication (not restricted by time or space) among humans (visual communication)

b) It was through a series of very natural steps that writing grew out of drawing (picture-writing)

c) Once the existence of written language was firmly established, people tried to beautify it and express concepts or ideas (caligraphy, typography)

d) The impact of writing on a society produced changes in the social structure of that system (sociology,

education).

e) The writing materials and tools used as well as the printing methods and techniques reflected the letterform design (technology).

Chapter 2

Computer-Aided Character Design

2.1 Notions on character generation

Among the many computer peripheral devices, printing devices play a major role in man-machine communications. Giving the computer the ability to "write" brings automated character generation.

L. C. Hohenstein [Hoh80, pp. 25-57] has classified printers into three broad categories, one of which is: fully formed versus dot-matrix characters. In fully formed printers all parts of a character are embossed in reverse on the type bars of the typewriter. Digital evolution leads to the character representation in the form of binary (black and white) matrix. New printing devices based upon ink jet, laser, and other technologies generate characters in the form of dot-matrices, thus offering considerable flexibility in the style and form of the character fonts.

P. Coueignoux [Cou81] has stated that a character generation system can be characterized by five criteria:

a) Speed. It refers to: i) the speed of creation which may range from a few minutes to a few hours, and ii) the speed of production which goes beyond 1000 characters/sec depending on the type size and device resolution.

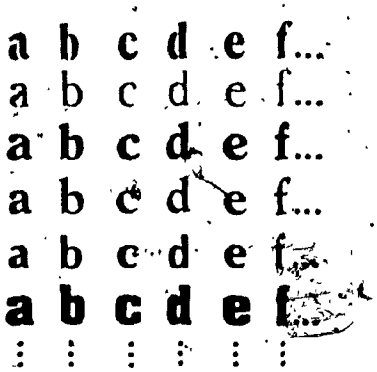
b) Space. It refers to the average size of the code for one character as well as to the size of the internal buffers used for decoding.

c) Quality. It is proportional to the largest dot matrix which can be used to represent a character.

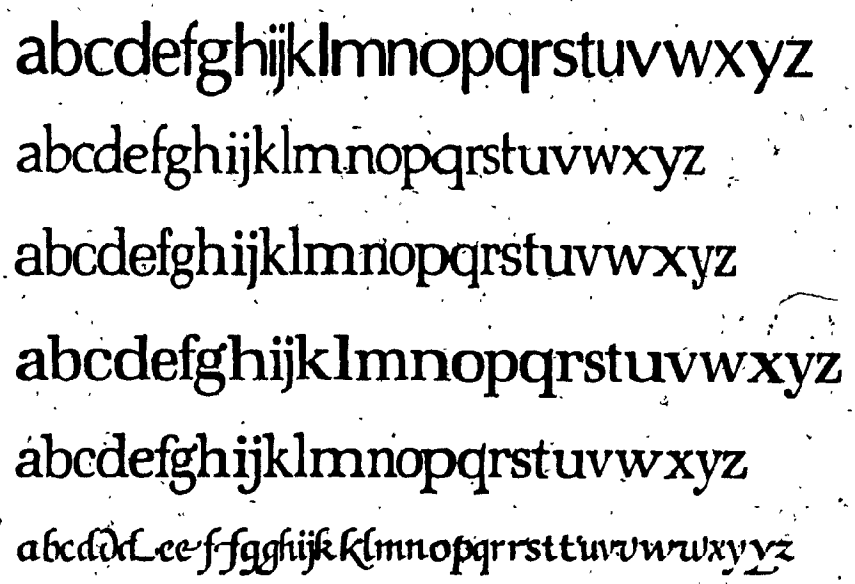
d) Flexibility. It refers to the different automatic modifications which are supported by the code, i.e. scaling, rotating, family font variations, etc.

e) Cost

An important aspect among character fonts is their "consistency" [Cou75, Cox74, Hof82]. This means: i) consistency among characters sharing the same letter, and ii) consistency among all characters sharing the same "spirit(s)" (artistic or stylistic consistency). Notice that a character bears the name of the letter as well as its style, i.e. artistic embellishments [Cou75, Cox82] (see also section 2.2). For example, looking at Figure 2.1 (a) it can be asked: "what do all characters in any column have in common ?" or "what do all characters in any row have in common ?". Looking at Figure 2.1 (b) the question to be asked may be: "what do all these typefaces have in common ?". Similar questions may be asked about type faces



(a)



(b)

Figure 2.1 Consistency among characters. (a) The vertical and horizontal problems. (b) A few typefaces designed by Hermann Zapf [Hof82].

corresponding to distinguishable periods of typographic design. What is important to note here is not only to find answers to these questions, but what questions should be asked in order to discover more about the mystery of the letterforms.

We are now in the computer era. The evolution of the computer introduces the digital type design [Big83]. Efforts should be made to consider the characteristics (both advantages and limitations) of the new display devices. This new technological evolution can by confronted by the following two phases:

a) Imitation, in which the outstanding letterforms of the previous typographic generations serve as models for the new designs

b) Innovation, in which innovative designs emerge that are not merely imitative but exploit the strengths and explore the limitations of the medium.

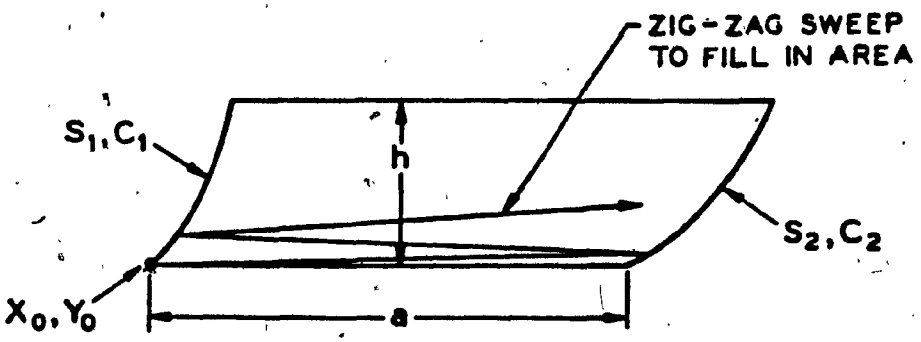
Since type is ultimately intended for the reader, the technology of type production however is not the only factor that influences the final letterform. Typographic design remains an art, and the successful design subtly reflects the tension between imitation and innovation. With the development of the computer and digital electronics, new specialized machines can be built for the use of graphic

arts [Nis83].

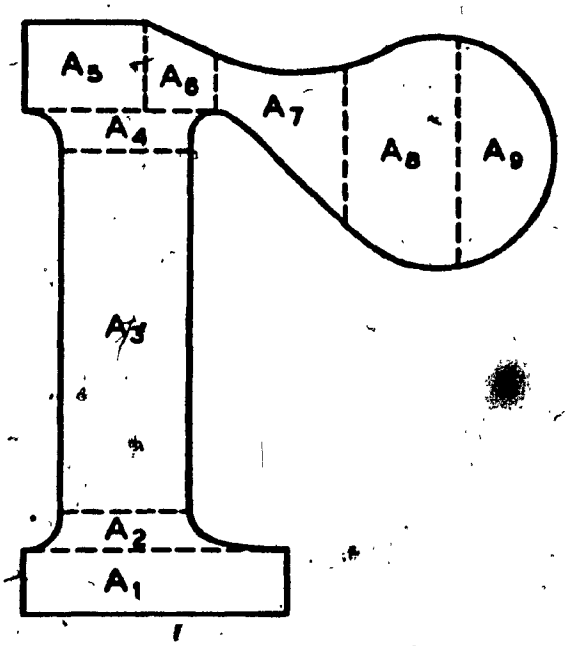
In the subsequent section several approaches used in the past will be reviewed in which the computer provides great assistance to the design of the letterforms. Afterwards, the basic structure of the proposed system will be described.

2.2 Review of related work

In 1965 M. V. Mathews and J. E. Miller [Mat65] developed a Computer Editing, Typesetting and Image Generation System. Images were constructed as the sum of a number of PATCH's (Parameterized Area To Construct Holograph). Three types of patches (rectangle, trapezoid, and curved) were considered. A patch required eight parameters: width (a), height (h), the coordinates of one corner (X_0,Y_0), and the curvature and slope of each curved side (C_1,S_1,C_2,S_2). Figure 2.2 shows a Patch with its eight parameters (a) as well as the formation of the lower case "r" by Patches (b). Rectangles and trapezoids were treated as special cases. The concept of stylistic consistency was introduced, i.e. sub-areas possessed by several letters. The division of an image into patches and the extraction of the parameters (by trial and error) were done manually, thus



(a)

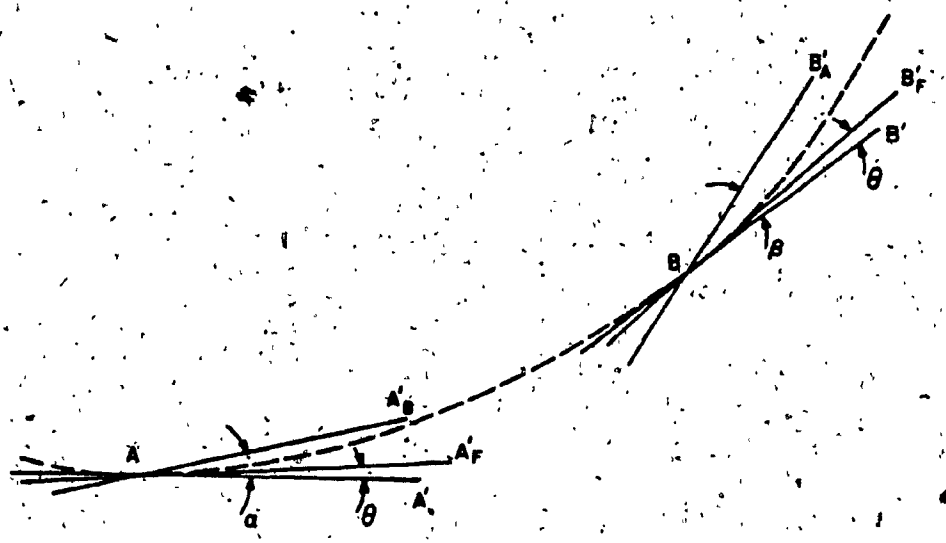


(b)

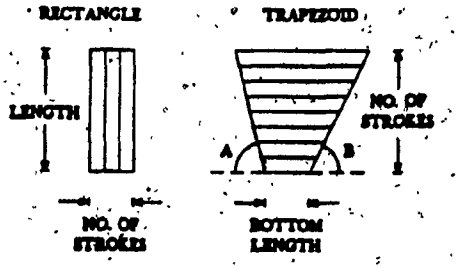
Figure 2.2 Image generation using the Patch method [Mat65]. (a) A Patch and its eight parameters. (b) Formation of lower case "r" by Patches.

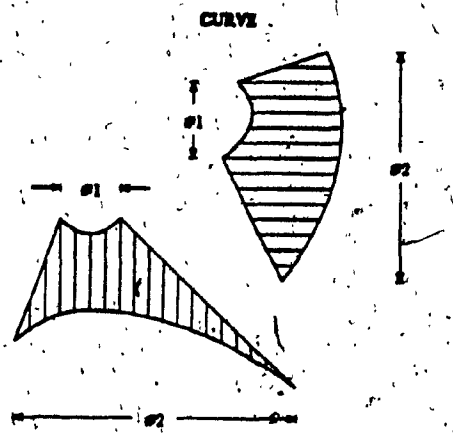
considerable time and effort was required. Furthermore, different sizes of an image required new definitions. On the average, 10 patches were required for each 'good' quality letter, thus a letter required about 80 (= 10x8) numbers for its description.

A. J. Frank [Fra71] continued the method proposed by M. V. Mathews and J. E. Miller [Mat65]. Again the three types of patches were used (see Figure 2.3 (b)). However, the two straight lines in curved patches could not be parallel. In general, she improved the patch method by: a) automating to a large extent the encoding part, b) approximating the contours with parabolas whose principal axes were either parallel to the X- or Y-axis, and c) generating different sizes of the same image definition. The first task in curve fitting was to determine the points of tangency with 0, 45, 90, and 135 degree lines. Then between successive points a parabola was fitted by "equalizing the angle error" method. This method is illustrated in Figure 2.3 (a) and can be stated as follows: Given points A and B, and their derivatives A' and B', choose that parabola such that the angle made by the tangent lines corresponding to the derivatives A' and A_F' is equal to the angle made by the tangent lines corresponding to the derivatives B' and B_F' (where A_F' and B_F' are the derivatives of a family of parabolas which pass through A and B). The coding size was about 25 words per character



(a)





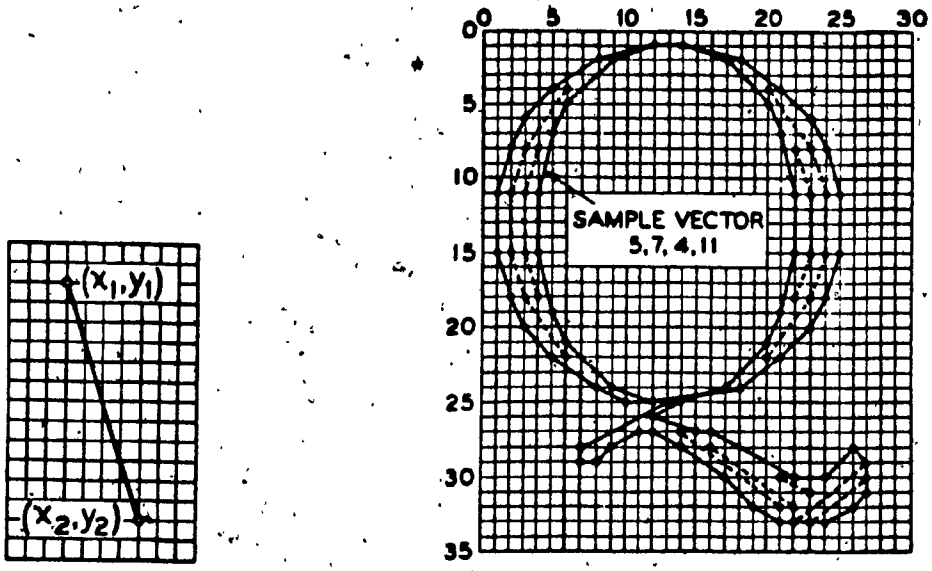
(b)

Figure 2.3 The improved Patch method [Fra71]. (a) Equalizing the angle error. (b) The three types of Patches.

and the printing speed about 100 characters per second.

Another attempt to encode black and white pictures was made by M. V. Mathews et al [Mat67]. Letters were formed by a number of vectors (see Figure 2.4 (b)). A vector could be drawn between any two points (see Figure 2.4 (a)). The process of determining the coordinates of the vectors was as follows: An enlargement of a letter was placed on a suitably-scaled raster, and the appropriate vectors were chosen visually. All measurements were in units of raster spaces. Line widths were determined by the number of vectors, i.e. a double vector would produce two shades in the Cathode Ray Tube. The resolution was limited by the width of the vectors (ten of them) which was about 2.3 raster points (on the average). An average of 25 vectors per character was used. Hence, the code size was about 25x4x10 = 1000 bits. A great disadvantage of this method is that different fonts or even different sizes of the same font required a separate and distinct set of vectors. Thus, the manual operations had to be repeated and the storage required to hold the definitions increased proportionately for larger-sized characters, and for more complex fonts.

A very interesting piece of work was done by H. W. Mergler and P. M. Vargo [Mer68]. They designed a computer system which is considered as the first attempt for parametric type design. The system consisted of 24 routines



(a) (b)

Figure 2.4 Image generation using vectors [Mat67]. (a) A vector drawn between (X_1, Y_1) and (X_2, Y_2). (b) Formation of upper case "Q" by vectors.

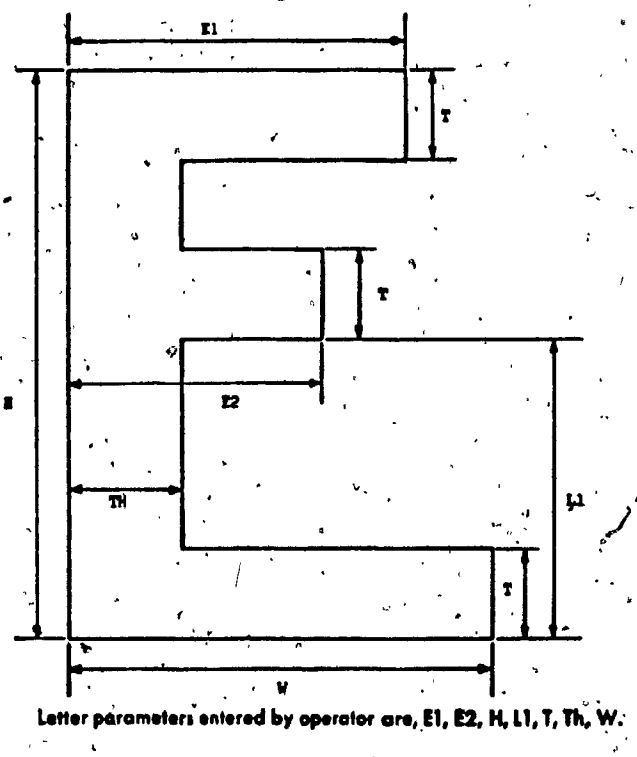
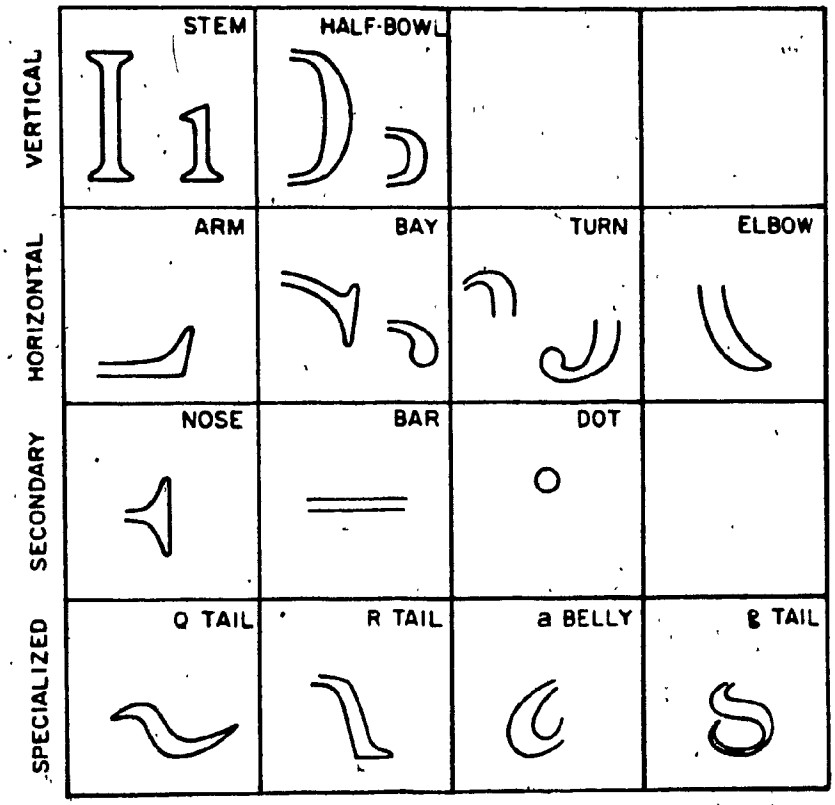


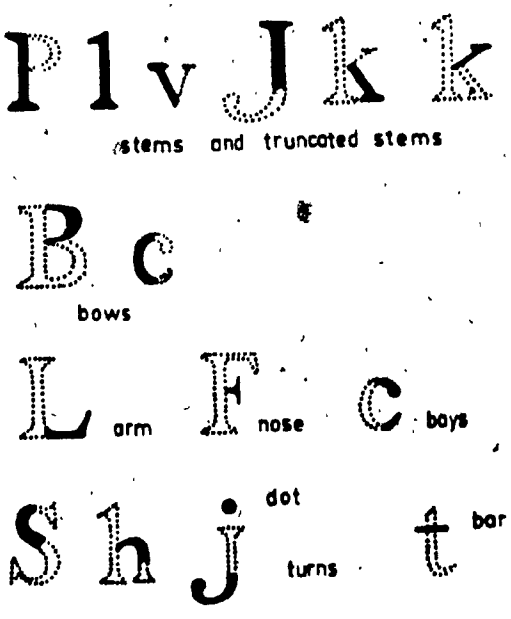
Figure 2.5 Parameter definitions for upper case "E" [Mer68].

for the generation of 24 capital letters (Q and J were omitted because they look so similar to O and I). It ran under two modes: In the manual mode the operator was able to select various routines for execution, define their parameters, or determine the states of the system. When running was under the control of the "Automatic Executive System", parameters for the 23 other letters were calculated based on values input by the operator for the letter E. Figure 2.5 shows the parameter definitions for the upper case "E". In general, the approach was based on the edge generation with a limited number of strokes, i.e. straight lines, curved strokes (made out of superellipses). Despite the fact that the model for certain letters was oversimplified, the generation of serifs was rather primitive, and some details were omitted, nevertheless their attempt made a positive contribution to the subsequent developments.

In 1975 P. Coueignoux [Cou75, see also Cou81 and Cox76] developed a system for character generation (CSD, Character Simulated Design) which in many aspects is a continuation of the work done by M. W. Mergler and P. M. Vargo [Mer68]. It consisted of 44 routines for the 52 upper and lower case characters in the Roman font. Characters were composed from primitives. Three types of curves were used to generate the primitives: conic sections, straight lines, and super-ellipses. There were 13 different types of



(a)



(b)

Figure 2.6 J. M. Coueignoux's model for character generation [Cou75]. (a) The list of primitives. (b) Examples of primitives in context.

primitives and on the average ten parameters per primitive were needed. The list of primitives for the 52 upper and lower case characters is presented in Figure 2.6 (a), and illustrated in Figure 2.6 (b). Input to the system consists of the following: the letter name, a list of parameters for the primitives, and a list of parameters special to the letter. There were, on the average 27 parameters per character. At the font level, however, this number was reduced to 10. Four type faces were chosen (Baskerville, Bodoni, Cheltenham Medium, and Times Roman Bold), whose parameters were found manually and inaccuracies were adjusted by an interactive process. The average generation time was about 30 seconds per character.

Another interesting study was also done by C. H. Cox et al [Cox82, see also Cou75, Cox74, Cox76]. They tried to find a description that separates aspects of letterness from aspects of embellishments among the characters. In attempting to separate these two aspects, it was found convenient to introduce a formal structure, called skeleton. The three formal structures were characterized as follows: a) Letter, i.e. abstract description in terms of functional attributes. It distinguishes letters from letters, b) Skeletons, i.e. semi-abstract, semi-physical description in terms of vertices, spatial ordering, and vertex/edge relations. It distinguishes letters from non-letters, and c) Character, i.e.physical description in terms of strokes

and serifs. It distinguishes styles of letters. Then the process of generating a character can be obtained by the following steps: a) The abstract description of a letter is obtained, i.e. a set of functional attributes, b) Representation of sets of skeletons which combine the parts from (a) in a proper or improper manner is done, c) A skeleton is obtained from (b) by embodying some artistic constraints, and d) Physical character is obtained by applying some "redrawing" rules together with secondary embellishments to the skeleton from step (c).

Another piece of work that deserves appreciation was done by D. E. Knuth [Knu79, Knu80, Knu82]. He developed the METAFONT, a system for alphabet design. The system allows an entire font to be precisely specified mathematically so that it can be produced in different sizes and styles for different raster devices. A METAFONT user can either write a METAFONT routine or he/she can interactively define a shape and obtain the desired shape by trial and error. The shape is defined by a declarative algebraic language and is drawn by the motion of the pen's center. Characters are generated by cubic splines with complex coefficients or other simple shapes. The system mainly consists of 128 routines for ten fonts which belong to the Roman family. A variety of related fonts can be obtained by setting a number of parameters. There are totally 28 such parameters (for the Computer Modern Roman)

that can be categorized into the following main groups: a) parameters controlling the vertical dimensions, i.e. h-height, x-height, e-height, descender depth, etc., b) parameters governing the horizontal dimensions, i.e. unit width, the amount by which serifs of lower-case (or upper-case) letters project from the stems, etc., c) parameters affecting the pen sizes, i.e hairline width, stem width, curve width, etc., and d) miscellaneous parameters controlling special effects such as slanting, ellipticity of bowls, etc.

D. E. Knuth's article [Knu82] has received a variety of controversial comments [Bau82, Hof82] which are worthwhile noticing. Such comments emerged from the relationship between artistic design and mechanizability (by the computer). Some people saw METAFONT as a threat that would limit the designers' creativity, deprive them of jobs in coming years, and pointed out its limitations. Others saw it as a tool that would assist type designers, and yield new insights so that we can learn more about the mystery of the letters and raise the artistic level higher.

T. Y. Mei [Mei81] developed the LCCD, a Language for Chinese Character Design. The LCCD is based on METAFONT and has to take into account the particular problems related to Chinese characters, such as the number of strokes and shape complexity. The approach taken was a combination of

"top-down" and "bottom-up" procedures. In the "top-down" procedure, analysis of the structure of the characters was done and a sequence of sets was obtained. Each set contained sub-characters or radicals (which are of higher order), and basic strokes or atoms (which are of lower order). The structure becomes more complex when the order gets higher. In the "bottom-up" procedure, the generation of characters was performed. A radical set having the lower order was built first, and the higher order ones built later. The first set to be built contained the basic strokes.

J. Hobby and G. Gu [Hob82] tried also to use METAFONT to design Chinese characters. T. Y. Mei [Mei81] developed 108 routines. On the contrary, J. Hobby and G. Gu [Hob82] implemented a number of stroke routines named horizontal, vertical, pie, D-stroke, F-stroke, and vertical stroke with "na-ends" (10 strokes routines for the Mincho style). These routines were parameterized fully enough so that one routine could be used to draw all stroke variations. It is evident that their contribution was the reduction of the stroke routines by parameterization.

Several suggestions have been proposed for the design of character fonts using the curve-fitting method. T. Pavlidis [Pav82, pp.267-269] suggested the use of the quadratic splines with uniform knot distribution to obtain

the outline of the letters. The same author [Pav83] also proposed the "curve fitting with conic splines" method. In his suggestion, characters may be drawn by: a) obtaining the font definition by finding the approximate polygon, and b) applying the curve fitting to the definition obtained in (a). A similar approach has been suggested by M. Plass [Pla83]. He used piecewise parametric cubic polynomial approximations to obtain the outline of the character fonts. The curve-fitting methods automate to a large extent the encoding process and produce resolution-independent font representations. However, the code representation, the computational expense and the lack of stylistic consistency are among the other drawbacks inherent to these approaches.

2.3 Basic structure of the proposed system

The proposed Character Generation System for this thesis is shown diagrammatically in Figure 2.7, and it consists of mainly two processes:

a) The Encoding or Description process (Analysis), and

b) The Decoding or Generation process (Synthesis).

The Encoding or Description process is performed by the Picture Descriptor (P.D.). Its function is to transform input digitized character representations in the form of

binary (black and white) matrices into graph representations in the form of edges and vertices. Furthermore, code derivation and modification (to adjust possible inaccuracies) may be performed by: i) an Interactive Graphics Editor (I.G.E.), and ii) manually. This can be easily accomplished since the code inherits the basic structure of a character.

The output of the Description process is held in a Data Base (D.B.).

The Decoding or Generation process is performed by the Character Generator (C.G.). Its function is to accept the code from the Data Base and generate different character fonts. This is accomplished by considering certain 'elementary' operations or rules depending on the specific type style. Family variations in size, boldening and orientation are also taken into consideration.

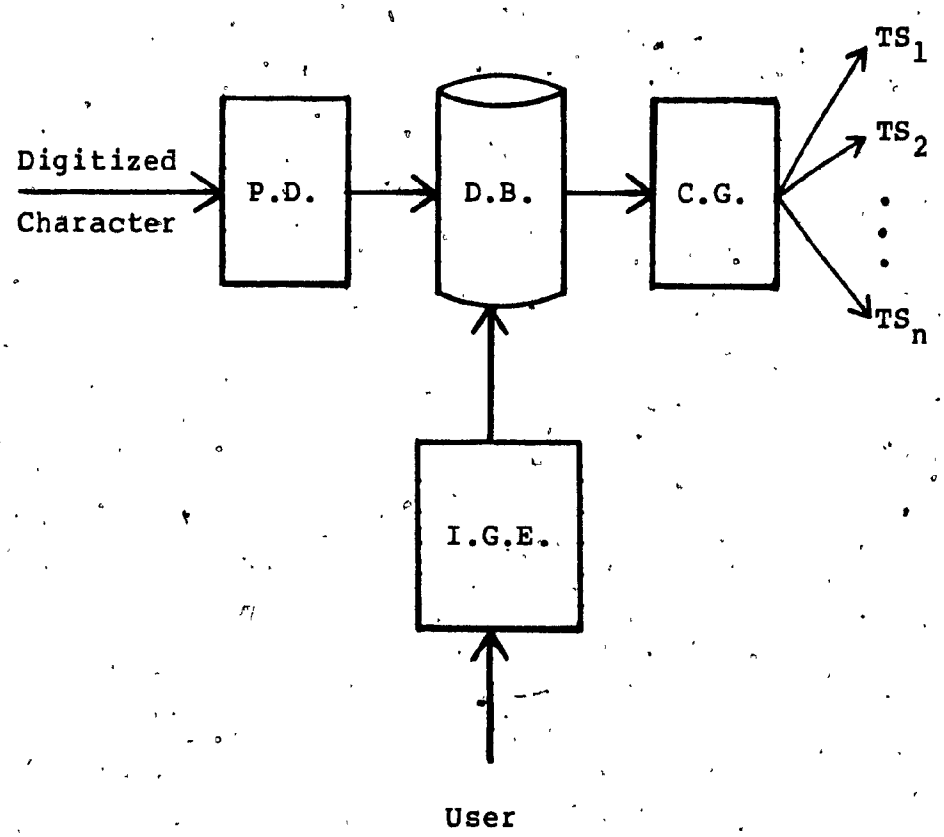


Figure 2.7 Basic structure of the proposed system.
TS_i stands for the i-th Type Style generated.

Chapter 3

The Picture Descriptor

3.1 Introduction

A pattern may be transformed into an abstract description, i.e. a set of special points or numbers, a string of symbols, graph, etc. This description can be used for data compression, image transmission, pattern recognition, medical diagnosis, etc.

Many investigators have devoted attention to this challenging topic and a variety of approaches have been proposed. H. Amiri et al [Ami79] used fill-up, bends, end and cross points for line-drawing descriptions. M. Beun [Beu73] used end and fork points for numerical classification. P. Chinnuswamy [Chi80] described the Tamil characters in terms of line-like elements. T. CH. M. Rao [Rao76b] used forks (downwards, upwards, open rightwards, open leftwards) for fingerprint classification. K. C. V. Rao and K. Black [Rao76a] converted the fingerprints into a set of symbols. K. J. Udupa [Udu75] transformed a line pattern into a sequence of turning and end points.

P. Coueignoux [Cou75, Cou81] reviewed the coding schemes dealing with type faces. He classified them into three main classes: a) Area coding (bit map, and run length coding), b) Contour coding (differential run length, chain link, spline encoding, and patch coding), and c) Structural coding (structural coding with contour, and structural coding with skeletons). Area coding is simple, best for high speed and low quality printing, but it offers limited flexibility. Contour coding is derived from the fact that area is just what is enclosed in a contour. This scheme behaves better when high quality is accommodated by a reasonable amount of space. Scaling is well implemented, but it can hardly be said to provide flexibility. Structural coding works at a more global level. A character is described by a set of "primitives" (stem, bow, arm, bay, turn, etc.) that respect certain rules (see previous chapter). This method is of great interest because of the provided flexibility.

In this chapter, the encoding or description process is presented. Its purpose is to derive a code that will: a) reduce the storage requirements, b) inherit the basic structure of the characters and hence be easily understandable by the user, and c) facilitate the generation of different type styles. The code is represented in the form of a graph (edges and vertices). This is achieved by mainly developing the thinning, smoothing, point labelling,

and graph traversal algorithms. First, the description of a simple algorithm will be given. Such algorithm is used for pictorial information extraction, and will facilitate the description of the subsequent algorithms.

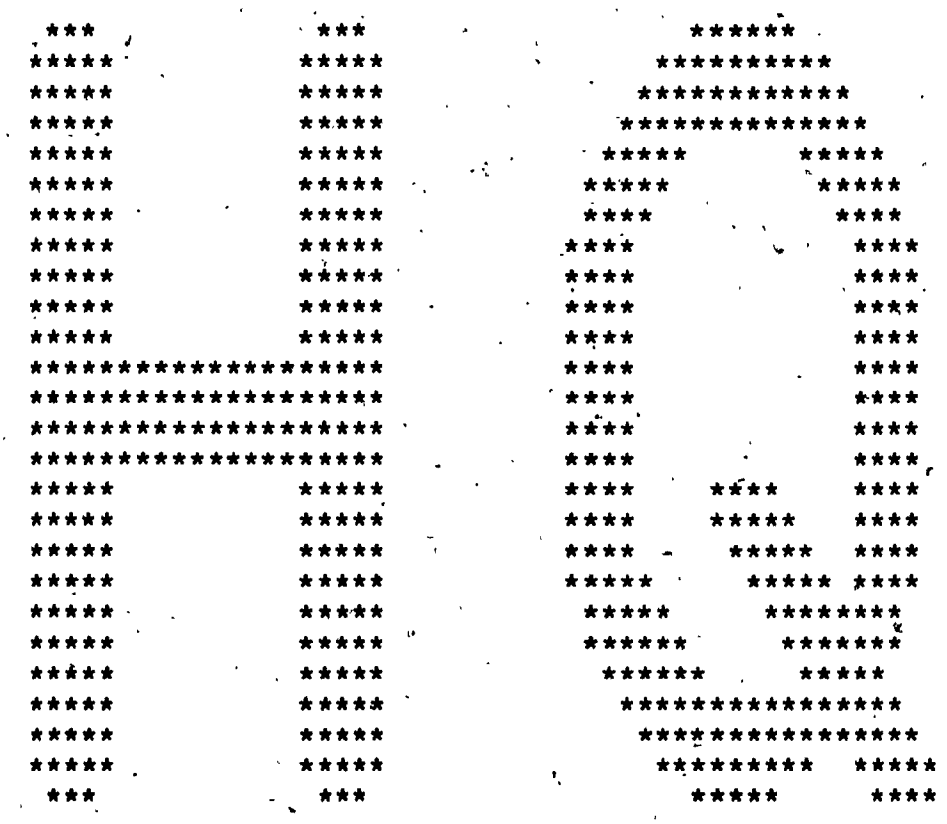
3.2 Input data / Definitions

Input characters are scanned by a high-speed optical scanner (ECRM Autoreader) and are transformed into binary (black and white) matrices. Figure 3.1 shows some sample input characters.

The pattern is stored in a two-dimensional array, M, with TOP $\leq$ i $\leq$ BOTTOM rows and LEFT $\leq$ j $\leq$ RIGHT columns.

The (i,j)-th element of the array may be either black or white. Figure 3.2 shows a point P1 along with its eight neighbors. The points P2, P4, P6, P8 form the set of the orthogonal elements of P1.

Let S be the entire set of elements in the pattern, and $S_i \subseteq S$, i = 1,2,...,n. S_i is called the observation window and it is a rectangle with defined center (i,j), height and width.



(a) Upper-case "H"

(b) Upper-case "Q"

(c) Lower-case "u"

(d) Lower-case "x"

Figure 3.1 Digitized character samples.

P5(i-1,j-1)	P4(i-1,j)	P3(i-1,j+1)
P6(i,j-1)	P1(i,j)	P2(i,j+1)
P7(i+1,j-1)	P8(i+1,j)	P9(i+1,j+1)

Figure 3.2 A point PI and its eight neighbors.

3.3 The algorithm for pictorial information extraction

Given an observation window and a gray-level Algorithm 3.1 (shown below) finds the black/white areas within the window. Notice that since we are dealing with binary patterns, the gray-level of an element is either black or white, and its value, denoted by "GrayLevel", is either 1 or 0, respectively.

The algorithm starts by scanning each element within the window and checking for a point having a value "GrayLevel". Upon locating such a point, it calls routine FINDGROUP(i, j, NP) to find the group NP of points with "GrayLevel" values, i.e. each point in the group has the same value and equal to "GrayLevel". It does this by recursively considering the orthogonal neighbouring

elements, including and starting with point (i,j). The whole process continues by detecting the next group, if any, and stops when all groups with "GrayLevel" points have been found.

Algorithm 3.1 Extraction of white/black areas

Calling: Call Window(x,y;height,width,GrayLevel)

Input: (a) Observation window, i.e. Center(x,y), height, width, and (b) The value "GrayLevel"

Output: (a) NG, number of groups with value "GrayLevel", and (b) GR, an array, with the i-th entry containing the number of points in the i-th group.

Steps

1. Initialize
   1.1 NG = 0
   1.2 $GR_i = 0$; i = 1, 2, ...
2. For each Point (i,j), within the window, Do Step 3
3. If Point (i,j) = "GrayLevel" then
   3.1 NG = NG + 1
   3.2 Call FINDGROUP(i,j,NP)
   3.3 GR_{NG} = NP

As an illustration consider Figure 3.4 (a). The 5x5 window in (i) has two white groups: one at the top-left with 17 elements; and the other at the bottom-right with 4 elements. The 5x5 window in (ii) has three white groups: one at the top with 11 elements; another at the bottom-left

with 4 elements; and another at the bottom-right with 4 elements.

3.4 Thinning / Smoothing

The digitized characters, obtained by the scanner, are thinned by applying the thinning algorithm proposed by A. Bel-lan and L. Montoto [Bel81]. Prior to that, some simple smoothing operations, such as those suggested by S. H. Unger [Ung59], were done to remove isolated points and fill single point gaps.

The thinning algorithm consists of an iterative process. At each iteration step some black points are deleted and replaced by white ones. The iterative process is executed until the thinned version (representing the skeleton) of the pattern is obtained. At a given iteration the pattern is scanned four times. The four scanning sequences (see step 2.4) are: a) top to bottom and left to right, b) top to bottom and right to left, c) left to right and top to bottom, and d) left to right and bottom to top. At each scan, points are tested for Interior or Boundary elements. Interior points are set to the current iteration number plus one. Boundary points are deleted unless they belong to the Medial line. More details about the

scannings, and the definition of the Boundary, Interior and points on the Medial line can be found in reference [Bel81].

Algorithm 3.2 A. Bel-lan and L. Montoto's thinning algorithm [Bel81]

Notation: M is an array to hold the pattern and M' holds the resulting pattern after each iteration.

Steps

1. Initialize
   1.1 Get the digitized character M'
   1.2 Iteration = 0
   1.3 Thinned = False
2. Repeat Step 2 Until Thinned = True
   2.1 M = M'
   2.2 Iteration = Iteration + 1
   2.3 Thinned = True
   2.4 Scan the image four times and
       at each scan Do step 3
3. For each Point (i,j), Do
   3.1 $M'_{i,j} = M_{i,j}$
   3.2 If Point (i,j) is an Interior point Then
       $M'_{i,j}$ = Iteration + 1
   3.3 Else if Point (i,j) is a Boundary point and
       it is not in the Medial line Then
       $M'_{i,j} = 0$
       Thinned = False

Figure 3.3 shows some of the results of thinning applied to Figure 3.1.

The resulting thinned character undergoes a smoothing process which consists of two operations:

- The filling operation to fill in sharp corner points
- The deleting operation to eliminate small bumps

Every point in the input matrix is scanned (from top to bottom - left to right). Each time, a 5x5 or 3x3 observation window (see algorithm below) is checked to determine whether its central point will be changed (from white to black - filling operation, or from black to white - deleting operation) or maintained. The two operations are done simultaneously in one scan. Figure 3.4 shows the matrix configurations used for the smoothing process (including their counter-clockwise rotations by 90, 180, or 270 degrees).

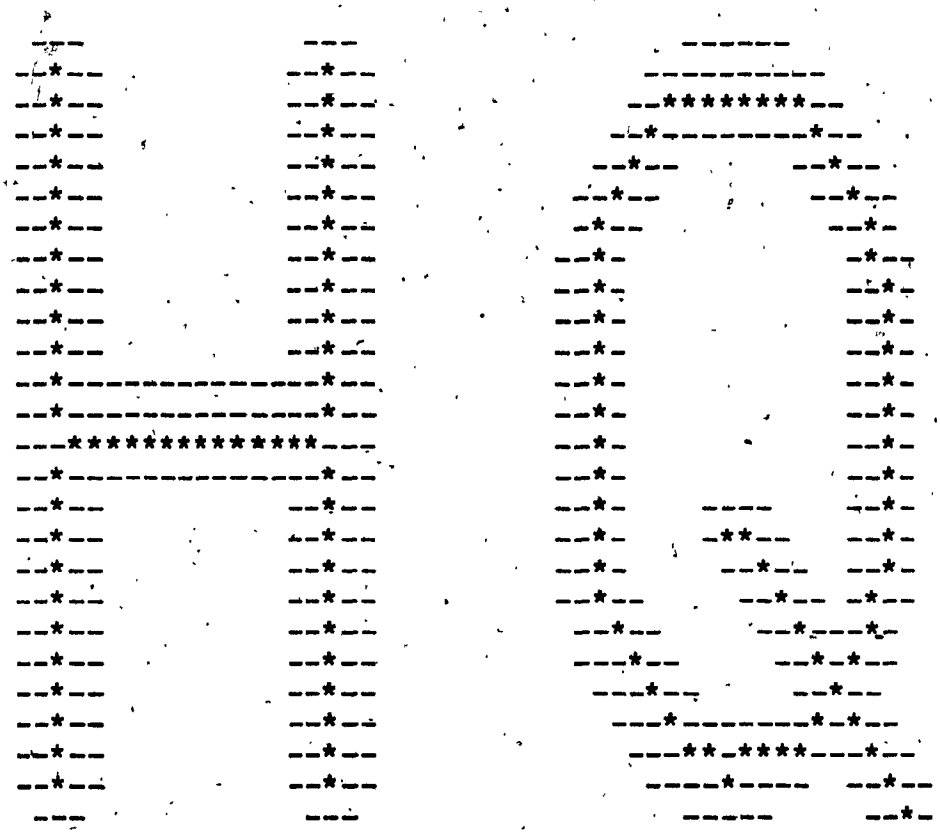
Algorithm 3.3 Smoothing operations

Note: See section 3.3 for the description of procedure WINDOW, NG and GR.

a) Filling operation

Steps

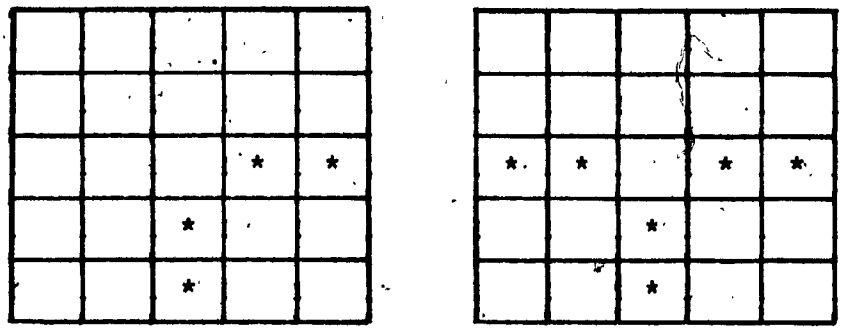
1. For every Point (i,j), Do Steps 2-4
2. Call WINDOW(i,j,5,5,white)



(a) Upper-case "H" (b) Upper-case "Q"

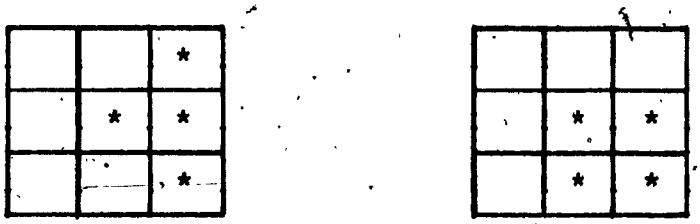
(c) Lower-case "u" (d) Lower-case "x"

Figure 3.3 Results of thinning. The *'s show the skeleton; the -'s are the deleted points.



(i) (ii)

(a) 5x5 matrix configurations for filling operations.



(i) (ii)

(b) 3x3 matrix configurations for deleting operations.

Figure 3.4 Smoothing operations. The *'s and blanks represent the black and white points, respectively.

3. If NG = 3 and $GR_k = GR_1$ and $GR_m = 11$ Then

   { k,l,m can be either 1 or 2 or 3 }

   Set Point (i,j) = Black

4. If NG = 2 and $GR_k = 17$ and $GR_1 = 4$ Then

   { k,l can be either 1 or 2 }

   4.1 Call WINDOW(i,j,3,3,white)

4.2 If NG = 2 Then

Set Point (i,j) = Black

b) Deleting operation

Steps

1. For every black Point (i,j), Do Steps 2-3

2. Call WINDOW(i,j,3,3,white)

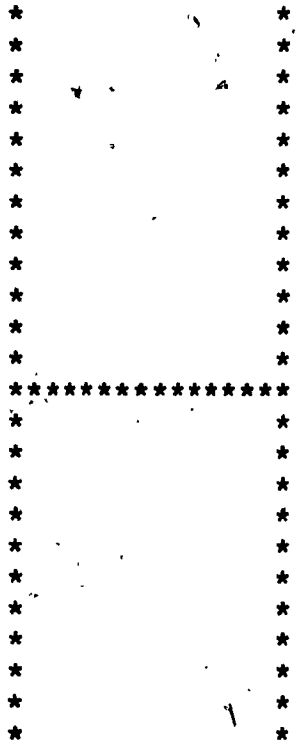
3. If NG = 1 and GR_1 = 5 Then

Set Point (i,j) = White

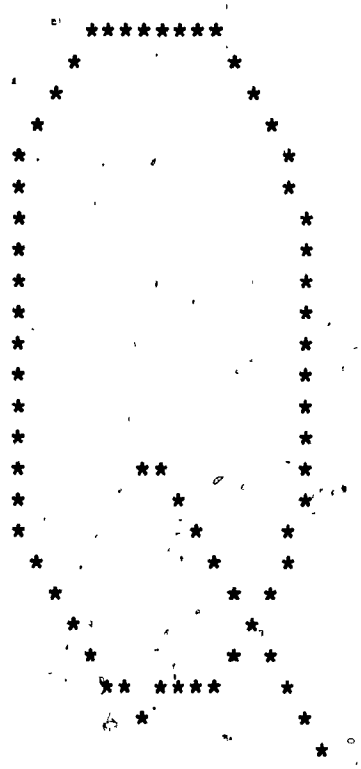
Figure 3.5 illustrates the effects of the smoothing operations.

3.5 Point Labelling and Reduction

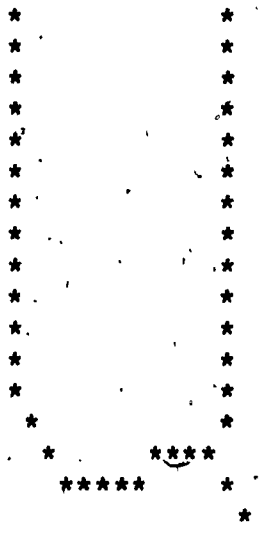
Each point of the thinned version of a character is classified as a cross-point (C), bend-point (B), end-point (E), or Interior-point (I) (see Algorithm 3.4 below). For each point, the number of the white groups, NG, and the number of the white points in each group, GR_i, (i being the i-th group) occurring in the 8-neighborhood, is found. Depending on these two numbers a point is classified accordingly, i.e. an end-point (E) has only one group of white points, etc.



(a) Upper-case "H"



(b) Upper-case "Q"



(c) Lower-case "u"

(d) Lower-case "x"

Figure 3.5 Effects of Smoothing.

Algorithm 3.4 Point Labelling

Note: See section 3.3 for procedure WINDOW, NG, and GR

Steps

1. For every Point (i,j), Do Steps 2-8
2. Call WINDOW(i,j,3,3,white)
3. If NG = 1 Then
   Set Point (i,j) = 'E'
4. Else if NG = 3 Then
   Set Point (i,j) = 'C'
5. Else if NG = 2 and $GR_1 = GR_2 = 3$ Then
   Set Point (i,j) = 'I'
6. Else if NG = 2 and $GR_k = 5$ Then
   { k can be either 1 or 2 }
   Set Point (i,j) = 'C'
7. Else if NG = 2 and $GR_k > 2$ Then
   { k can be either 1 or 2 }
   Set Point (i,j) = 'B'
8. Else Set Point (i,j) = 'C'

During the labelling process as well as after its completion, a reduction process is performed to reduce the number of points labelled as cross- or bend-points.

Algorithm 3.5 Point Reduction

Steps

1. For every Point (i,j), Do Steps 2-4
2. For every 8-neighbouring Point (x,y), Do Steps 3-4

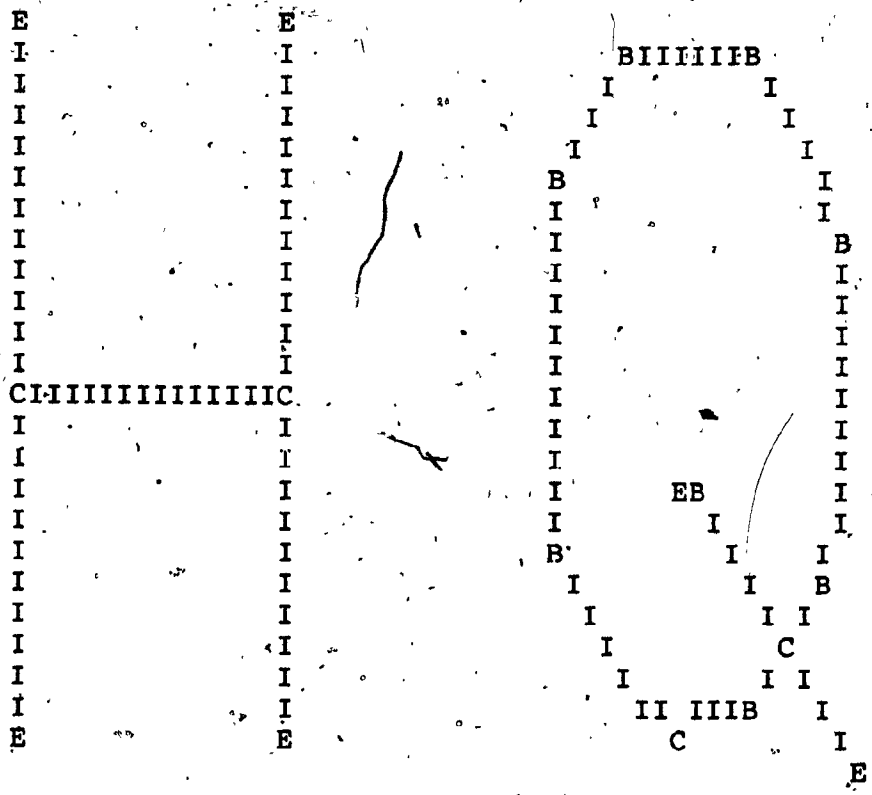
```
3. If Point (i,j) = 'C' and if
      Point (x,y) = 'C' or 'B' Then
   3.1 Call WINDOW(x,y,3,3,white)
   3.2 If NG < 2 Then
         Set Point (x,y) = 'I'
4. Else if Point (i,j) = 'B' and
         Point (x,y) = 'B' Then
         Set Point (x,y) = 'I'
```

The results of the processes described in this section are shown in Figure 3.6.

3.6 Graph Traversal

The skeleton (with labelled points) of a character is traversed and a graph associated with it is generated. Figure 3.6 illustrates the labelling of points in the skeletons of several characters.

A graph consists of a set of points, node-points, connected by lines, called branches. A branch can be: a) A straight stroke which is a single straight line drawn between the two node-points, or b) A curved stroke which is a sequence of straight lines drawn between two branch-points, or a node-point and a branch-point. A



(a) Upper-case "H"

(b) Upper-case "Q"

(c) Lower-case "u"

(d) Lower-case "x"

Figure 3.6 Point Labelling and Reduction

node-point is: a) an end-point (E), or b) a cross-point (C) that its eight neighbors form at least three groups of white points, i.e. NG > 2 (see section 3.3). A branch-point is either: a) a bend-point (B), or b) a cross-point (C) that is not a node-point. Notice that by changing the definition of the node-points the resulting graph representation of a letter may be changed.

The basic graph traversal algorithm was taken from [Pav82, pp. 100-102] and is listed as follows.

Algorithm 3.6 Graph Traversal

Notation: Procedure ADJACENT(p,n,N) receives point p and returns the number of adjacent node-points, n, stored in the array N. Functions POP(p,S) and PUSH(p,S) are used to pop or push point p from/to the stack S, respectively.

Steps

1. Find a cross- or end- or bend-point, p
2. Place p on the stack
3. While S is not empty, Do Steps 4-11
4. Call POP(p, S)
5. Repeat Steps 6-11
6. Mark p
7. Call ADJACENT(p, n, N)
8. If n = 0 Then exit from the loop
9. Set p = N_1
10. If n > 1 Then

11. Call PUSH(N_i, S), i = n, n-1, ..., 3, 2

The graph traversal algorithm is repeated for all connected components of a character (i.e. i or j has two connected components). Single points can be easily detected.

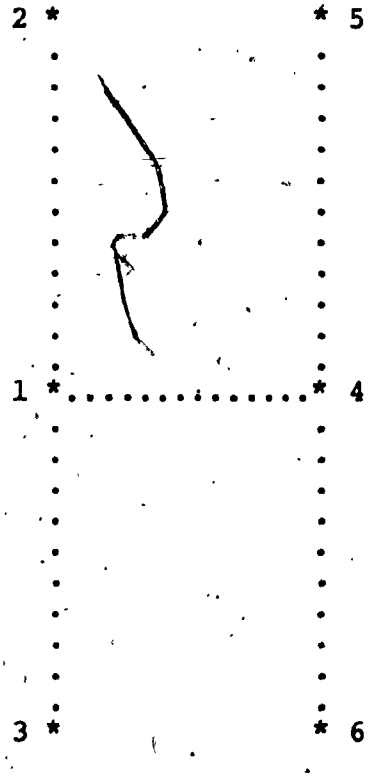
Apart from finding the node-points, procedure ADJACENT outputs all the node- and branch-points that form the graph. What follows is the description of the adjacent procedure.

Algorithm 3.7 Adjacent procedure

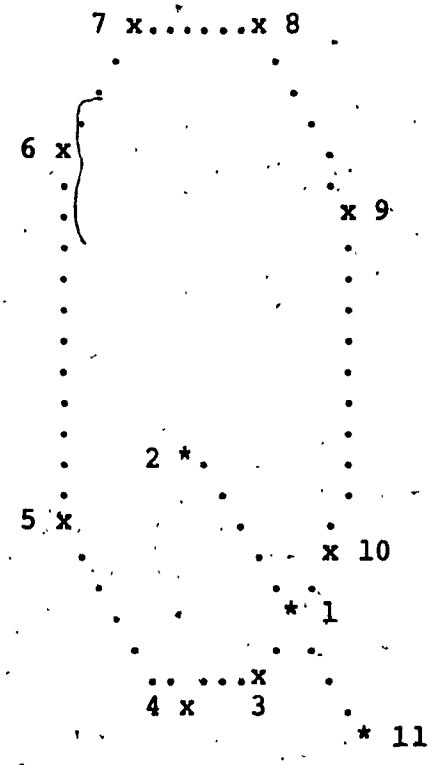
Notation: Same as in Algorithm 3.6

Steps

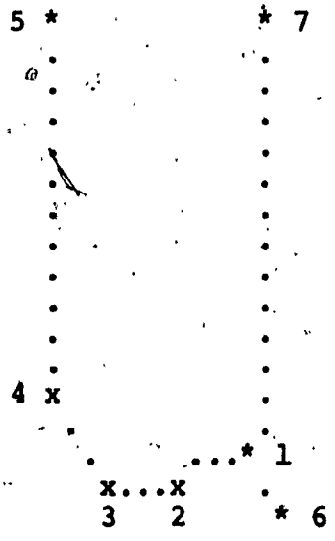
1. Set n = 0
2. For every skeleton and un-marked point (i,j),
   belonging to the set of the 8 neighbours of point p,
   Do steps 3-5
3. If Point (i,j) = 'C' or 'B' Then
   3.1 Output it
   3.2 Mark it
4. If Point (i,j) is a node-point then
   4.1 Set n = n + 1
   4.2 Set N_n = Point (i,j)
5. Else Get the next un-marked neighbouring Point (i', j')
   of Point (i,j) and Go To Step 3



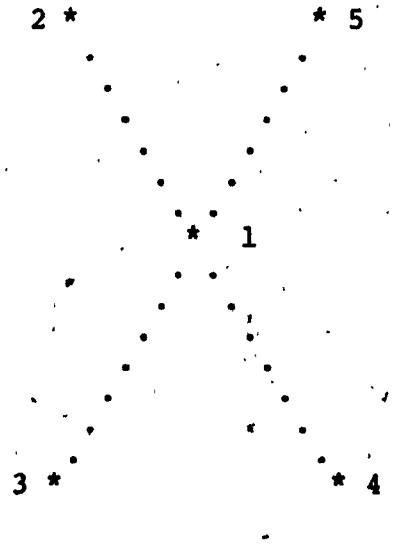
(a) Upper-case "H"



(b)) Upper-case "Q"



(c) Lower-case "u"



(d) Lower case "x"

Figure 3.7 Graph representations of characters. The *'s are the node-points; the x's are the branch-points; the .'s are the points ignored. The numbers show the traversal sequence.

{ there are 8 such points }

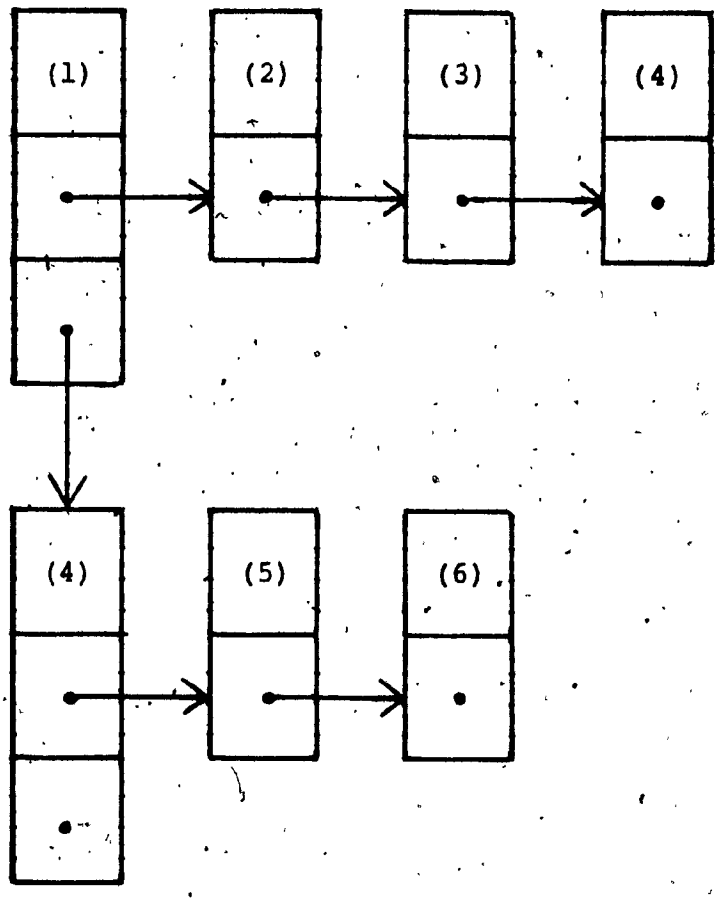
Step 3.1 of algorithm 3.7 may be omitted if the distance between two successive points (that are output) is less than a threshold (a threshold of 3 was set).

Figure 3.7 shows the graph representations of certain characters. The *'s and x's are the node-points and the branch-points respectively. These are the points that form the code. The .'s are the points ignored. Besides each node-point or branch-point there is a number indicating the sequence of points visited during the graph traversal algorithm.

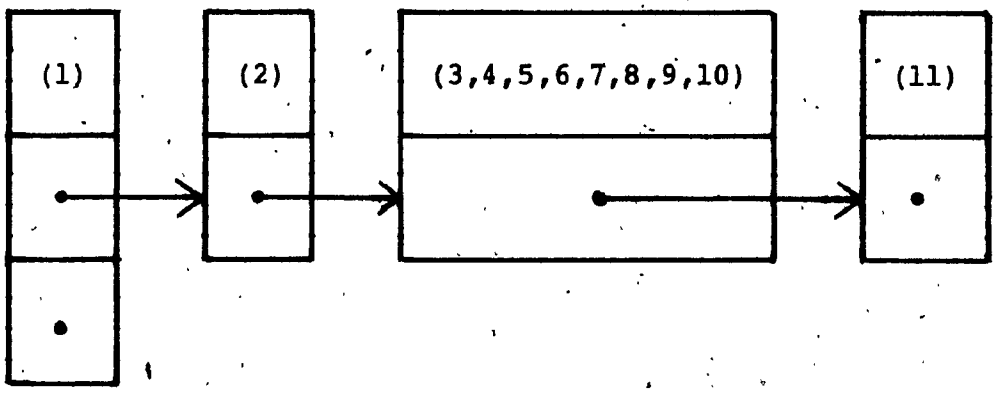
Figure 3.8 illustrates the linked representation of such graphs as maintained by the Character Generator (C.G.).

3.7 Interactive Graphics Editor

The Interactive Graphics Editor (I.G.E.) provides a user-computer interface and incorporates a simple shape management system. Its purpose is twofold: a) To modify the graph representation (i.e. code) of a character in order to adjust possible inaccuracies, and b) to define new codes for related characters or other shapes. Notice that for high

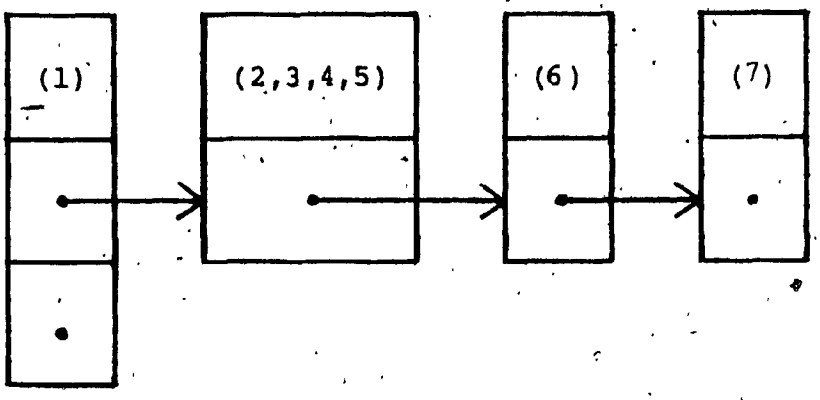


(a) Upper-case "H"

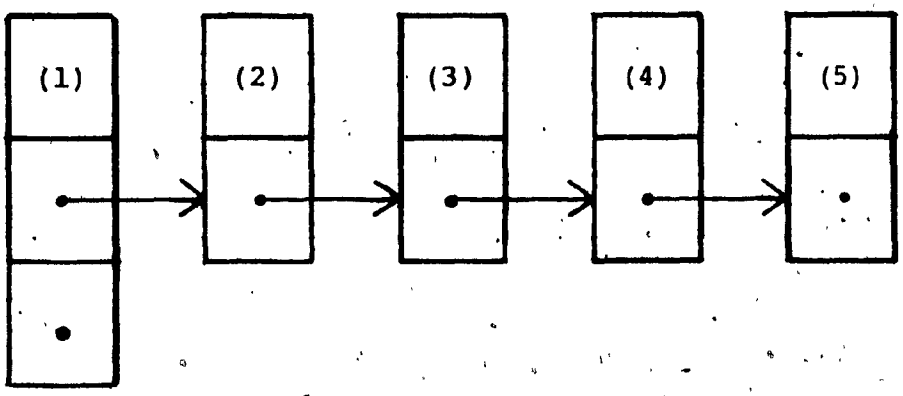


(b) Upper-case "Q"

Figure 3.8 Linked representations of graphs.



(c) Lower-case "u"



(d) Lower-case "x"

Figure 3.8 Continued.

resolution devices, certain code inaccuracy adjustments (i.e. the branch-point labelled as 'C' of upper-case "Q" in Figure 3.6 (b)) will not be necessary.

The I.G.E. executes in the form of a closed loop. The first step of the loop prints out a prompt sign ("?")

indicating that it is ready for a command input. What follows is a list of the main commands along with a description of their functions.

	Command	Description
1.	Get, name	Get a character shape*
2.	Display, name	Display a character shape**
3.	Erase	Clear the screen**
4.	Create, name	Create a character shape***
5.	Change, name	Change a point*** of a character shape
6.	Rename, n1, n2	Rename a character shape
7.	Delete, name	Delete a character shape
8.	Store, name	Store a character shape
9.	Save	Save the whole data base
10.	End	End the program

* A code is assigned to each character shape, i.e. 1 for upper-case "A", 2 for Upper-case "B", etc.

** The Tektronix (PLOT 10) Graphics system was used.

*** The coordinates of points are received by the joystick or thumbwheels input device.

The functions of the I.G.E. as well as the code derivation process itself may be done in an interactive mode or manually. Notice that this manual operation does not

require considerable time and effort, since the code bears the main structure of a character.

Chapter 4

The Character Generator

4.1 Introduction

In this chapter the second process of the system, i.e. the decoding or generation process, is discussed. Its function depends in a way on the code - represented in the form of graph - the derivation of which was described in the previous chapter.

By applying simple and different transformational operations on a single graph of a letter, different styles of characters are obtained. A total of five such styles are discussed. A variety of other styles may be devised by either mixing their properties or introducing new ones. The characteristics of the type styles under consideration may represent standard or non-standard character fonts and are used in many applications such as advertisements, etc. Experimental results and illustrations of the type styles will be given in the subsequent chapter.

As in [Knu79, Knu82], characters are generated by

describing (using the code in our case) the motion of the center of a pen. The different types of pens and their parameters will be discussed. The type styles may be a serif or non-serif design. The origins, importance and the different types of serifs will be included. Family variations in size, boldening and orientation have been achieved.

What is challenging and important to note, in this chapter, is the generation of characters in different styles and sizes, from a single description (i.e. its code).

4.2 Type Styles

The code of each letter is read from the Data Base (D.B.) and is maintained by the program in the form of a linked list (see Figure 3.7).

4.2.1 Type Style 1

This is the easiest style to generate, since it is based, in a way, on a look-up operation. Characters are generated by following their corresponding linked

Figure 4.1 Samples of characters of Type Style 1.

representations in a straight forward manner. The MOVETO a point and the LINETO a point are the basic operations to draw straight lines. Curved strokes are formed by joining pairs of adjacent points with straight-line segments.

For example, looking at Figures 3.7 and 3.8 upper-case "H" is generated as follows: A line is drawn from point 1 to point 2, another from 1 to 3, and another from 1 to 4. Then, a line is drawn from point 4 to point 5, and another from 4 to 6. The curved stroke of lower-case "u" is made of small straight-line segments starting at point 1, going through points 2, 3, 4, and finishing at point 5. Figure 4.1 shows some samples of characters drawn in this Type Style.

4.2.2 Type Style 2

The design of this style is the same as style 1 with the exception of the curved strokes. A curved stroke is a smooth curve passing through a set of points (obtained from the code), and is derived by an interpolation process. Some interpolation techniques are discussed in [Har83, pp. 377-399] (see also [Pav82, pp. 215-230]). The interpolation process used to generate curved strokes was taken from [Har83] and is summarized as follows:

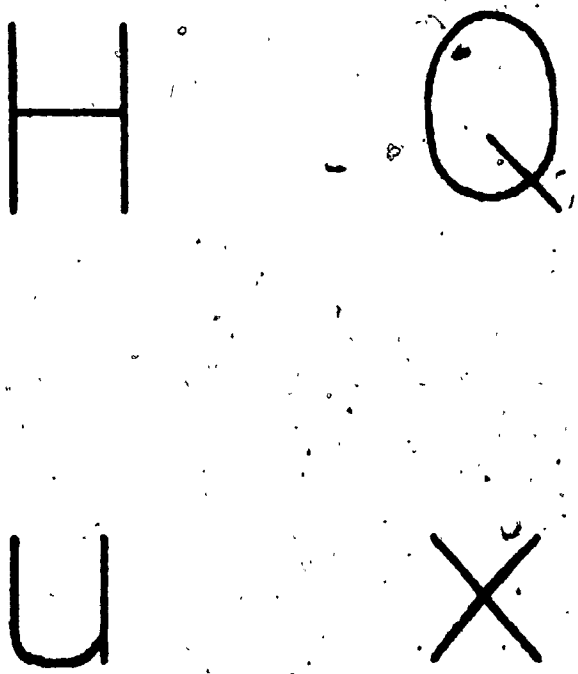


Figure 4.2 Samples of characters of Type Style 2.

Let (X_1,Y_1), (X_2,Y_2), ..., (X_n,Y_n) be a set of points. Then, a polynomial curve passing though these points, is given by the following function:

$$f_x(u) = \sum_{i=1}^{n} x_i B_i(u)$$

$$f_y(u) = \sum_{i=1}^{n} y_i B_i(u)$$

Where $B_i(u)$ are called the blending functions. For a set of four sample points (as this is the case) the four blending functions are:

$$B_1(u) = u(u-1)(u-2)/(-6)$$
$$B_2(u) = (u+1)(u-1)(u-2)/(-2)$$
$$B_3(u) = (u+1)u(u-2)/(-2)$$
$$B_4(u) = (u+1)u(u-1)/6$$

Hence, given n sample points, we first take the four consecutive points (1, 2, 3, 4), such that the approximated curve will pass through the two middle points (2, 3). Then, we step up one sample point, picking up a new sample point at one end and discarding a point at the other end (2, 3, 4, 5). We can then approximate the next portion of the curve (3, 4). We continue moving through the sample points until

the curve is drawn. Notice that, if each section is approximated by three straight-line segments, then each section will require the blending function values for u at 0, 1/3, 2/3, and 1.

Some samples of characters drawn in this Type Style are shown in Figure 4.2. Notice the smoothed curved strokes in the upper-case "Q" and lower-case "u".

4.2.3 Type Style 3

The characteristics of this style are mainly the same as in style 2. Characters are formed by a combination of straight and/or smoothed curved strokes, depending on their corresponding code. What makes this style unique is that the width of a straight stroke varies uniformly along the stroke.

Furthermore, certain strokes need to be extracted from the code. Such strokes are obtained whenever the absolute slope difference between two adjacent lines (i.e. strokes having a common node-point) is less than a threshold (a threshold of 0.3 was set for this condition). That is, a stroke is present, if:

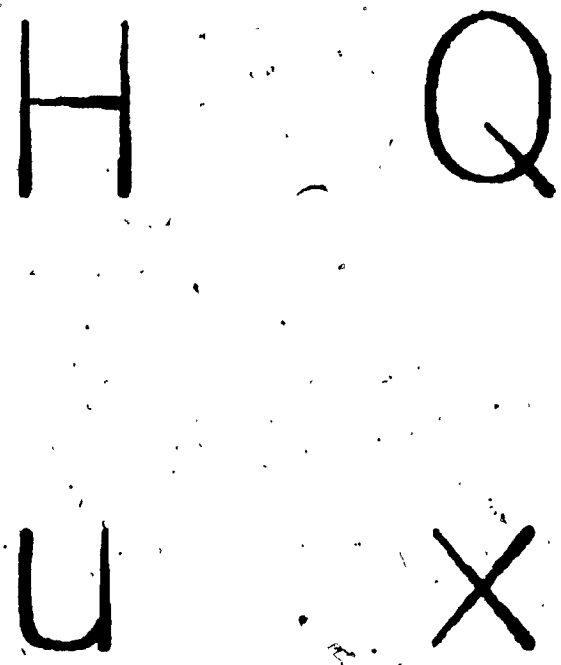


Figure 4.3 Samples of characters of Type Style 3.

$$|S_1 - S_2| < \text{Threshold}$$

where S_1 and S_2 are the slopes of the two adjacent lines.

For example, looking at Figures 3.7 and 3.8 the following strokes will be extracted for upper-case "H" : a) A stroke from point 2 to point 3. It is formed from the lines: 1 to 2 and 1 to 3, b) A stroke from 1 to 4, and c) A stroke from 5 to 6. It is formed from the lines: 4 to 5 and 4 to 6.

Stroke width variation is accomplished by changing the parameters (i.e. height, width) of the pen (see section 4.3) used. If LEN is the length of a stroke, P stands for the one parameter (either height or width) of a pen, then, the resulting size of the pen varies from 1 to P along the stroke with an increment of P/LEN.

Figure 4.3 shows samples of characters representing the Type Style discussed in this section. Notice the stroke width variation in the straight strokes.

4.2.4 Type style 4

This type style is characterized by circular arcs made out of straight strokes. Again, the curved strokes may be

smooth curves, and the extraction of certain straight strokes is needed and performed as described in section 4.2.3. The major rules used to transform a straight stroke into a circular arc are:

a) A vertical stroke (i.e. no slope) is converted into a circular arc, opened : i) leftwards if it lies in the left part of a character, or ii) rightwards if it lies in the right part of the character

b) A horizontal stroke (i.e. slope is zero) is converted into a circular arc opened : i) upwards if it lies in the top part of the character, or ii) downwards if it lies in the bottom part of the character

c) An inclined stroke (i.e. slope is positive or negative) is converted into a circular arc opened : i) leftwards-upwards if its slope is positive, or ii) rightwards-upwards if its slope is negative.

Figure 4.4 illustrates the construction of a circular arc. $A(X_A,Y_A)$ and $B(X_B,Y_B)$ are the end points of the straight stroke AB and $M(X_M,Y_M)$ is its mid-point. The center $C(X_C,Y_C)$ of the curvature is found as follows :

Let: $a_1 = X_A - X_M$ and $a_2 = Y_A - Y_M$.

Since CM is perpendicular to AB, we have:

$$a_1(X_C - X_M) + a_2(Y_C - Y_M) = 0 \quad \text{or}$$

$$X_C = X_M - a_2/a_1(Y_C - Y_M) \qquad (4.1)$$

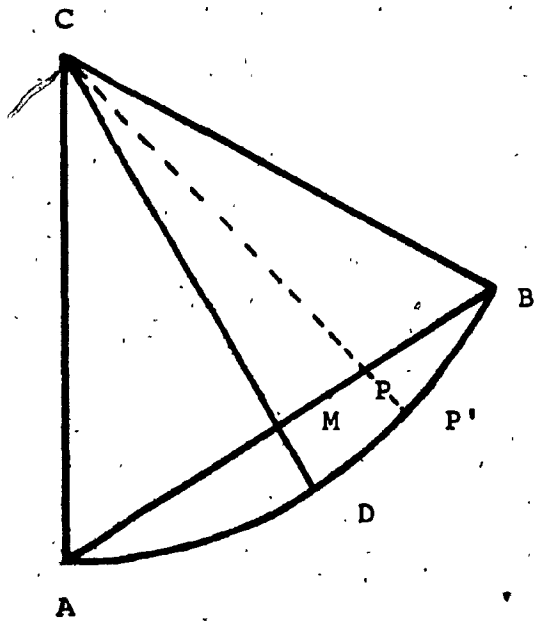


Figure 4.4 Finding the center C of curvature.

Furthermore, the distance r between A and B (i.e. the radius of the curvature) and the distance d between C and M are, respectively:

$$r^2 = (X_B - X_A)^2 + (Y_B - Y_A)^2 \quad (4.2)$$

$$d^2 = (X_C - X_M)^2 + (Y_C - Y_M)^2 \quad (4.3)$$

Since the triangle ABC is equilateral (by definition), we have:

$$d^2 = r^2 - (r/2)^2 \quad (4.4)$$

From (4.2) and (4.4) we can find the value of d. From (4.1) and (4.3) we can find Y_C. Hence, the center $C(X_C,Y_C)$ of the curvature is given by:

$$X_C = X_M - a_2/a_1 (Y_C - Y_M)$$

$$Y_C = Y_M \pm d/\sqrt{(a_2/a_1)^2 + 1}$$

Then, a point $P(X_P,Y_P)$ of line AB (and this may be, for example, a cross-point) is placed in position $P'(X_{P'},Y_{P'})$ of the circular arc, and is found as follows: Since lines CP and CP' have the same slope m, and lines CP' and AB have the same length r, we have:

$$m = (Y_C-Y_P)/(X_C-X_P) = (Y_C-Y_{P'})/(X_C-X_{P'}) \quad (4.5)$$

$$r^2 = (X_C - X_{P'})^2 + (Y_C - Y_{P'})^2 \quad (4.6)$$

From (4.5) and (4.6) it follows that:

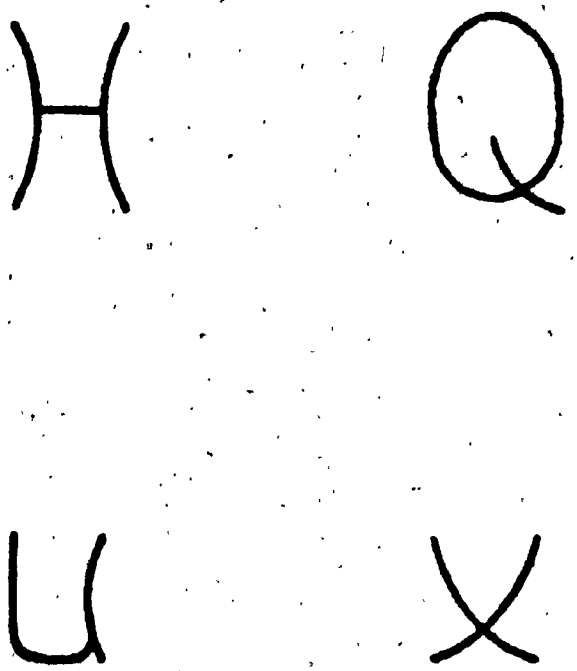


Figure 4.5 Samples of characters of Type Style 4.

$$Y_{P'} = Y_C \mp r/\sqrt{1/(m^2) + 1}$$

$$X_{P'} = X_C - 1/\{m(Y_C - Y_{P'})\}$$

Samples of characters representing this style are shown in Figure 4.5. Notice the conversion of the straight strokes into circular arcs, and the adjustments, if applicable, to certain cross-points.

4.2.5 Type Style 5

This style bears certain characteristics of the Roman family of fonts. It is mainly characterized that within a character: i) certain strokes (either straight or curved) are thin or thick, and ii) the serifs used are of the "slab" type (see section 4.4.2). The major rules that determine whether a stroke should be thin or thick are, in sequence of order:

a) A straight stroke with positive slope must be thin. Exceptions to this rule are the upper and lower case "Z"
b) Two or more thick strokes never meet (i.e. this is the case for upper case M or N, for example).
c) Use a horizontal elliptical pen (see section 4.3). This has as a consequence the following:
   1) Inclined and vertical strokes are thick (except those

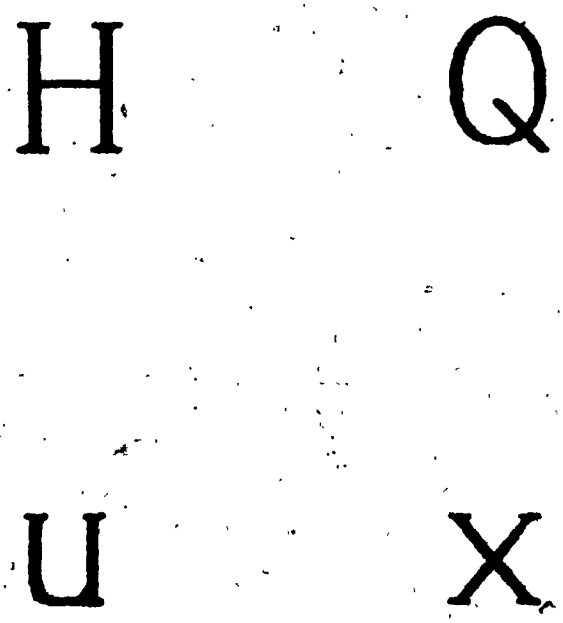


Figure 4.6 Samples of characters of Type Style 5.

that violate rule a).

2) Horizontal strokes are thin

3) Curved strokes with upward or downward openings are thin

4) Curved strokes with leftward or rightward openings are thick

The exception to this rule is the case of the right vertical stroke the upper-case "U".

Figure 4.6 shows some samples of characters drawn in Type Style 5. Notice the thickness of the strokes as well as the serifs placed at the stroke ends.

4.3 Pens / Erasers

Pens/erasers for character generation have been used in [Knu79, Knu82], and their use represents the imitation of the human hand. The center of a pen is going (moving) from point to point, while drawing a stroke. The shape of a stroke (straight or curved) is affected by the type of pen used (i.e. stroke width). The different types of pens used are:

- Quadrangular
- Horizontal rectangular

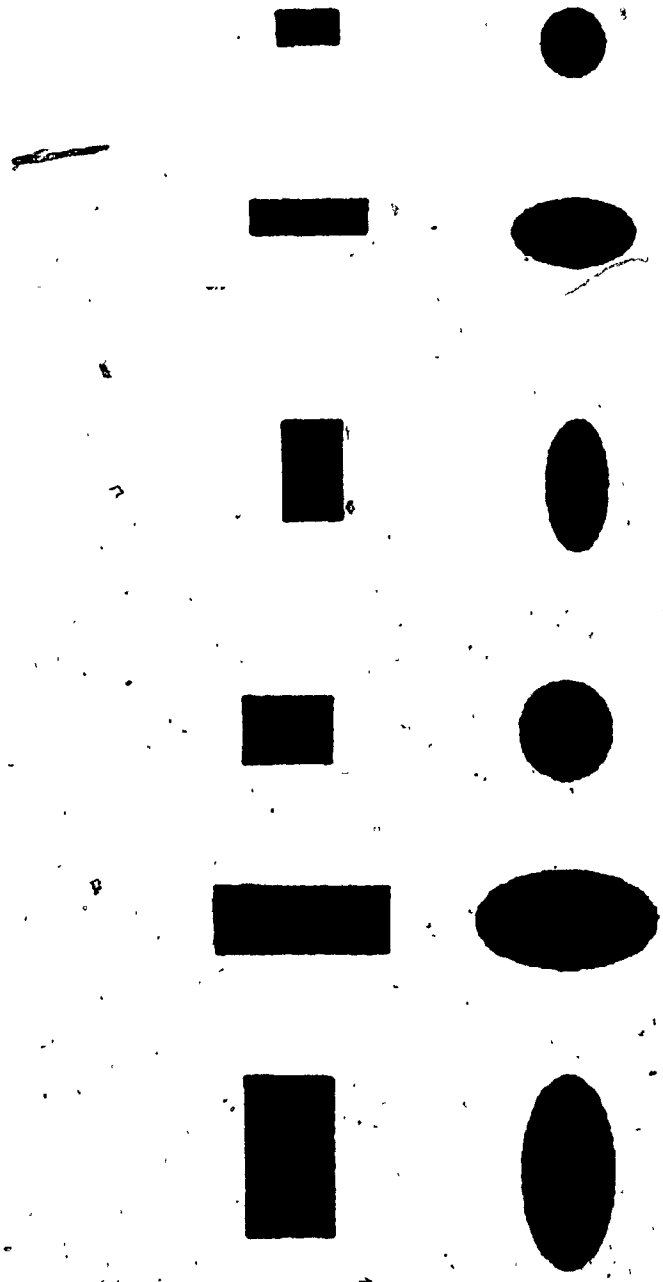


Figure 4.7 Pens in different types and sizes.

- Vertical rectangular
- Circular
- Horizontal elliptical
- Vertical elliptical

The parameters of a pen are its height and width. The quadrangular or circular pen need only one parameter, i.e. the length of one side or its radius, respectively. An elliptical pen with h-height and w-width consists of all points (x,y) such that:

$$x^2/(w/2)^2 + y^2/(h/2)^2 \leq 1$$

Similarly, a rectangular pen consists of all points (x, y) such that:

$$1 \leq x \leq w \quad \text{and} \quad 1 \leq y \leq h$$

The pen or eraser is determined by the state of the points, which may be black or white, respectively. Figure 4.7 shows various types of pens in different sizes.

4.4 Serifs

4.4.1 The origin and importance of the serif

Serif (or marker) is a portion of a character which, if removed, will not result in a loss of or an ambiguity in the character's identity [Cox74]. Serifs can be viewed as embellishments in a character and are placed to the ends of strokes.

Two questions are associated with serifs: i) How did the serif arise ? That is, its origin, and ii) What purpose does it serve ? That is, its importance. There exist several theories on the origin of the serif [Hoc73]. Among them, one believes that the form of the serif was conditioned by the chisel, while others are concerned not so much with the stone-cutting tool as with the method of design or draft used by the lettercutter. In [Hoc73], it is believed that the use of the brush is the key to the origin of the Roman serif.

In answering the second question, again, several theories exist [Rob71]. One theory believes that the choice of typefaces is a matter of conditioning. People prefer certain typefaces because they have grown accustomed to them. Another theory states that the serifs increase the horizontal continuity of a line of type. While still another theory states that serif-form characters convey more information to readers because there are more lines present in each character than in equivalent sans-serif forms. In [Rob71] such theories are being refused and it is stated

that serifs are important in the perception of small characters by humans. It is suggested that the neurological structure of the human visual system benefits from serifs in the preservation of the main features of small characters during neural processing. Furthermore, serifs are not useful for large-sized characters.

4.4.2 Derivation and types of serifs

There exist several types of serifs, i.e. slab (or square), hairline, bracket, wedge, etc. [Gas76]. In this thesis, two types were considered: slab and bracket, because there are the most commonly used. The hairline may be obtain from the slab type (when w = 1, see serif parameters below). The wedge also can be derived from the bracket, accordingly. A variety of serifs in different sizes is obtained by setting its two parameters: height and width (usually width is greater than height).

Figure 4.8 illustrates the construction of the serif. The h and w are the height and width, respectively. Point p is the center of the serif and represents the location of the serif placed within the character (note: it is extracted from the code). The slab type can be easily constructed from the mentioned parameters. However, the bracket type

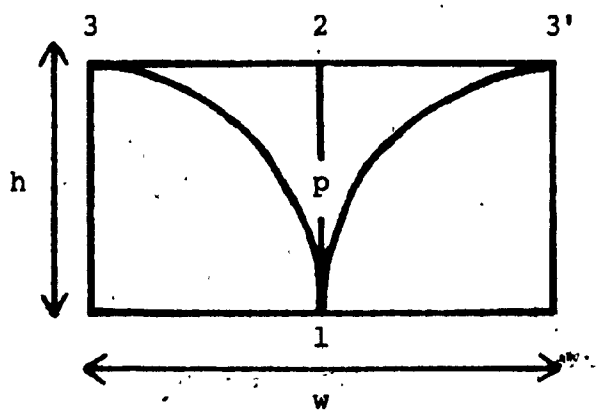


Figure 4.8 The construction of the serif

needs the extraction of the points 1, 2, 3, and 3' (see Figure 4.8). It is formed from two parts 123 and 123'. Each set of points (either 1, 2, 3 or 1, 2, 3') is approximated by a curve using the Bezier polynomials. Some curve fitting problems are discussed in [Pav82, pp. 215-230]. The Bezier polynomial used for serif construction can be summarized as follows:

Given points $P_1, P_2, \ldots, P_m$ (m is set to 3 in our case) the Bezier polynomial is defined as follows:

$$P_{1,m}(t) = P_{1,m-1}(t) + t\{P_{2,m}(t) - P_{1,m-1}(t)\}$$

where $P_{i,j}(t)$ denotes the Bezier polynomial for $P_i, P_{i+1}, \ldots, P_j$.

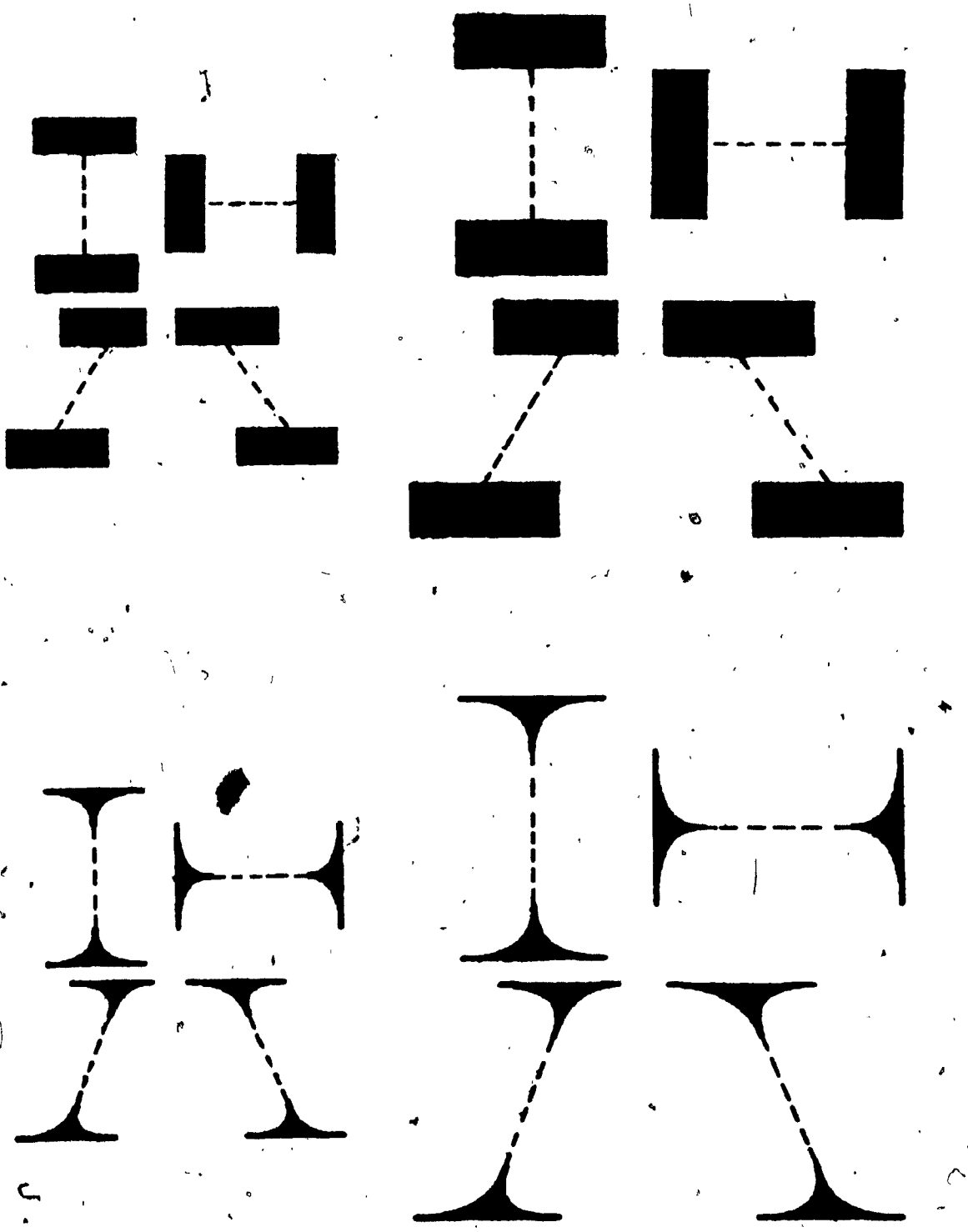


Figure 4.9 Serifs in different types and styles.

Serifs are mainly placed at the ends of strokes. Their orientation depends on the stroke orientation as follows:

Stroke Orientation	Serif Orientation
Vertical	Horizontal
Horizontal	Vertical
Curved	Vertical

The major rules to place serifs on a character are:

a) All stroke ends take serifs

b) A horizontal stroke end on the top or bottom of a character takes a half-sized serif. This serif is placed on the top or bottom side of the stroke if the stroke lies in the bottom or top part of the character, respectively

c) A Vertical stroke corner takes a half-sized serif. This serif is placed on the left or right side of the stroke if the stroke lies in the left or right part of the character, respectively.

Note: A stroke end is an end point of a stroke without intersection with another stroke. A stroke corner is an end point of a stroke intersected by another stroke.

Figure 4.9 shows the various types of serifs, in different sizes. They are placed at the ends of different stroke orientations.

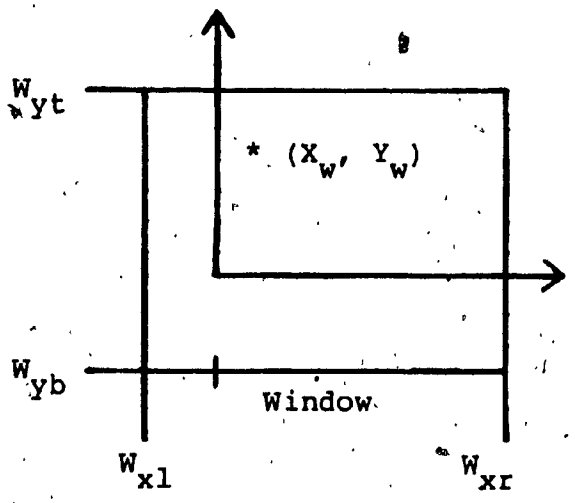
4.5 Variations of type styles

Among the advantages of the proposed method is the flexibility provided. Given a type style, a variety of different automatic modifications can be accomplished, including:

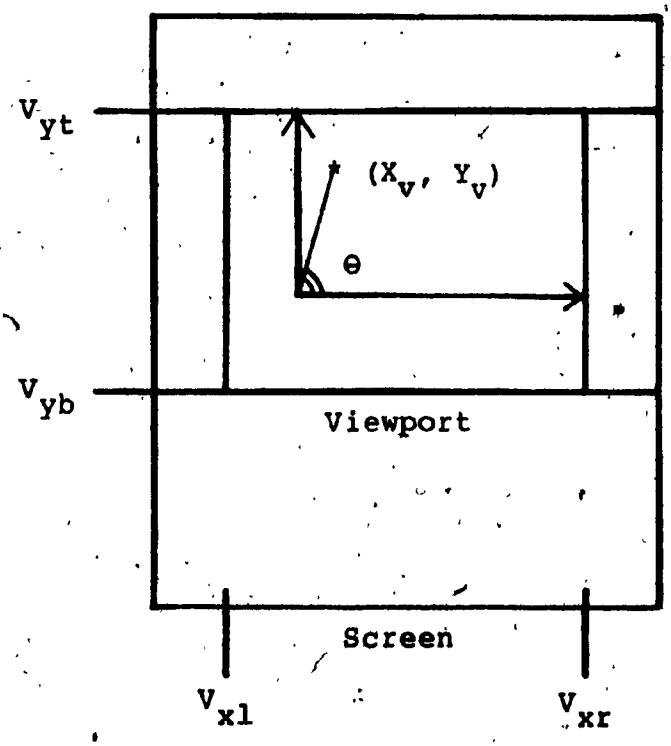
- Boldening
- Size variation
- Orientation

Boldening refers to the thickness of a stroke (i.e. thin or thick stroke). This is accomplished by the use of pens. One has to select a type pen and set the appropriate values to its parameters (see section 4.3).

Size variation refers to the different character sizes (i.e height and width). This is accomplished by the so-called windowing transformation described as follows: The coordinates of the code (representing the characters) reside in the so-called virtual space. It is a rectangle, called window, (defined by the user) with lower-left corner at (W_{xl}, W_{yb}) and upper-right corner at (W_{xr}, W_{yt}) (see Figure 4.10 (a)). Every point in the virtual space is to be mapped onto the so-called absolute (or screen) space. It is a rectangle, called viewport, with lower-left corner at (V_{xl}, V_{yb}) and upper-right corner at (V_{xr}, V_{yt}) (see Figure 4.10 (b)). The viewport is part of the screen where the window's



(a) Virtual space



(b) Absolute space

Figure 4.10 The windowing transformation.

contents are to be displayed. A point (X_w, Y_w) in the window is mapped onto a point (X_v, Y_v) in the viewport as follows:

$$X_v = \frac{V_{xr} - V_{xl}}{W_{xr} - W_{xl}} (X_w - W_{xl}) + V_{xl}$$

$$Y_v = \frac{V_{yt} - V_{yb}}{W_{yt} - W_{yb}} (Y_w - W_{yb}) + V_{yb}$$

By changing the ranges for the virtual and/or absolute space(s), different character sizes may be obtained. In practice, we keep W_{xl}, W_{xr}, W_{yb}, W_{yt}, V_{xl} and V_{yb} constant and we change V_{xr} and V_{yt} (representing the width and height of the character, respectively).

Orientation refers to the different character slantings, and can be:

- Horizontal orientation. A character is pushed to the left or right from its top part keeping its bottom part fixed
- Vertical orientation. A character is pushed to the top or bottom from its left part keeping its right part fixed.

There exist two parameters, Θ_1 and Θ_2, for the two orientations which, if desired, may be performed simultaneously. The horizontal orientation is performed as follows (see Figure 4.10 (b)):

Given an angle Θ, we have:

$\text{Tan}(\Theta) = Y_v / X_v$ or

$$\text{Tan}(\theta) = \text{LENGTH} / (V_{xr} - V_{xl} + 1)$$

Where LENGTH is a variable which can be obtained from the above equations. Then, starting from the top (V_{yt}) and going down to the bottom (V_{yb}) the slanting varies from LENGTH to 0, with a decrement of LENGTH/($V_{yt} - V_{yb}$). Similarly, the vertical orientation is accomplished. Illustrations will be given in the next chapter.

Chapter 5

Experimental Results

The algorithms described in chapter 3 and 4 were written in PASCAL and run on the CDC Cyber 170/835 Computer. The generated characters were displayed on the Tektronix 4010/1 display device. It is a storage-display CRT terminal with a screen of 1024x780 addressable points, and measuring 8x6 square inches. It should be emphasized that the implementation of the proposed system is device independent. A generated character was temporarily held in a buffer and then displayed (using the run-length coding) on the device.

A variety of related fonts can be obtained by choosing a number of global parameters. These parameters are entered interactively by the user, and there are totally eight of them, viz:

- Two parameters are used to determine the size (height, width) of a character
- Two parameters are used to determine the size (height, width) of the pen. The circular pen needs only one

parameter, i.e. the radius

- Two parameters are used to determine the size (height, width) of a serif
- Two parameters are used to determine the orientation (horizontal, vertical) of a character. For illustration see Figures 5.15 - 5.19.

Notice that the parameters used to determine the size of the character, the size of the pen, and the size of the serif take their values from the absolute space. The rest of the parameters take their values from the virtual space (see chapter 4, section 4.5).

Tests were conducted on five character fonts. One of them was intuitively designed by the author. The other four are the following standard character fonts:

a) Orator. It is a sans-serif type style recommended especially for speeches, charts or headlines requiring the utmost legibility and impact
b) Letter Gothic. It is a sans-serif design, and ideal for invoices, statements or other typing applications
c) Courier. It is a square-serif design, and ideally suited for reports, official correspondences, judicials, and offset reproductions
d) Pica. It is a square-serif design, and well suited for general correspondences, stencils and multiple copies.

Table I. The size (rows, columns) of different character fonts.

Character Font	Height-H (rows)	Width-W (columns)	Size (HxW)
Orator	24.15	19.21	463.92
Gothic	27.00	16.26	439.02
Courier	21.25	19.26	409.27
Pica	25.05	20.61	516.28

These standard character fonts were typed by an IBM typewriter and digitized by an ECRM Autoreader. Table I shows the average size (height, width) of a character in different character fonts. Notice that all results (shown below) were derived by averaging over the 52 (i.e upper and lower case) characters in a given font.

Each digitized character font was fed into our system to derive the code from which the font could be reproduced subsequently. Some experimental results are shown in Table II, where: The coding size refers to the average number of points needed to describe a character. The derivation time refers to the average CPU-time (in seconds) consumed by the

Picture Descriptor to derive the code of a character (I/O time is not included). The generation time refers to the average CPU-time (in seconds) consumed by the Character Generator to generate a character (I/O time is not included). To generate the characters, the same parameters were used for all the fonts, with the following specifications: a 35x35 window, a 80x80 viewport, and a circular pen of radius two.

From Tables I and II the following may be concluded:

a) The coding size depends on the complexity of the character font, i.e. the coding size of the sans-serif character fonts (Orator,Gothic) is smaller than that of the serif design fonts (Courier, Pica) because the former are less complex

b) The derivation time depends on the size of the characters, i.e. the Courier type style with the smallest size needs the smallest amount of time to derive its code

c) The generation time depends on the complexity (serifs, sans-serifs) as well as on the size of the characters, i.e. compare the generation time of the Gothic, Courier and Pica character fonts.

Figures 5.1 (a) - 5.4 (a) show the enlarged (four times) versions of the actual IBM type styles: Orator (This font was designed by my supervisor Dr. C. Y. Suen and

Table II. The coding size, derivation and generation CPU-time (sec) of different character fonts.

Character Font	Coding size	Derivation CPU-time	Generation CPU-time
Orator	7.0192	2.3757	0.1737
Gothic	7.5576	2.2313	0.1943
Courier	11.2500	2.0484	0.1992
Pica	11.9230	2.5369	0.2346
Our	7.0769	-	0.1908

given to me in a digitized form. Then, it was plotted by the TRILOG PRINTRONIX printer/plotter and enlarged (twice) as shown in Figure 5.1 (a)), Letter Gothic (12 pitch), Courier 72 (10 pitch) and Pica 72 (10 pitch). Figures 5.1 (b) - 5.4 (b) show the C.G. reproductions corresponding to the original character fonts. In all these figures a 80x80 viewport and a circular pen of radius two were set (in Figure 5.1 (b) the radius was set to six units long). Notice that characters with descenders (i.e. g, p, q, y) were arbitrarily raised above the base line. The two shapes (originals, reproductions) are to a large extent similar. However, it should be emphasized that:

a) Various distortions or inaccuracies exist due to:

i) Typewriter, i.e. ribbon

ii) Photocopying during the enlargement process of the original shapes

iii) Resolution of both the scanner and display devices

b) The similarity of the shapes was based on human perception. The author may be anyone but a type designer.

Given a character font, a total of five type styles (discussed in chapter 4, section 4.2) can be derived. These type styles are used in a variety of applications (i.e. advertisements) and their characteristics may represent standard or non-standard character fonts, i.e.:

- Type Style 1 does not express smoothness, like OCR
- Type Style 2 is used to reproduce the original character font in question, i.e. Orator, Courier, etc.
- Type Style 3, though seemingly crude, expresses freedom and brings to mind Chinese writing
- Type Style 4 is used to imitate (simulate), to some extent, handwriting, i.e. script, cursive
- Type Style 5 is used to imitate fonts related (belonging) to the Roman family.

Table III shows the CPU-time (in seconds) taken to generate the different Type Styles (derived from our

Table III. The generation CPU-time (sec) of the different Type Styles.

Type Style	Generation CPU-time	Pen size	Serif size	Figure number
1	0.0914	2x2	-	5.5
	0.1644	6x2	-	-
2	0.2275	2x2	-	5.6
	0.4076	6x2	-	-
3	0.1989	2x2	-	-
	0.4229	6x2	-	5.7
4	0.2527	2x2	-	5.8
	0.4745	6x2	-	-
5	0.1992	2x2	16x6	-
	0.3514	6x2	16x6	5.9
	0.6739	6x2	16x10	5.10

description). In all cases, a 60x90 viewport was set. The different sizes of the elliptical pen used, are shown in the third column. The last column indicates the Figures, when applicable, shown in the subsequent pages. From this Table we can conclude that:

a) The generation time depends on the type style in question, i.e. Type Style 1 takes the least time
b) The generation time increases as the pen size increases
c) Much of the generation time is consumed in the smoothing of curved strokes
d) Serifs (especially the bracket type) increase considerably the generation time.

Below are some different type styles derived from various character descriptions along with their figures (shown in the subsequent pages):

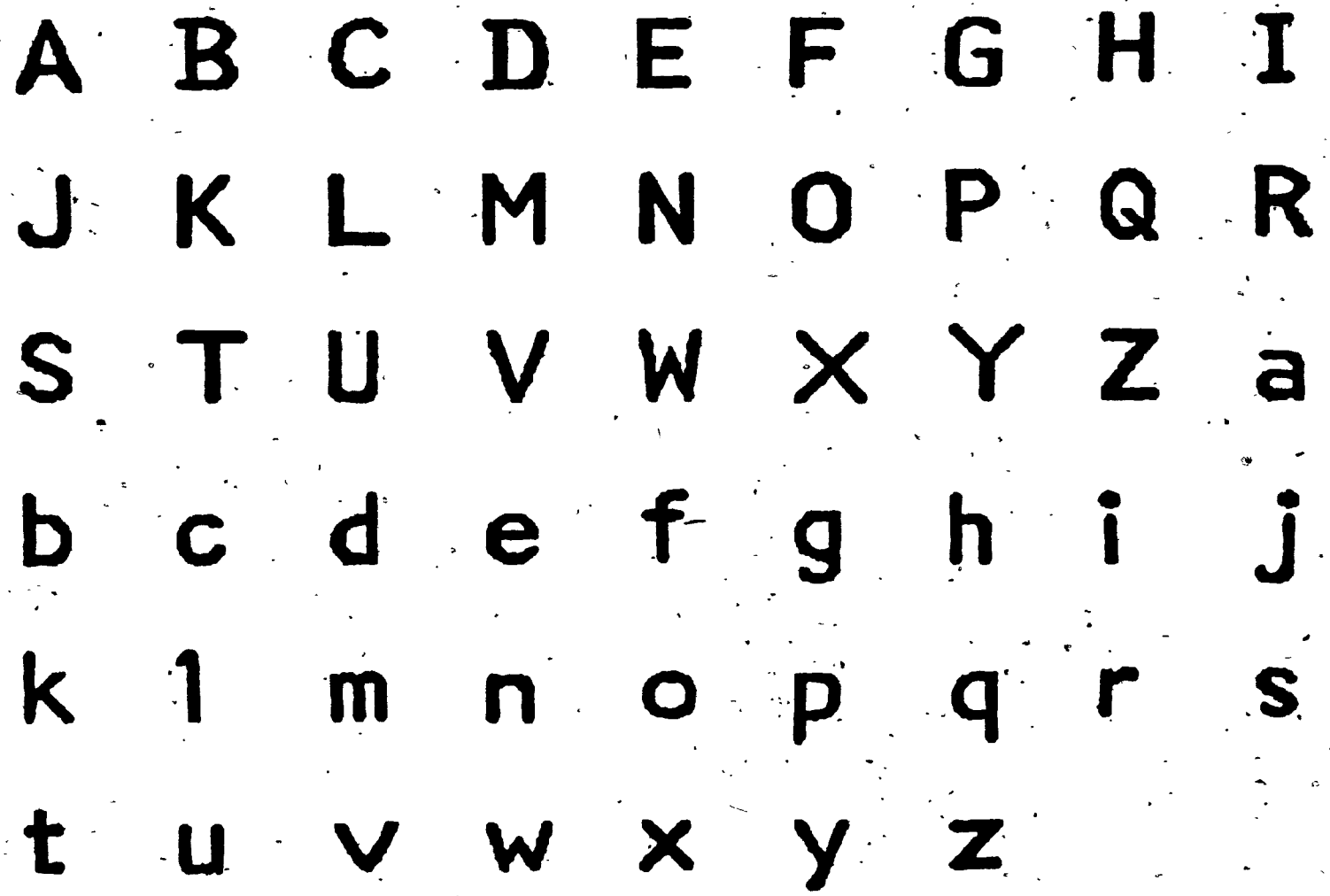
Type Style	Character Font description	Figure number
1	Pica	5.11
3	Courier	5.12
4	Orator	5.13
5	Gothic	5.14

Notice that in Type Styles 1, 3 and 4 the curved strokes were not retaining their smoothness. Serifs were

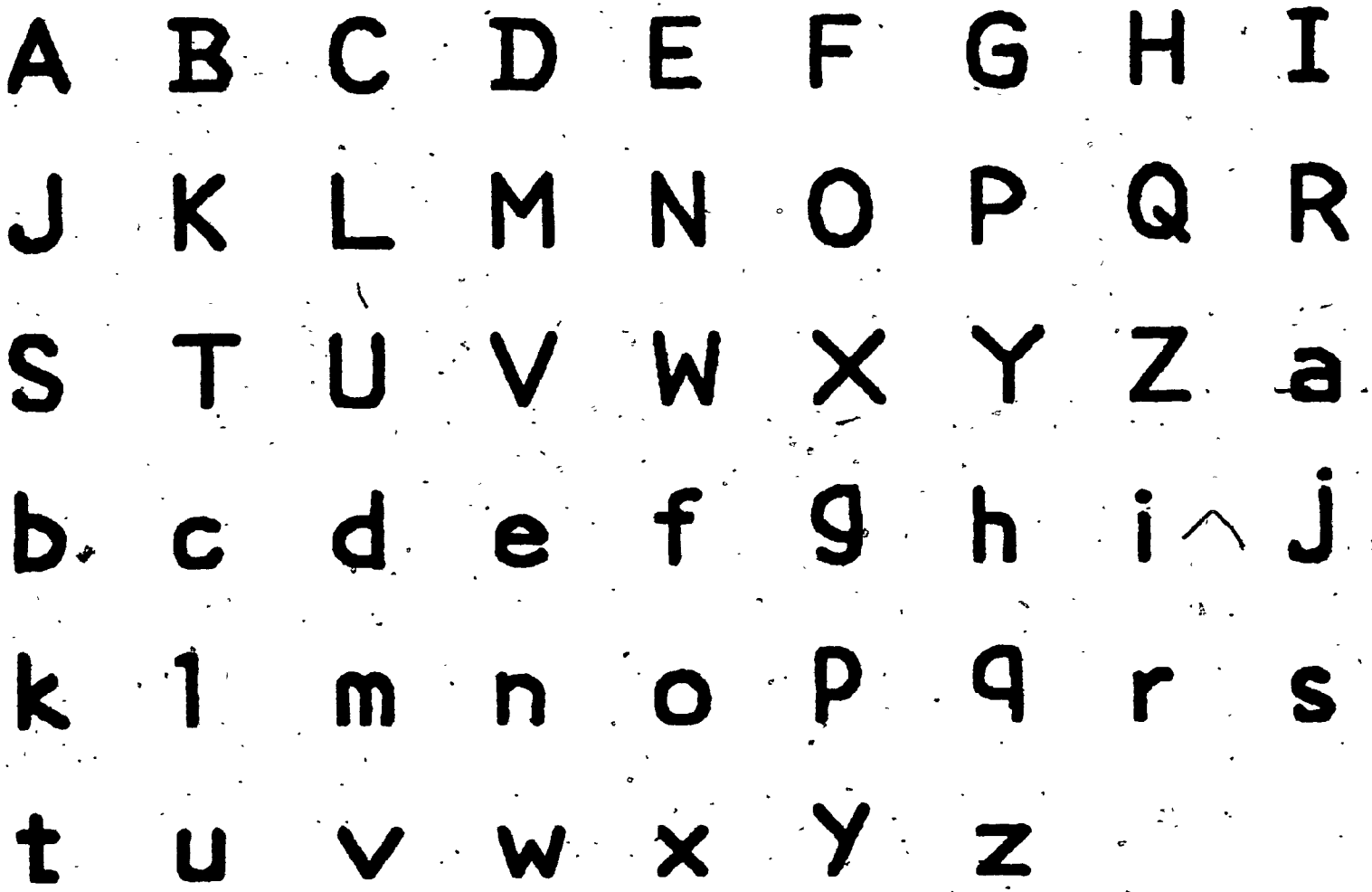
added in Type Style 5 (obtained from the Gothic description).

Below are some different slantings in degrees (of different character fonts) that affect the orientation (horizontal, vertical) of the characters, along with the figures (shown in the subsequent pages):

Slanting		Figure number
Horizontal	Vertical	
20	0	5.15
-30	0	5.16
0	20	5.17
0	-20	5.18
30	20	5.19


A B C D E F G H I
J K L M N O P Q R
S T U V W X Y Z a
b c d e f g h i j
k l m n o p q r s
t u v w x y z

(a) Original.



(b) C.G. reproduction.

Figure 5.1 Orator.

A B C D E F G H I

J K L M N O P Q R

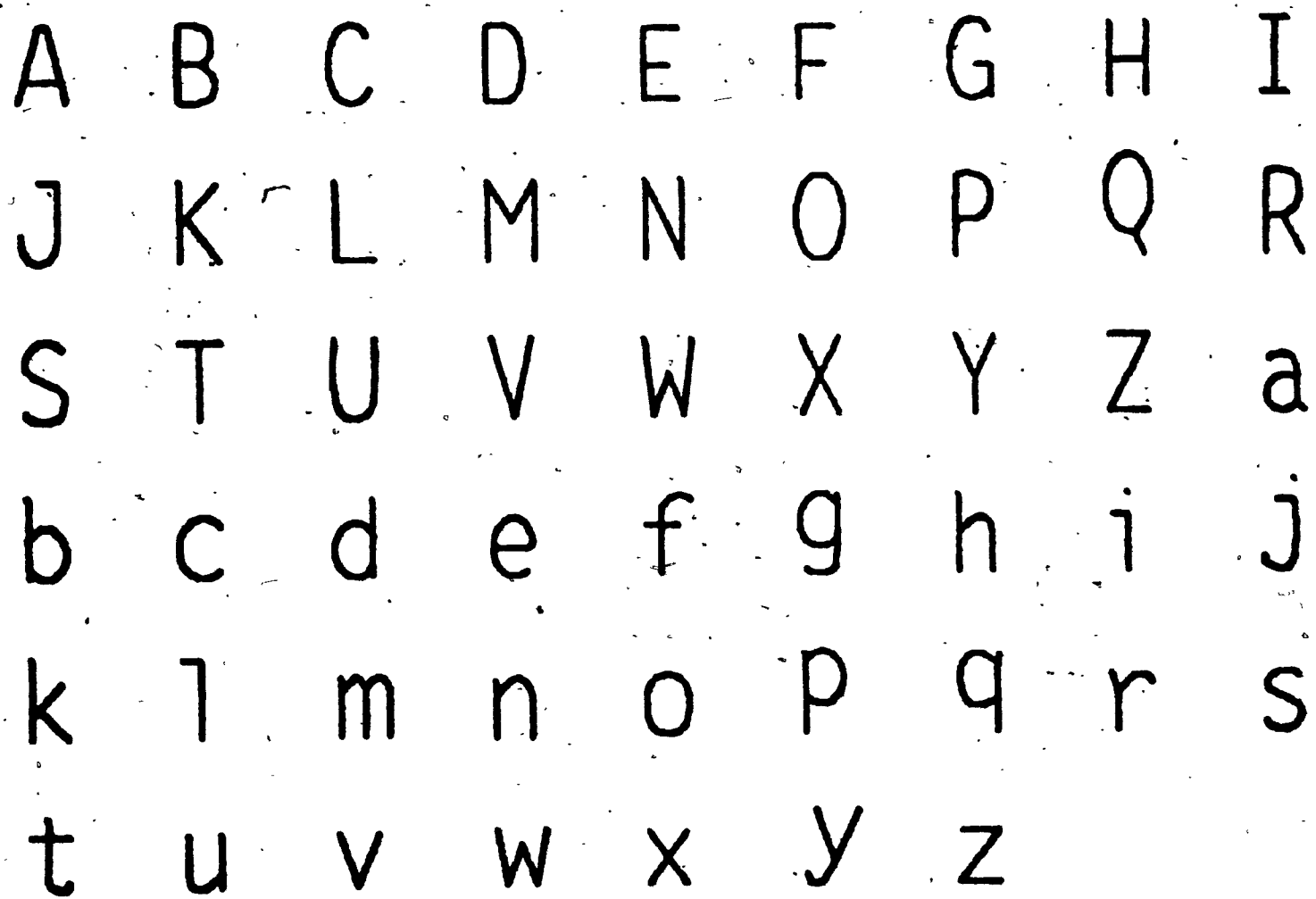
S T U V W X Y Z a

b c d e f g h i j

k l m n o p q r s

t u v w x y z

(a) Original.



(b) C.G. reproduction.

Figure 5.2 Gothic.

A B C D E F G H I

J K L M N O P Q R

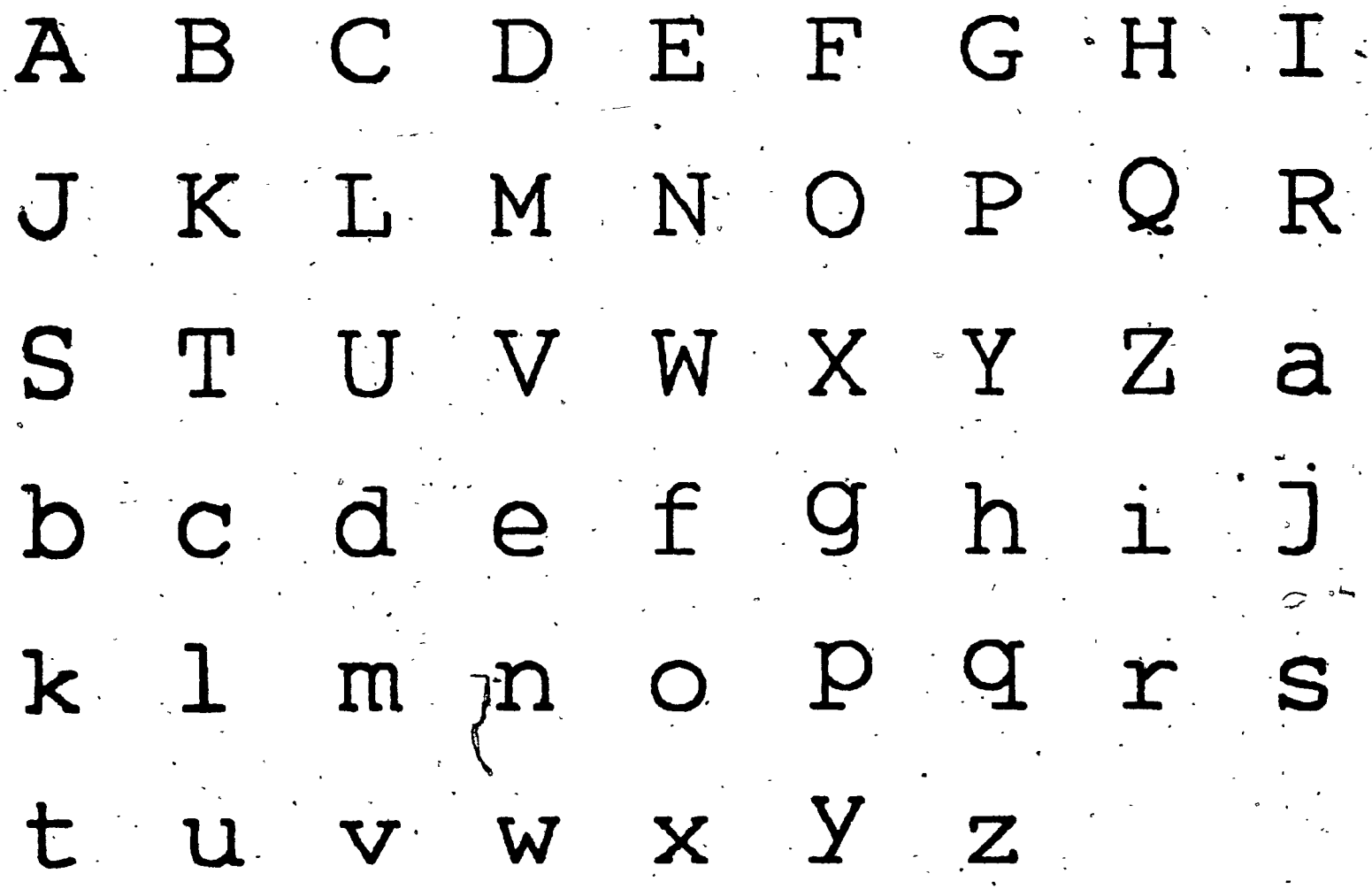
S T U V W X Y Z a

b c d e f g h i j

k l m n o p q r s

t u v w x y z

(a) Original.



(b) C.G. reproduction.

Figure 5.3 Courier.

A B C D E F G H I

J K L M N O P Q R

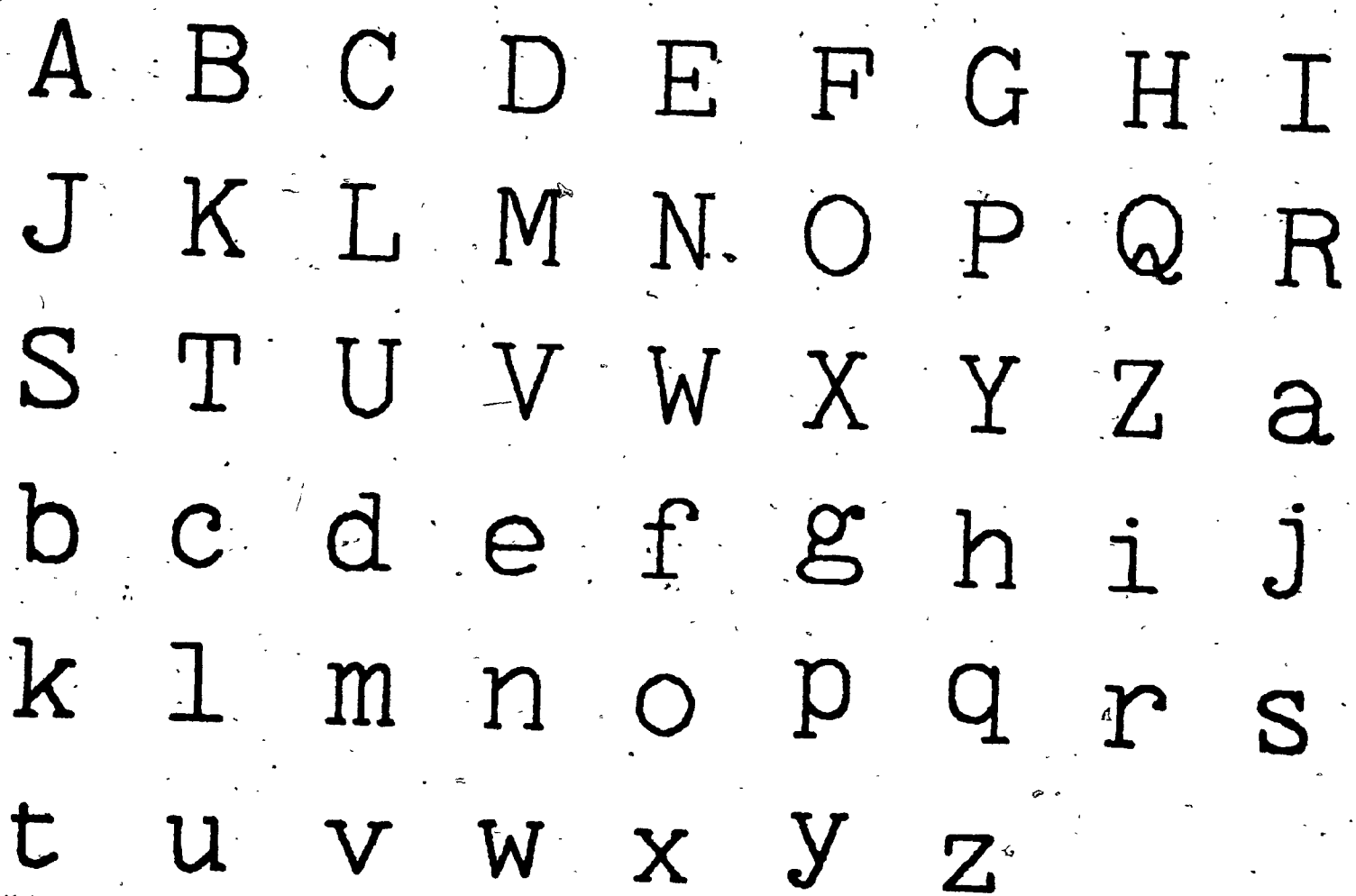
S T U V W X Y Z a

b c d e f g h i j

k l m n o p q r s

t u v w x y z

(a) Original.



(b) C.G. reproduction.

Figure 5.4 Pica.

Figure 5.5 Type Style 1.

Figure 5.6 Type Style 2.

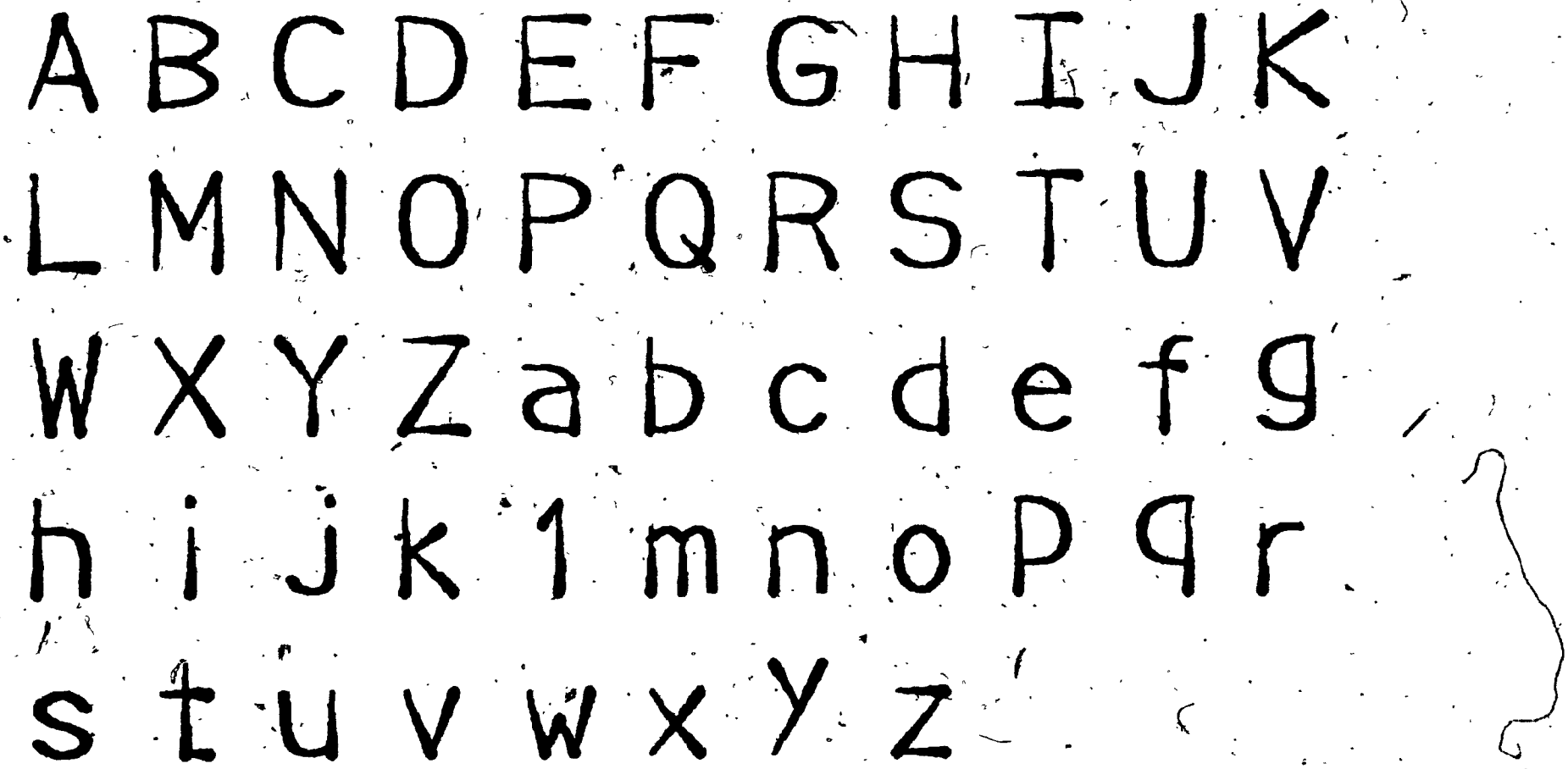


Figure 5.7 Type Style 3.

Figure 5.8 Type Style 4.

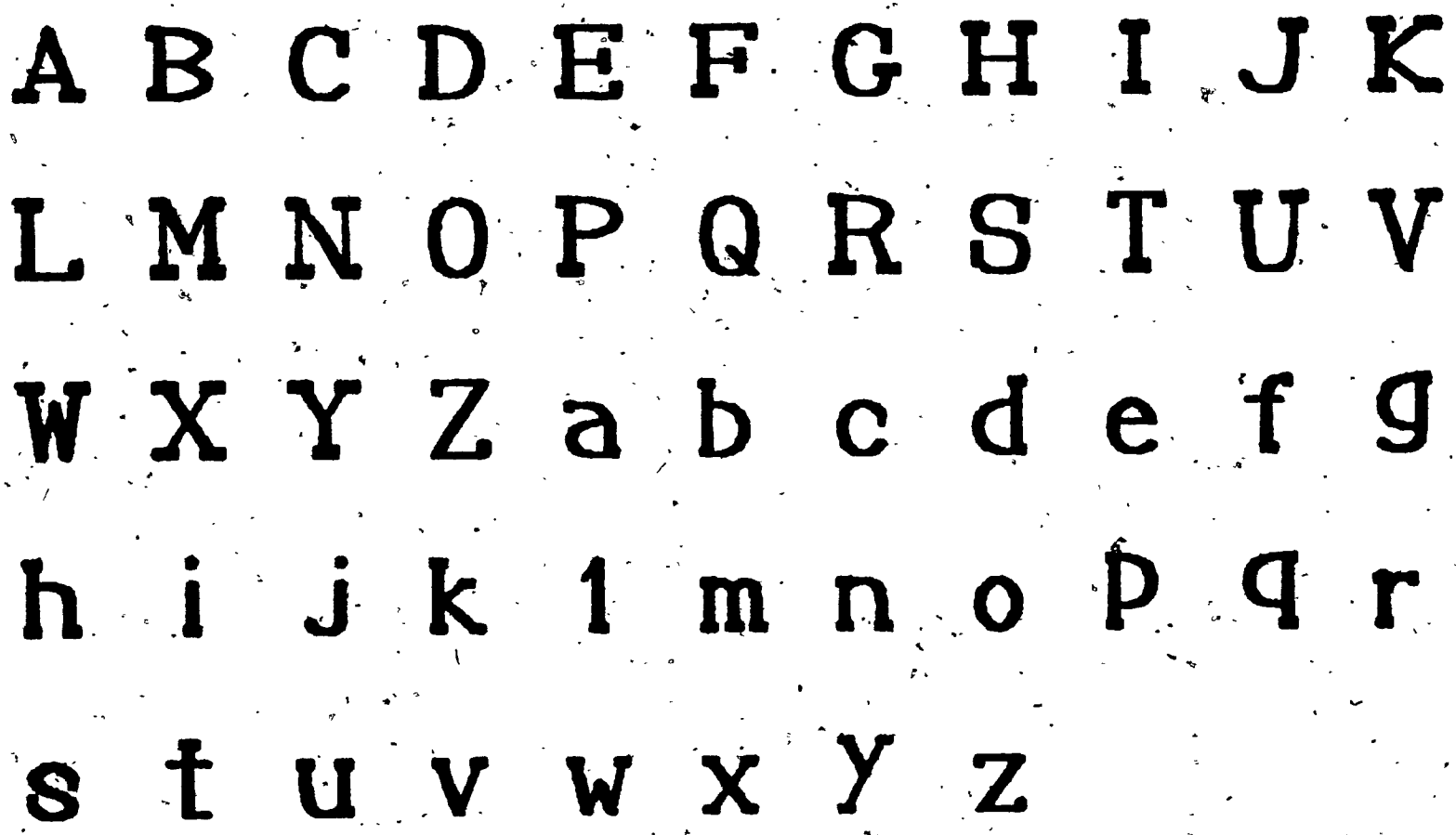


Figure 5.9 Type Style 5 with slab serifs.

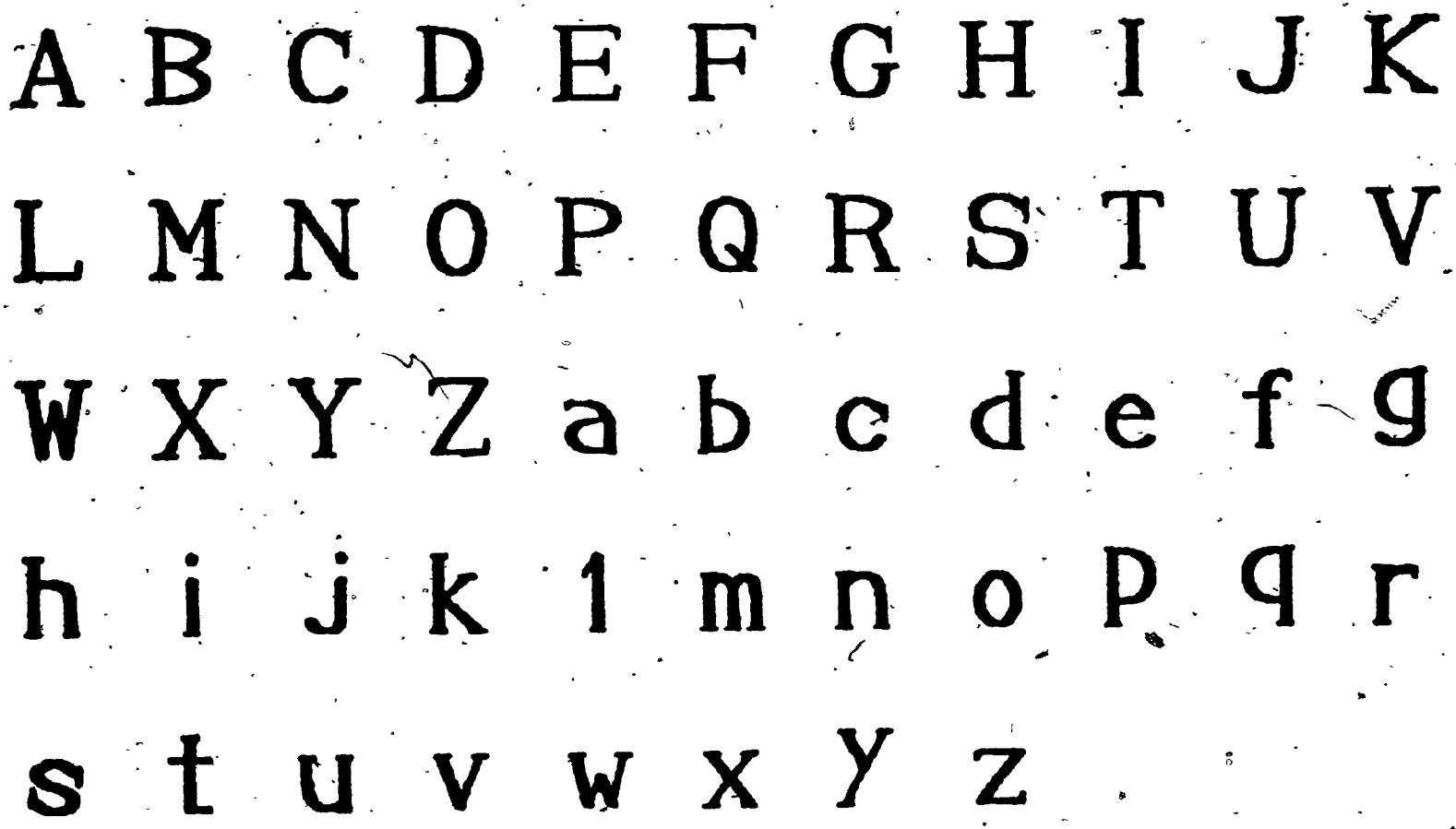


Figure 5.10 Type Style 5 with bracket serifs.

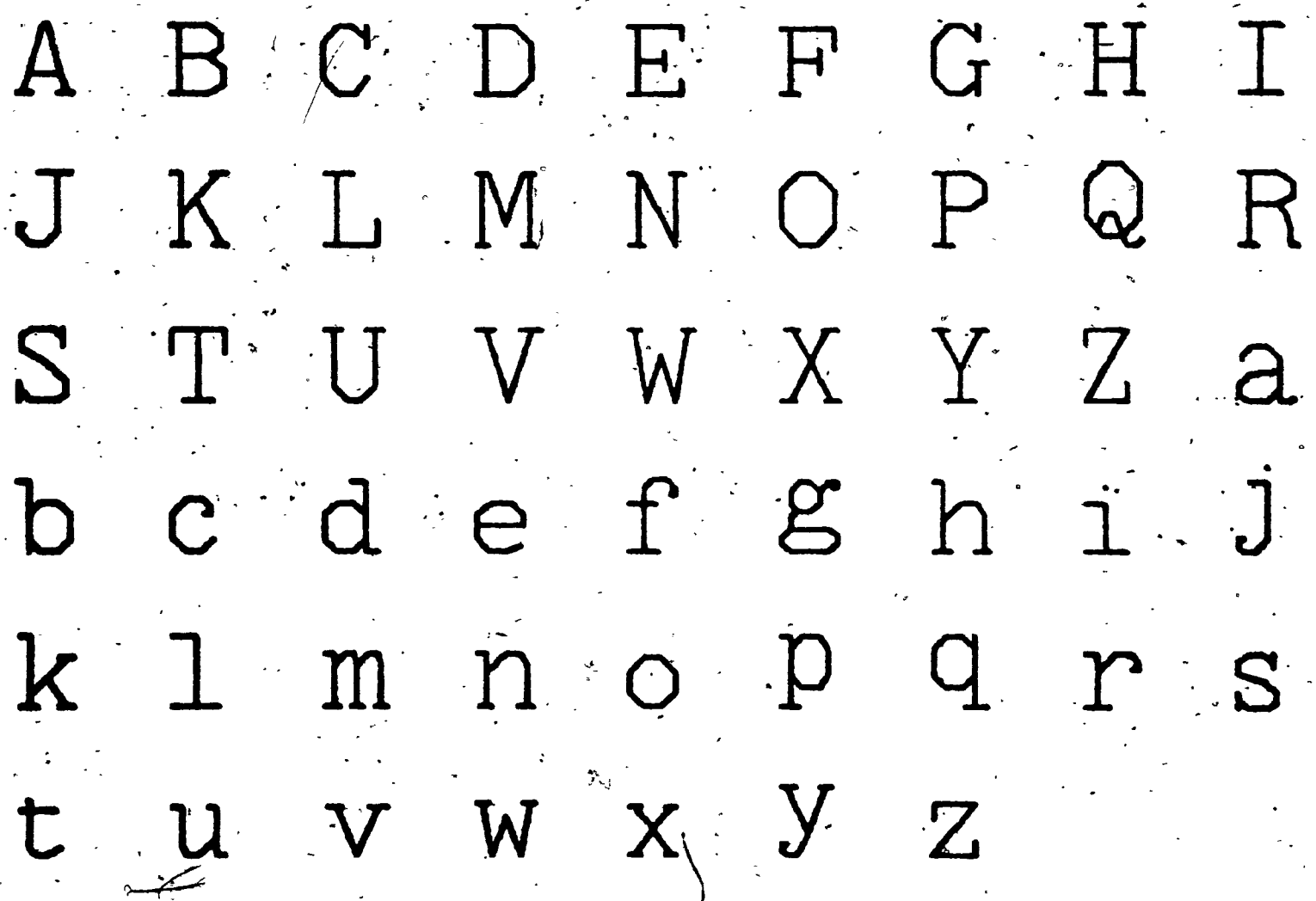


Figure 5.11 Pica in Type Style 1.

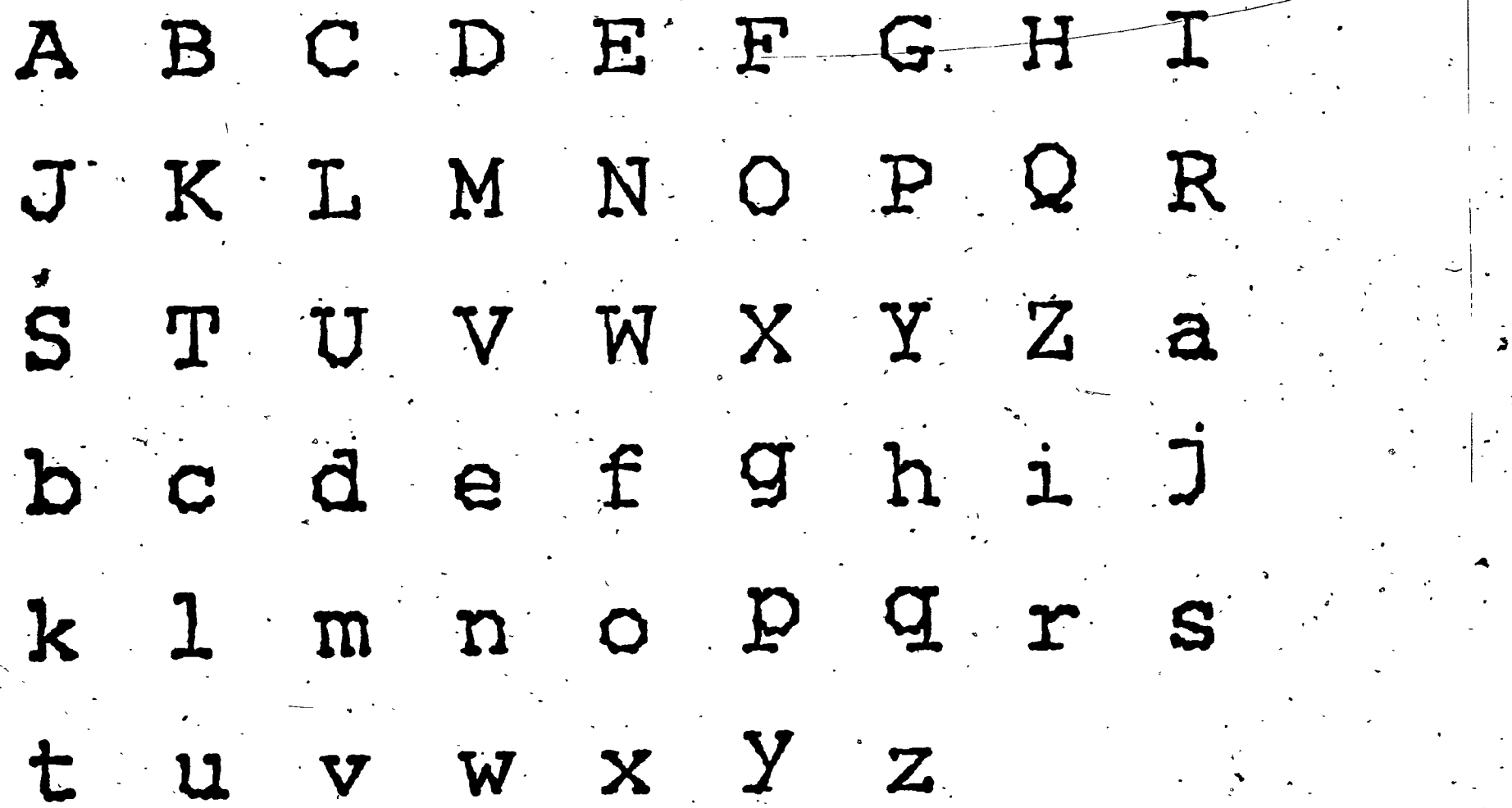


Figure 5.12 Courier in Type Style 3.

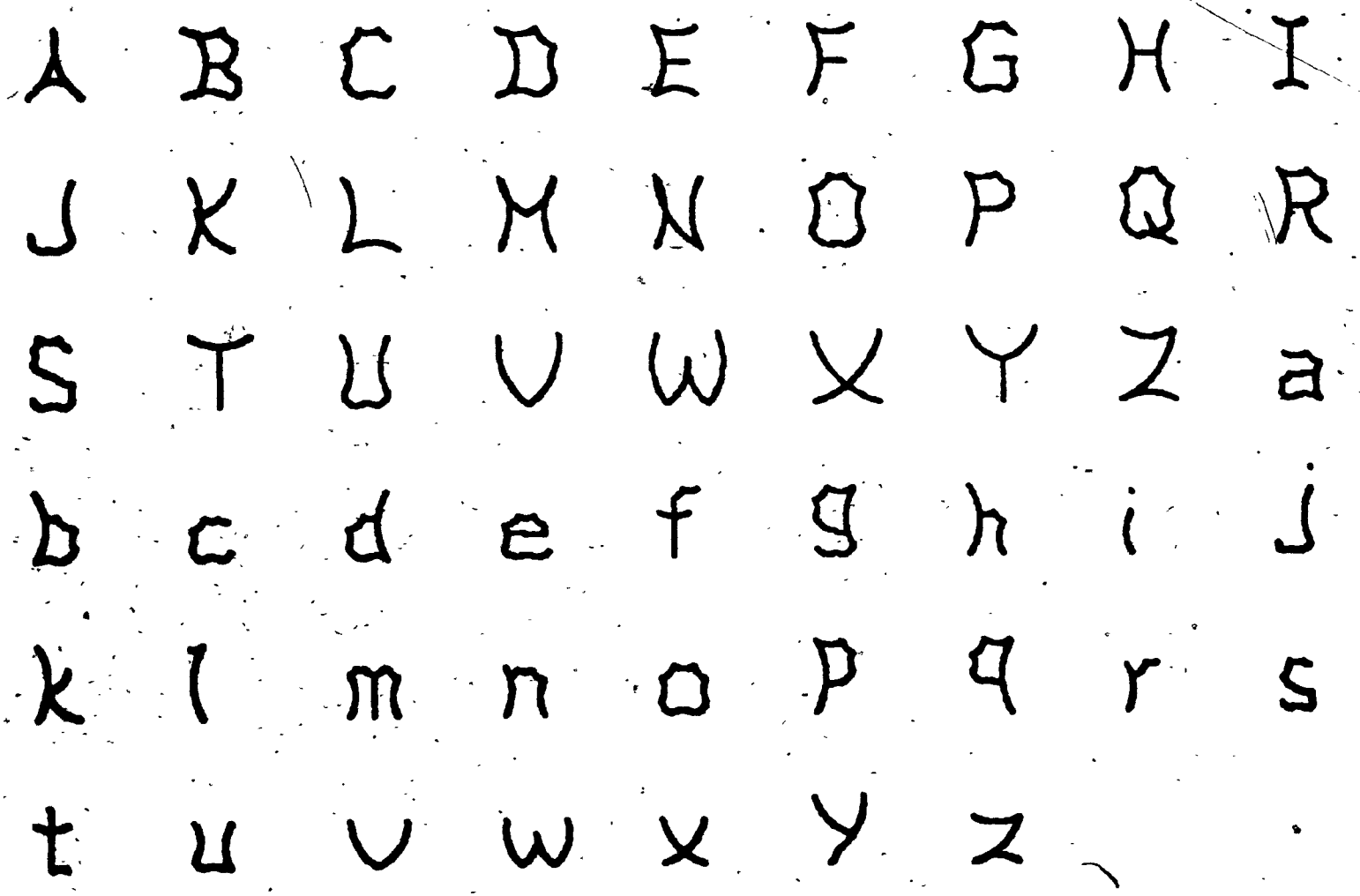


Figure 5.13 Orator in Type Style 4.

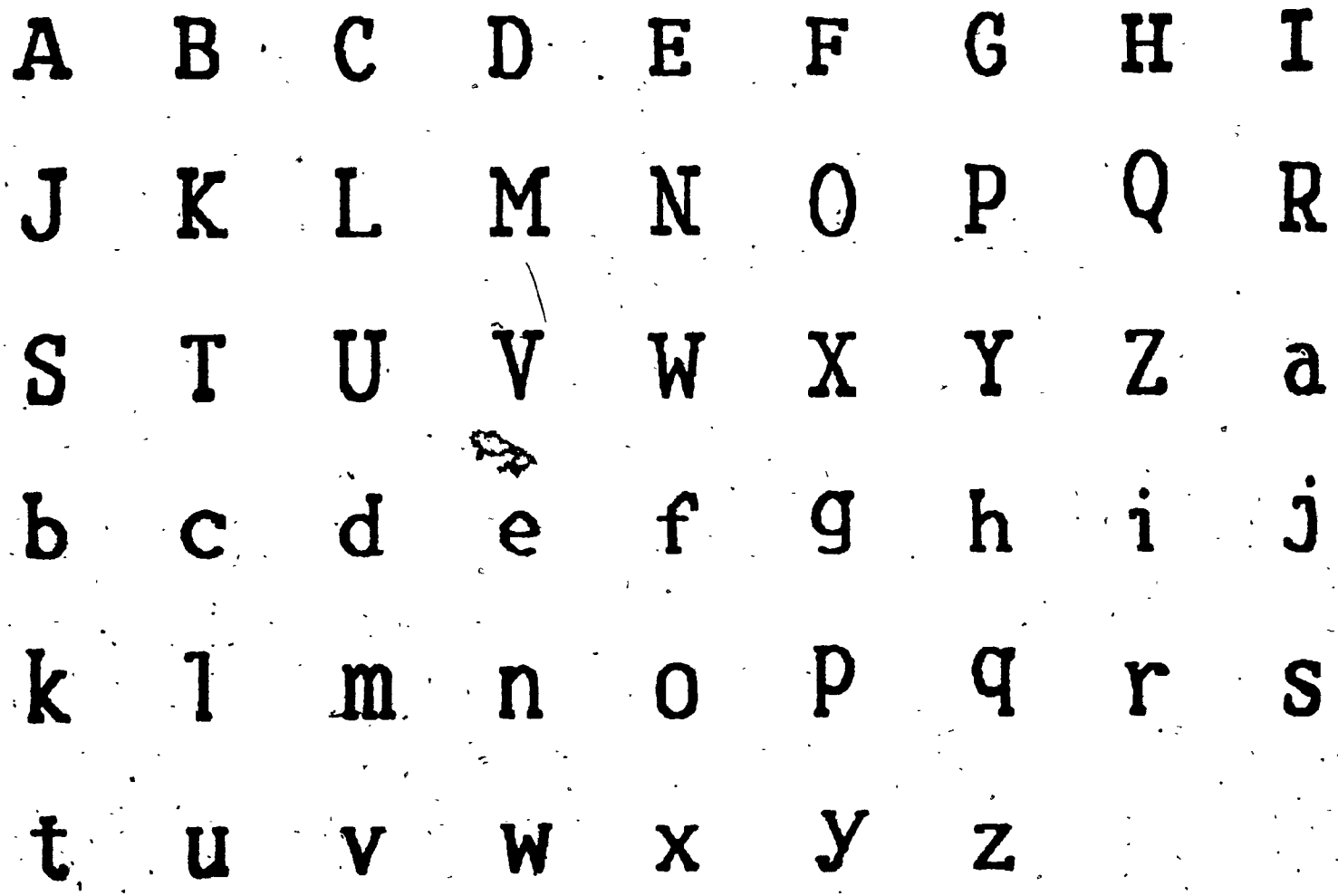


Figure 5.14 Gothic in Type Style 5 with slab serifs.

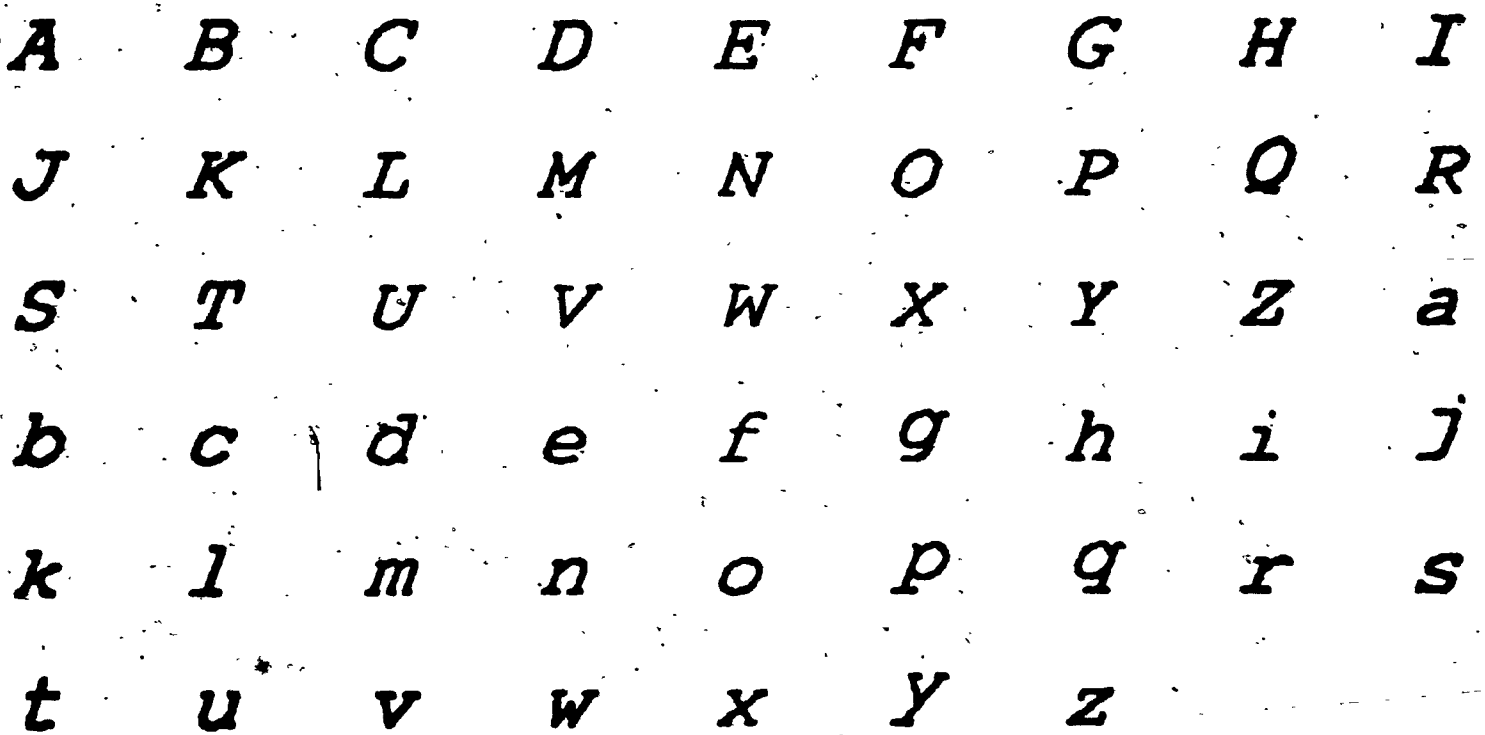


Figure 5.15 Horizontal orientation to the right.

Figure 5.16 Horizontal orientation to the left.

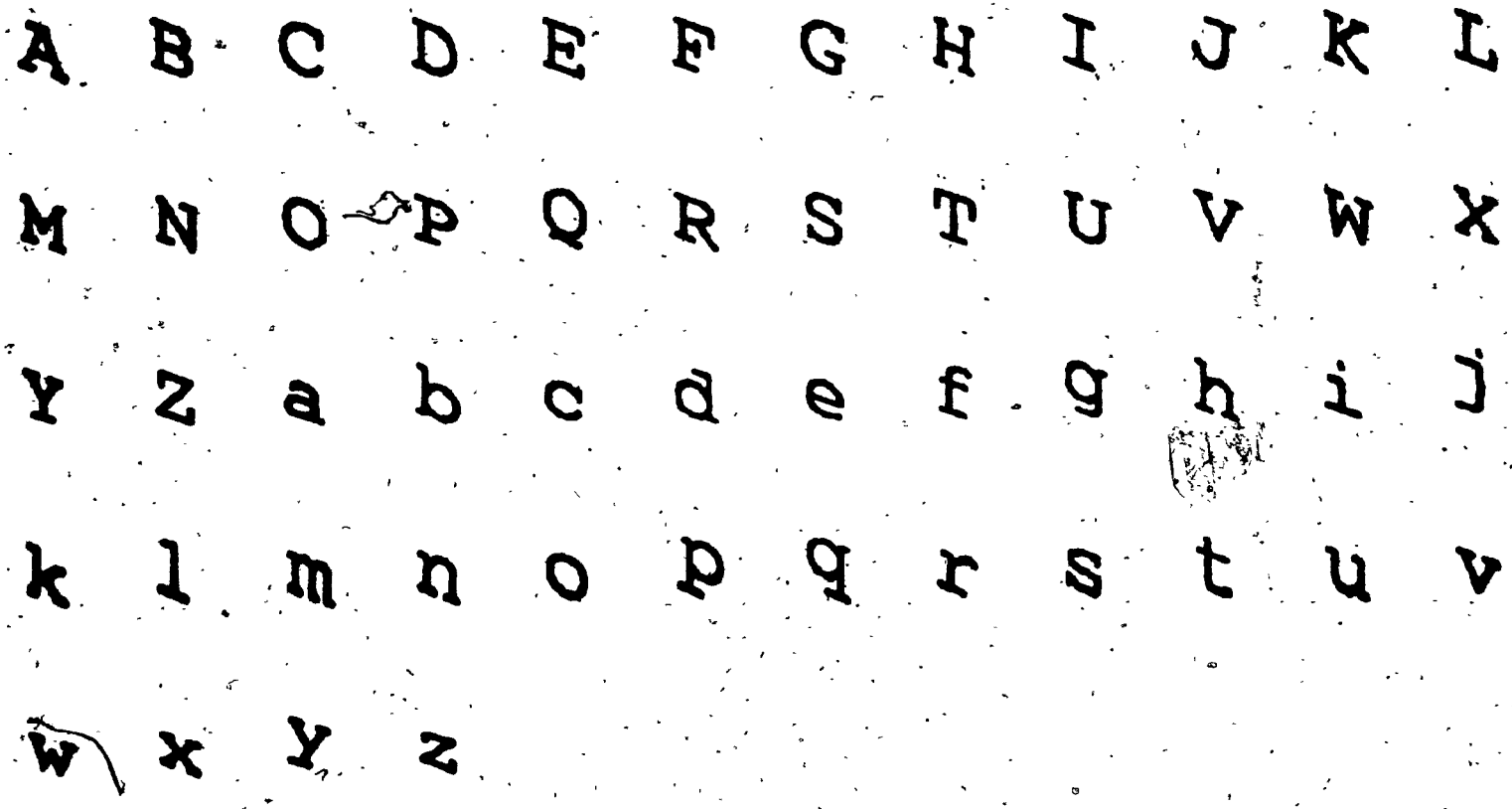


Figure 5.17 Vertical orientation to the top.

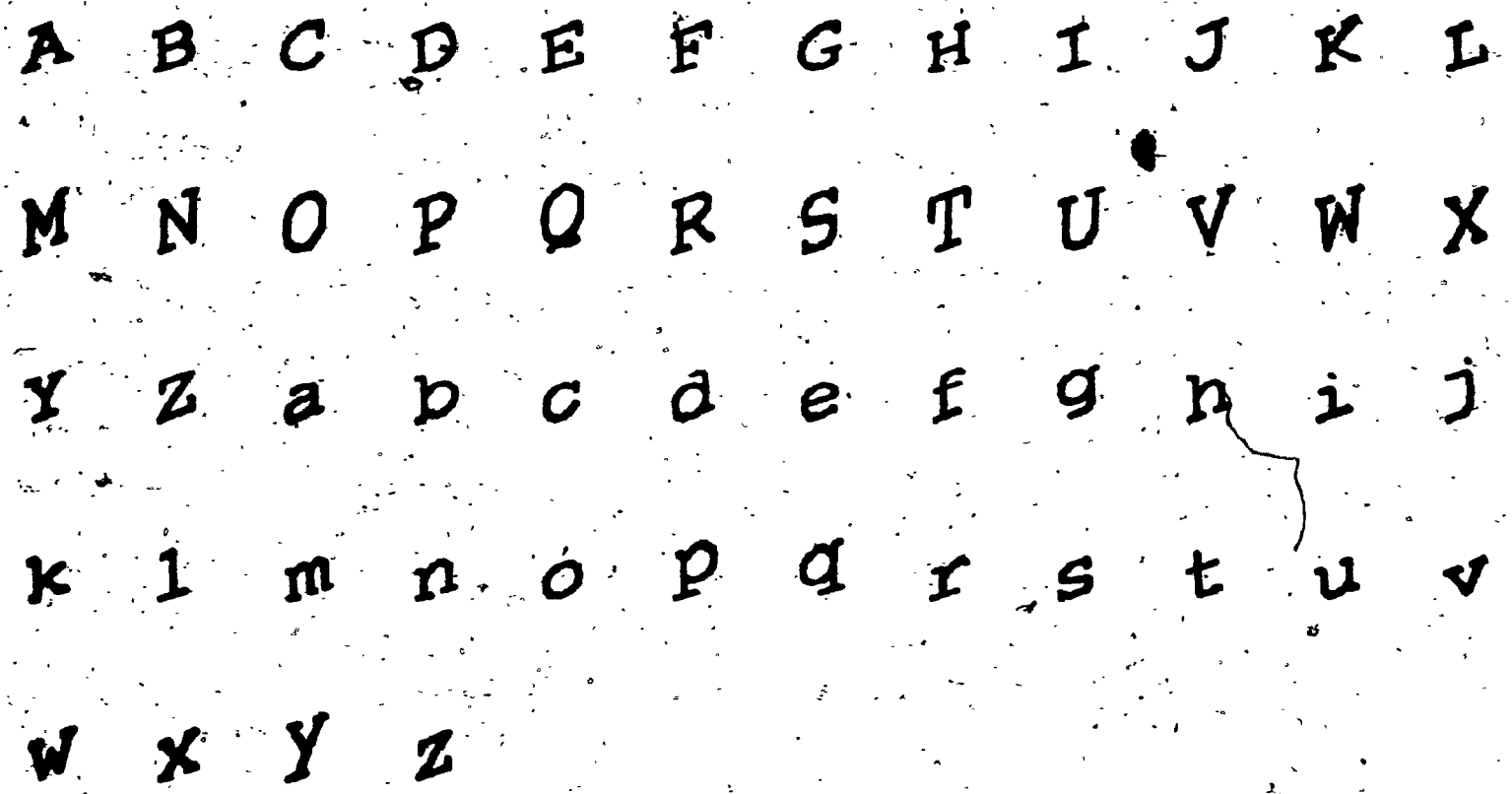


Figure 5.18 Vertical orientation to the bottom.

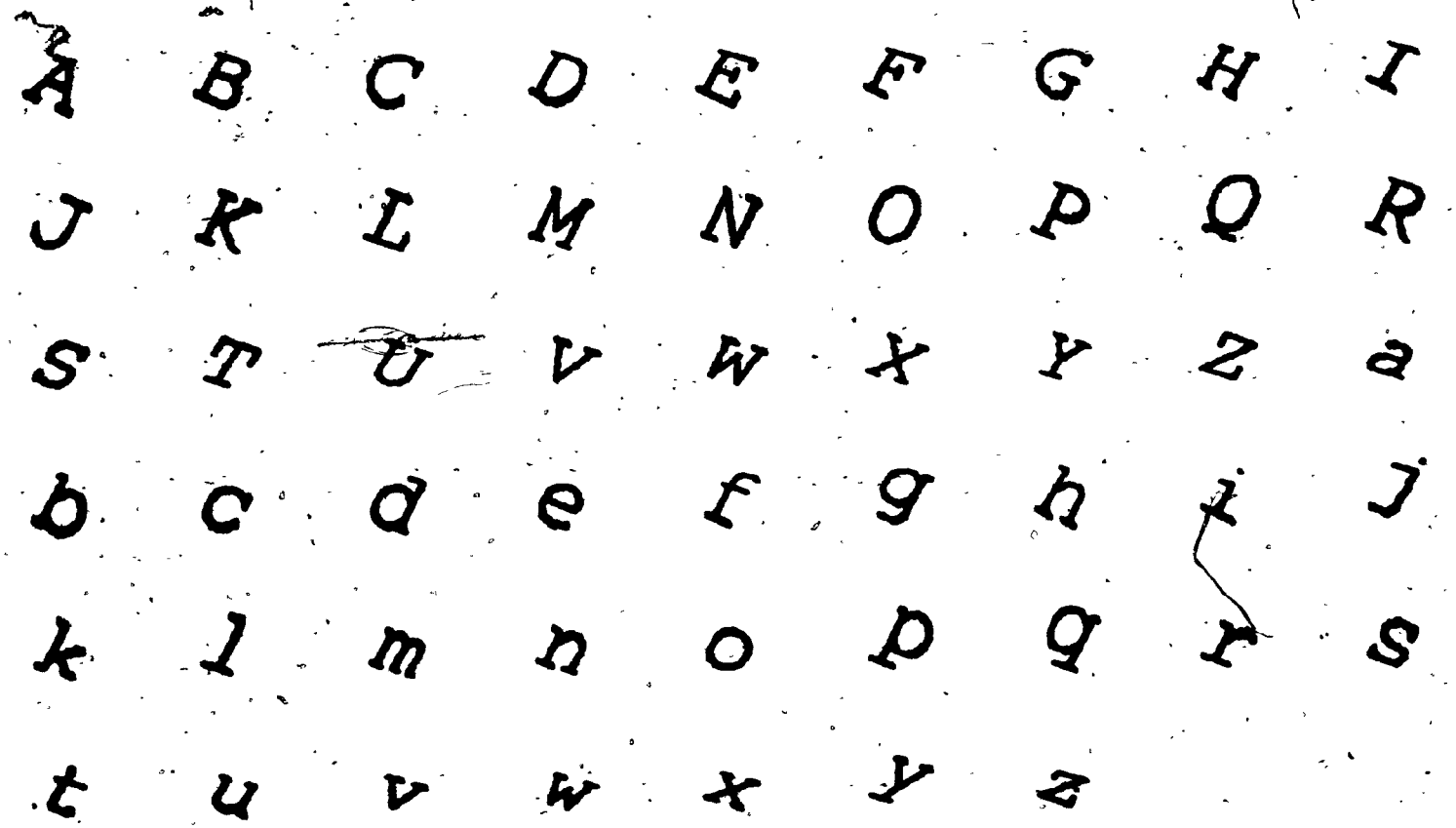


Figure 5.19 Vertical (to the top) and horizontal (to the left) orientations.

Chapter 6

Conclusions and Discussions

6.1 Conclusions

In this thesis a system for digital type design has been proposed. The system consists of two main processes.

The first process, the encoding or description process, was described in chapter 3 and performed by the Picture Descriptor (P.D.). This process represents the learning phase of the system, and its purpose was to capture the spirit or characteristics of a character font. Input digitized character representations in the form of binary (black and white) matrices were transformed into graph representations in the form of edges and vertices. This was achieved by developing a variety of algorithms, including: smoothing, thinning, point labelling and reduction, and graph traversal. Possible code inaccuracy adjustments or new font definitions were performed either in an interactive manner or manually.

The second process, the decoding or generation process,

was described in chapter 4 and performed by the Character Generator (C.G.). This process represents the generation phase of the system, and its purpose was to generate different type styles (a total of five) from a single font definition (derived from the first process). This was achieved by applying some transformational operations or rules (on the code) depending on the specific type style. Different types of pens and serifs in different sizes were used. Family font variations in size, boldening and orientation have been accomplished. Tests were conducted on standard or non-standard character fonts. Experimental results show the effectiveness of the proposed method.

6.2 Contributions and advantages of the proposed method

The work of this thesis encompasses many interdisciplinary fields, including: Pattern Recognition, Image Processing, Computer Graphics, Type Design, etc. The contributions of the thesis are centered in both its breadth and its depth; breadth, in the sense that the overall system is new; depth, in the sense that some new ideas have been explored, including:

i) The presentation of some algorithms during code derivation, i.e. the algorithms for pictorial information extraction, point labelling and reduction,

etc.

ii) The code representation in the form of graphs (represented in linked lists) is new

iii) Different Type Styles (their derivation and rules), serifs (their derivation and placements rules), and family font variations have been achieved.

The proposed method differs sharply from other past methods in its emphasis on human effectiveness as the most important design criterion. Some of the advantages are:

a) The overall structure of the system models the human communication process. The Picture Descriptor (encoder) and the Character Generator (decoder) are the basic units in a communication system

b) The code derivation process was automated. In most (if not all) of the past approaches, manual operations were performed to derive the code, thus considerable time and effort was required.

c) Contrary to the other approaches, the code inherits the basic structure of a pattern, and as a result it can be easily understood by the user. This has the following positive consequences:

   i) The user is no longer required to be familiar with curve-fitting methods or any other advanced mathematical code representation

   ii) Possible code inaccuracy adjustments or new

definitions (of any pattern) can be performed even manually without requiring considerable time and effort

d) The generation process imitates the human hand. Characters were generated by pens whose centers passed though their skeletons. Pens (or erasers) were also used in [Knu79].

e) The method provides great flexibility. Given a single character definiton (i.e. its code), different styles (of that character) in different sizes, boldenings and orientations can be obtained.

Furthermore, we strongly believe that because of the code representation and the simple operations performed during the generation process, the derivation and generation time, and the coding size are compatible with the other approaches. However, in most cases, this information was not given. When given, in some cases, the comparison is difficult, because, for example, the time depends on the computer under consideration, or the design criteria and achievements of the various approaches were different.

6.3 Applications

The wide applicability of the proposed system includes

the following fields:

a) Printing industry and Text Processing applications. The system can be used in conjunction with a typesetting system (i.e. TEX, see [Knu79], see also [Ker74]) for book production, newspaper editorial material, advertisements, etc. In such applications the multiplicity of different character fonts in different variations is required

b) Type Design. The system can be used as a tool to assist type designers. He/she can quickly and easily define and manipulate the typefaces based on his/her concepts. At the same time, it saves time-consuming manual drawing work

c) Teaching. The system can also be used as an aid to teach type design. This will allow us to get deeper knowledge about letterforms, and subsequently the artistic level will be raised even higher. Furthermore, the system can be used not only for typefaces, but for any other graphical pattern

d) Computer-aided prostheses for the handicapped. The system provides great assistance to the visually impaired in the interactive use of computers [Gli84]. Furthermore, it may relieve problems (such as eye fatigue, back pain, etc.) commonly associated with prolonged sessions at the computer terminal. In such applications, the user may select the appropriate

character size, boldening, etc. to suit his/her needs

e) Pattern Recognition. The representation of a character as a graph can be used in the context of Pattern Recognition. For example, a classification scheme can be easily devised, based on the graphical structure of the letters [Har73]

f) Digitization. The system provides a new way for digitizing typefaces (or any other pattern) or reproduction of old ones

g) Font scaling. Sometimes, it is required to reduce or magnify a font within a machine, or alternatively, to reproduce the same character on another printing device having a different resolution. The font scaling problem can be easily solved (by default) by adjusting the appropriate sizes of the viewport and/or pen

h) Palaeography. Alphabets have been undergoing significant changes from their beginning. Someone, then, has to determine what kinds of operations need to be performed to graphs (representing the alphabets) so that the development of alphabets could be traced through their various stages [Bon72]. This type of analysis that could be an extension to our system, will certainly help the palaeographer to make up his/her theory

i) Psychology. Using the system, experimental tests could be conducted on different character fonts by psychologists to investigate their legibility, i.e. good reading conditions [Spe69]

j) The system can be also used for image transmission and data compression

k) Some of the methods described, apply equally well to other related fields in computer vision.

6.4 Suggestions for further research

Some suggestions are listed below for both further development of the system (to ensure its reliability) and generally further research in this area:

a) The output (generated characters) could be used as input to a typesetting system for a manuscript preparation

b) Tests should be extended to other type styles related to a given font, other character fonts, characters belonging to other languages, 3-dimensional characters or any other graphical pattern

c) Studies could be carried on by those interested in the fields of Pattern Recognition, Palaeography or Psychology. See section 6.3, applications d), g), and h)

d) Consistency among characters was not taken into consideration. To achieve this, the function of the Picture Descriptor should be either extended or properly modified. Then, the data structure to represent the characters would be hierarchical. At the lower level

would be the items (or primitives) shared among the characters. At the higher level would be a list of items (and perhaps some relevant information) that compose the characters. This type of extention would certainly reduce the coding size, but the derivation and perhaps the generation time would be increased

e) A promising effort would be the development of a special device for graphics arts. Perhaps, a combination of hardware and software units. The hardware unit could be equipped with different device primitives to generate automatically, for example, strokes (straight, curved), pens, serifs, etc. Furthermore, other types of devices such as stylus or light pen, high resolution tablet and screen, etc. would be included. The software unit would be capable of selecting (and keeping) different types of knowledge about characters such as: structural (i.e. description of characters), linguistic (i.e. consistency among characters in a given font and/or among fonts), typographic (i.e. artistic talents), etc.

Glossary

Calligraphy: The art of fine handwriting.

Characters: Generally, characters are the individual letters (both upper and lower case), numerals, punctuation marks, figures, etc. of the alphabet. In this thesis, characters are only the upper and lower case letters and are treated as 2-dimensional pictures.

Face: The actual printing surface of a piece of type. Also, the style or design of the type: typeface.

Font: A complete assembly of all the characters of one size and style.

Printing: A permanent, graphic and visual medium that encompasses all those ideas that result in methods or devices to manipulate or mupliply graphic visual messages.

Sanserif: A style of type which has no serifs.

Serif: The finished-off strokes of a letter.

Type: All the characters used singly or collectively to create words, sentences, block of text, etc.

Typesetting: The assembling of typographic material suitable for printing or incorporating into a printing plate.

Typography: The art, or skill of designing communication by means of the printed world.

Bibliography

[Ami79] H. Amiri, V. Margner and P. Zamperoni, "A method for analysing and synthesizing line drawings", Signal Processing, Vol. 1, 1979. pp. 5-13.

[And71] D. M. Anderson, "Cresci and His Capital Alphabets", Visible Language, V 4, 1971, pp. 331-352.

[Bau82] F. Baudin, et al, "Other Replies to Donald Knuth's article, 'The Concept of a Meta-Font' (Letters to the Editor)", Visible Language, XVI 4, 1982, pp. 339-359.

[Bel81] A. Bel-lan and L. Montoto, "A thinning transform for digital images", Signal Processing, Vol. 3, 1981, pp. 37-47.

[Beu73] M. Beun, "A flexible method for automatic reading of handwritten numerals", Philips Tech. Rev., Vol. 33, 1973, pp. 89-101 and 130-137.

[Big83] C. Bigelow and D. Day, "Digital Typography", Scientific American, Vol. 249, August 1983, pp. 106-119.

[Bon72] J. A. Bondy, "The 'Graph Theory' of the Greek

Alphabet", Graph Theory and Applications, Y. Alavi et al (eds.), Springer Verlag, Berlin, 1972, pp. 43-54.

[Chi80] P. Chinnuswamy and S. G. Krishnamoorthy, "Recognition of handprinted Tamil characters", Pattern Recognition, Vol. 12, 1980, pp. 141-152.

[Cou75] P. Coueignoux, "Generation of Roman Printed Fonts", Ph.D. Thesis, MIT, Dept. Elec. Eng., June 1975.

[Cou81] P. Coueignoux, "Character Generation by Computer", Computer Graphics and Image Processing, Vol. 16, 1981, pp. 240-269.

[Cox74] C. H. Cox, B. Blesser and M. Eden, "The application of type font analysis to automatic character recognition", Proc. 2nd Int. Joint Conf. Pattern Recognition, Copenhagen, August 1974, pp. 226-232.

[Cox76] C. H. Cox and P. Coueignoux, "Concise letter/type font description: Theory and computer implementation", TAGA Conf. Proc., Philadelphia, Pa., May 1976, pp. 331-355.

[Cox82] C. H. Cox, P. Coueignoux, B. Blesser and M. Eden, "Skeletons: A link between theoretical and physical letter description", Pattern Recognition, Vol. 15, 1982, pp. 11-22.

[Flo84] J. Flowers, "Digital Type Manufacture: An Interactive Approach", Computer, Vol. 17, May 1984, pp. 40-48.

[Fra71] A. J. Frank, "Parametric font and image definition and generation", Fall Joint Computer Conf., 1971, pp. 135-144.

[Gas76] P. Gaskell, "A Nomenclature for the Letterforms of Roman Type", Visible Language, X 1, 1976, pp. 41-51.

[Gli84] E. P. Glinert and R. E. Ladner, "A large font virtual terminal interface: A Software Prosthesis for the Visually Impaired", Communications of the ACM, Vol. 27, No. 6, June 1984, pp. 567-572.

[Har73] F. Harary, "Typographs", Visible Language, VII 3, 1973, pp. 199-208.

[Har83] S. Harrington, "Computer Graphics: A programming Approach", McGraw-Hill, 1983.

[Hay30] M. W. Hayness, "The student's history of printing", McGraw-Hill Inc., New York, 1930.

[Hob82] J. Hobby and G. Gu, "Using Metafont to Design Chinese Characters", Proc. Int. Conf. Chinese Language Computer Society, Sept. 1982, pp. 18-36.

[Hoc73] J. Hochuli, "Book Review-Origin of the Serif", Visible Language, VII 1, 1973, pp. 73-91.

[Hof82] D. R. Hofstadter, "Metafont, Metamathematics, and Metaphysics: Comments on Donald Knuth's Article 'The Concept of a Meta-Font'", Visible Language, XVI 4, 1982, pp. 309-338.

[Hoh80] L. Hohenstein, "Computer peripherals for minicomputers, microprocessors, and personal computers", McGraw-Hill, New York, 1980.

[Jas70] W. P. Jaspert, W. T. Berry and A. F. Johnson, "The Encyclopedia of Type faces", Barnes and Noble Inc., New York, 1970.

[Ker74] B. W. Kernighan and L. L. Cherry, "A System for Typesetting Mathematics", Bell Laboratories, Computer Science Tech. Rep. 17, 1974.

[Kin79] D. Kindersley and N. Wiseman, "Computer-Aided Letter Design", Printing World, Oct. 1979, pp. 12,13,17.

[Knu79] D. E. Knuth, "TEX and METAFONT: New Directions in Typesetting", Digital Press and American Mathematical Society, 1979.

[Knu80] D. E. Knuth, "The Letter S", The Mathematical Intelligencer, Vol. 2, 1980, pp. 114-122.

[Knu82] D. E. Knuth, "The Concept of a Meta-Font", Visible Language, XVI 1, 1982, pp. 3-27.

[Mat67] M. V. Mathews, C. Lochbaum and J. A. Moss, "Three Fonts of Computers-drawn Letters", Journal of Typographic Research, Vol. 1, 1967, pp. 345-356.

[Mat65] M. V. Mathews and J. E. Miller, "Computer editing, typesetting and image generation", Proc. Fall Joint Computer Conf., 1965, pp. 389-398.

[Mei81] T. Y. Mei, "LCCD, A Language for Chinese Character Design", Software Practice and Experience, Vol. 11, 1981, pp. 1273-1292.

[Mei69] M. Meiss, "The First Alphabetical Treatises in the Renaissance", Journal of Typographic Research, Vol. 3, 1969, pp. 3-30.

[Mer68] H. W. Mergler and P. M. Vargo, "One Approach to Computer Assisted Letter Design", Journal of Typographic Research, Vol. 2, 1968, pp. 299-322.

[Mur77] P. J. Murdock, "New Alphabets and Symbols for Typesetting Mathematics", Notices Amer. Math. Soc., Vol. 24, 1977, pp. 63-67.

[Nis83] M. Nisenholtz, "Graphics Artistry On Line", Byte, Vol. 8, No. 7, July 1983, pp. 104-110.

[Pav82] T. Pavlidis, "Algorithms for Graphics and Image Processing", Computer Science Press, Rockville, MD, 1982.

[Pav83] T. Pavlidis, "Curve Fitting with Conic Splines", ACM Trans. Graphics, Vol. 2, No. 1, 1983, pp. 1-31.

[Pla83] M. Plass and M. Stone, "Curve-Fitting with piecewise Parametric Cubics", Computer Graphics, Vol. 17, No. 3, 1983, pp. 229-239.

[Rao76a] K. C. V. Rao and K. Black, "Type classification of fingerprints. A syntactic approach", Proc. 3rd Int. Joint Conf. Pattern Recognition, 1976, pp. 778-782.

[Rao76b] T. CH. M. Rao, "Feature extraction for fingerprint classification", Pattern Recognition, Vol. 8, 1976, pp. 181-192.

[Rob71] D. O. Robinson, M. Abbamonte and S. H. Evants, "Why Serifs are Important: The perception of small print", Journal of Typographic Research, Vol. 4, 1971, pp. 353-359.

[Spe69] H. Spencer, "The visible word", Royal College of Art, London, 1969.

[Udu75] K. J. Udupa and I. S. N. Murthy, "Some new concepts for encoding line patterns", Pattern Recognition, Vol. 7, 1975, pp. 225-233.

[Ung59] S. H. Unger, "Pattern Detection and Recognition", Proc. IRE, Vol. 47, 1959, pp. 1737-1752.

[Wal69] G. O. Walter, "Typesetting", Scientific American, Vol. 220, May 1969, pp. 60-69.