XV. COGNITIVE INFORMATION PROCESSING

Academic and Research Staff

Prof. Murray Eden
Prof. Jonathan Allen
Prof. Barry A. Blesser
Prof. Goesta H. Granlund*
Prof. Francis F. Lee
Prof. William F. Schreiber
Prof. William Simon†
Prof. Hoo-min D. Toong
Prof. Donald E. Troxel
Prof. Ian T. Young

Dr. Jack E. Bowie
Dr. Brian E. Boyle
Dr. Ralph W. Gerchberg
Dr. Uri F. Gronemann
Dr. Martin J. Hewlett
Dr. Conrad C. Jaffe
Dr. Dennis J. Kfoury
Dr. Murad Kunt‡
Dr. Perry L. Miller

Dr. David M. Ozonoff
Dr. Stephanie E. Sher
Dr. Robert J. Shillman
Dr. Armando Roy Yarza
Francis X. Carroll
Jeffrey A. Hagerich
M. Sharon Hunnicutt
Eric R. Jensen
Kenneth A. Okin
Kenneth P. Wacks

Graduate Students

Jack A. Allweis
Robert P. Bishop
Ronald E. Boucher
Becky J. Clark
Philippe Coueignoux
Charles H. Cox
David R. Cuddy
Caleb J. Drake
James R. Ellis, Jr.
Irvin S. Englander
Antonio C. Gellineau
Richard S. Goldhor

Samuel M. Goldwasser
Monson H. Hayes
Marvin E. Jernigan
Malik Khan
Theodore T. Kuklinski
Sze-Ping Kuo
Donald S. Levinstone
Charles W. Lynn
Lee C. Makowski
José L. Marroquin
Dennis V. Marsicano
Joseph T. Olesik

Douglas O'Shaughnessy
Douglas B. Paul
Roger S. Putnam
Robert E. Rouquette
Robert D. Solomon
Brian R. Stanley
Christopher E. Strangio
Anthony Sun
Joseph E. Wall, Jr.
Allen W. Wiegner
Tonny Y. Wong
Gregory W. Zack

A. AUDIO RESPONSE TERMINALS

JSEP

Joint Services Electronics Program (Contract DAAB07-74-C-0630)

Jonathan Allen

During this period, our efforts have concentrated on three major problems associated with the audio response system that we are designing. First, two all-digital vocal-tract model systems have been designed, each using different serial arithmetic techniques, and each requiring less than 100 TTL MSI circuits. These designs are being implemented, and after testing we plan to select a scheme that will be suitable for LSI implementation on one or two chips. We are also starting to design a processor that computes vocal-tract model parameters from a narrow phonetic description.

In order to control pitch and timing by rule, we have two major projects under way. A set of rules for deriving pitch contours is already complete, and is undergoing testing. Several experiments on duration have been completed, but no set of rules has

JSEP

*Visiting Associate Professor, Linkoeping University, Linkoeping, Sweden.
†Visiting Associate Professor, University of Rochester.
‡Visiting Scientist, Laboratoire de Traitement de Signaux, Lausanne, Switzerland.

JSEP

yet been distilled from the data. A complete set of pitch and timing rules will allow us to modify stored segmental data for words in order to synthesize flexibly a wide variety of sentences.

An experiment has been performed to determine changes induced on the segments on either side of a word boundary. Data have been collected, and are now being analyzed.

B. FONT-INDEPENDENT CHARACTER RECOGNITION

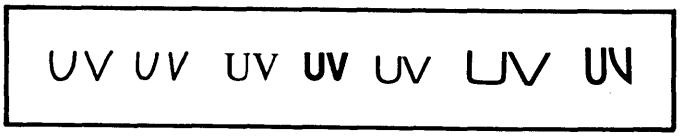
Joint Services Electronics Program (Contract DAAB07-74-C-0630)
National Science Foundation (Grant GK-33736X2)

Murray Eden, Robert J. Shillman, Charles H. Cox

This report summarizes three studies related to font-independent character recognition. In the first study we investigated the essence of U-ness and V-ness, the properties that characterize the letters U and V. In the second study we extended previous findings on idealized thin-lined handprinted characters to characters composed of thick strokes such as those found in machine-printed characters. In the third project we studied the apparent consistency within a type font and sought to define a set of primitives out of which every machine-printed character can be composed.

1. Investigation of U-V Discrimination

The goal of this study was to determine the features that all printed letters U have in common which distinguish them from all printed letters V. Figure XVI-1 illustrates the variety of U and V found in hand-printing and machine-printing. In general, the U have parallel arms that are smoothly joined (i.e., continuous first derivatives), whereas the V have antiparallel arms that intersect at a point. The last character in Fig. XV-1 has parallel arms and also a discontinuous derivative at the bottom; in the absence of any contextual information, the character's identity is ambiguous; it is as likely to be labeled U as V. It is interesting to note that the character is a U in Desdemona Solid, whereas this same character is a V in Busorama Bold. Psychophysical experiments were conducted to investigate the interaction of the two attributes (pointedness of a character's base and parallelness of a character's sides) on character



JSEP

Fig. XV-1. A variety of letterforms for U and V.

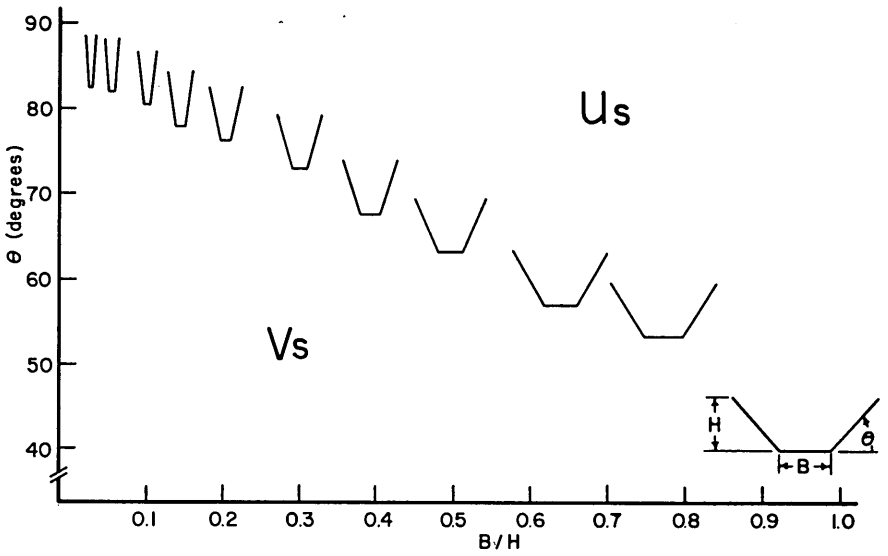


Fig. XV-2. The U-V boundary.

identity. The results, shown in Fig. XV-2, indicate that both angularity and pointedness (defined as the base of the character scaled by the character's height) are features in U-V discrimination. The U-V boundary is given by $\theta + 40.7(B/H) = 87.9$; thus for $\theta + 40.7(B/H) < 87.9$ characters are more likely to be called V than U and for $\theta + 40.7(B/H) > 87.9$ characters are more likely to be called U than V.

2. Recognition of Thick-Stroke Characters

The functional attribute LEG is the attribute that distinguishes Y from V, F from C, P from D, H from U, and A from O. In a previous study,[1] a rule for determining the state of the functional attribute LEG was experimentally obtained by using characters drawn with fine lines approximately 1/100" thick. In this study, we investigated characters with stroke widths of 1/10", 1/8", and 1/6". For all 15 characters investigated, a simple transformation was experimentally observed which maps each unknown "thick" character to a corresponding skeleton whose letter label can be directly obtained by using the rule given by Shillman.[1]

3. A Model for Type Font Description

This study investigated the apparent consistency among roman printed type fonts. A generative model based on a linguistic study of the consistency of roman printed characters was derived. Characters can be decomposed into primitives. To represent a letter by a printed character, we can use a specific combination of primitives. A grammar was derived that governs these combinations within the roman style. The repeated use of the same drawing to represent a primitive in more than one combination

JSEP differentiates the characters of a font from other fonts; furthermore, parameters are used to specify the drawings, and relationships exist among the parameters for the different drawings of a font. Related parameters are gathered into families, and global transformations on each of the families can describe elementary operations on fonts such as boldening, size variations, and so forth.[2]

The model for character formation given by Coueignoux[2] is relevant to automatic character recognition. This model is discussed more fully in Section XV-C.

References

1. R. J. Shillman, "Character Recognition Based on Phenomenological Attributes: Theory and Methods," Ph.D. Thesis, Department of Electrical Engineering, M. I. T., 1974 (unpublished).
2. P. Coueignoux, "Generation of Roman Printed Fonts," Ph.D. Thesis, Department of Electrical Engineering, M. I. T., 1975 (unpublished).

C. A PARAMETRIC REPRESENTATION OF ROMAN PRINTED FONTS

Joint Services Electronics Program (Contract DAAB07-74-C-0630)

Philippe Coueignoux

[This work was supported by a scholarship from the Institut de Recherche d'Informatique et d'Automatique, and the Ecole Nationale Superieure des Mines de Saint-Etienne (France) with which the author is currently associated.]

1. Introduction

The shapes of roman printed fonts offer a compromise between geometrical simplicity and the complexity of shapes found in nature or in art.

A theoretic model has been built to incorporate basic letter shapes and their variations. A computer program has been written and tested that generates real characters from the theoretic code. The work would be completed by a program for deriving the parameters of the code from real data, in a way that achieves recognition of the letter name as quickly as possible. Practical uses of such a study could be creation of new systems to invent or digitize type fonts without scanning analog drawings, and obtaining better algorithms for multifont automatic readers.

This work represents the conjunction of two trends. One is concerned with models that explain the shapes of the roman alphabet, the other with the generation of type fonts by computer.[1-3] Both have received attention in the past, although the latter is necessarily a youthful offspring of modern technology. The former, on the contrary, may be traced back to the Renaissance.[4-6] At that time the search was for a mathematical representation of ideal value, whose rules should become imperative, whereas the present program is considered as a means, of no intrinsic justification, of providing the

type designer with a maximum of freedom. The modern trend, computer generation, should be distinguished from what is called computer encoding. In the latter the goal is to provide a digital code, meaningless to the type designer, which has the power to represent characters in a compact format for easy reconstruction[7-10]. On the contrary, computerized type generation requires the digital code to be understandable, so as to be conceived, input, and modified by a type designer who has no knowledge of computers. No attempt in this direction was known to the author until the program had been completed. It is all the more striking, therefore, to see how close the program comes to being a natural continuation of work done in 1968 by Mergler and Vargo at Case Western Reserve University.[1] Their treatment of 24 uppercase letters, notwithstanding certain simplifications of shapes for the sake of economy, embodied the same line of thought that was at the origin of the present work, and demonstrated its practicality for computer generation of type fonts.

JSEP

2. Model to Describe Type Fonts

Our goal is to formalize the apparent consistency among roman printed type fonts. Because these data have an artistic quality, no model can be proved to fit them. Yet to determine a set of rules that are likely to be followed by a type designer provides a first approximation to trying to match the reality.

A character is defined here as any graphical drawing that can be recognized by human beings as one member of the roman alphabet. It is restricted to the set of printed characters for 52 ordinary uppercase and lowercase letters, to the exclusion of ligatures, accents, digits, and signs. Under such constraints, a character bears both the name of the letter recognized by the human reader, and the name given by the type designer to the whole alphabet from which the character is drawn. By name of a type font, we mean the complete specification, for instance, Garamond No. 3 linotype in 10 points. From this follows the idea of breaking down the problem of modeling consistency among characters into consistency among all characters sharing the same letter name, as well as consistency among all characters belonging to the same type font. Such a premise is exemplified in Fig. XV-3.

Looking along the vertical lines of Fig. XV-4, it seems reasonable to represent a character as being formed by basic building blocks. Thus the uppercase E is made of a stem, two arms, and a nose in a certain spatial relationship. Such blocks are called primitives. The following invariance explains the importance of this decomposition.

Observation 1: The way primitives are used to form a character depends only on the letter and not on the name of the type font. We are not concerned with the actual drawing of the letters as characters from primitives, and we shall not consider the implicit meaning that could be given to "primitives" as strokes ordered in time; given two-dimensional still-life forms, the emphasis is rather on matching.

JSEP

JSEP

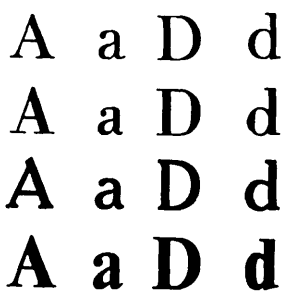


Fig. XV-3.

Consistency among characters by letter name and type font.

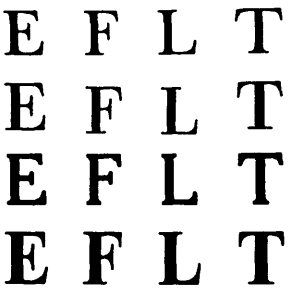


Fig. XV-4.

Letters and type fonts in terms of "basic building blocks" or primitives.

The use of primitives becomes evident as we look along the horizontal lines of Fig. XV-4. Within the same type font, the stem that is part of the uppercase E appears identical to the ones in L, F, and T; the arm is almost identical in those letters too; the nose is common to E and F. Hence we can make the next two observations.

Observation 2: A primitive may occur within the specifications of more than one letter.

Observation 3: The way a primitive is drawn is very stable from one occurrence to another within the same type font.

The physical shape of a primitive may be described uniquely by certain parameters. Those parameters should specify the likeness among the occurrences of a primitive within a type font, in the same way as primitives carry the likeness of the characters bearing the same letter name. And, as looking at all characters of the same type font is a method of enumerating the different primitives needed to cover all letter shapes, so looking at an occurrence of a primitive in all type fonts allows the identification of the parameters for this primitive, so that changes in value of these parameters will account for variation in shapes of the primitive from font to font. Observation 3 may then be restated.

Observation 3A: The values that specify the shape of a primitive from occurrence to occurrence within the same type font are very stable, independently of the letter.

Letters and type fonts were distinguished from characters. But both letters and type fonts are now described in terms of primitives whose shapes are specified by parameters. At this level of detail we can hope to regenerate the characters. Still further simplicity, or structural consistency, can be reached by going back from parameters to characters.

The characters have been replaced by parameters. A partial list is given in Table XV-1. The parameters are classified in accordance with the primitives to which they belong. The same names are used repeatedly; it is natural to gather the parameters

JSEP

Table XV-1. Partial table of parameters.

JSEP

stem height	majuscule bowl height	ascender height	body stem height
stem thickness	majuscule bowl thick thickness	ascender thickness	body stem thickness
stem serif thickness	majuscule bowl thin thickness		

along a line into a family. There are families of height and of thickness (thick or thin, etc.). Again, some invariance justifies this grouping.

Observation 4: There are definite relationships among the values of most of the parameters of one family for a given type font, and these relationships are independent of the type font. As examples, derived from measurements on 15 different fonts, in larger point sizes: height of O = 1.04 ± 0.04 height of I; total horizontal extension of m = 1.40 ± 0.10 total horizontal extension of O. These rules are called rules of proportion, and their application has been discussed by Rosenblum.[2]

Thus far two primitives that are not almost identical in some pair of occurrences within the same font could not have the same name, since this would violate observation 3. In any uppercase V, however, it seems that, apart from thickness and slope, the primitive on the right is similar in shape to the primitive on the left. Both resemble the stem of the uppercase I; that is, the same parameters describe them all, but two parameters, thickness and slope, take on different values. This calls for gathering primitives with the same parametric description, and hence similar shape except for the values of some parameters; for instance, the straight stroke or the half-bowl families. As an implicit consequence of the definition of primitives and parameters we make the following observation.

Observation 5: This partition of the set of primitives is independent of the type font. Within a primitive family, each common parameter may take on several values that satisfy observation 4. But all combinations of a possible value for each common parameter may not occur; for example, there is no straight stroke that will be thin and bent on the left. Such relations of mutual exclusion can be described by a set of rules called rules of disposition.

Primitives are put together eventually to form a letter. Because there is a finite number of letters, there is a finite set of spatial combinations, and not nearly all possible combinations are ever used. There is a second set of rules of disposition that represents the unary relation of existence on the set of all possible combinations. These rules are much simpler than the canonical description of the relations that they represent by enumeration. These rules of disposition have been studied[3] for the upper case and will not receive further treatment here.

JSEP

JSEP

The material presented in this model is not new. Its originality lies in making an attempt at completeness in order to produce a compact encoding of type fonts that will be close to the way humans think about fonts, and thus will be readily understood and can be modified in a generative application.

3. Program to Generate Type Fonts

The CSD program (for Character Simulated Design) is a set of 44 routines, one for each character shape, rather than for each letter. The shapes of two characters are said to be identical when the primitives that they use are taken from the same families, as defined in observation 5, and laid out in the same spatial disposition. This occurs when a lowercase character is a scaled-down version of its uppercase equivalent. When a character is to be generated for some letter, the corresponding routine receives the values of the parameters for the primitives that make the letter and values to specify their relative disposition. These parameters do not fall into the classification by primitive, but can still be kept in the families derived in observation 4; for example, height of the bar of H relative to the base line in the height family or distance separating the two stems of H in the horizontal extension family. The character routine transforms the relative information into absolute form and transmits it, together with the appropriate list of parameters, to routines of another kind, the primitive routines.

Such routines incorporate the figures that define the primitives through parametric representations. They draw the outlines of the actual shapes at their absolute locations. To form the character, these outlines are filled in and the result is sent to an output device. As for the primitive routines, it is clear that they are not specific to a primitive but rather to a family, as defined in observation 5. The rules of disposition allow for an even more integrated treatment, by singling out constant spatial combinations. This is especially true of the relative arrangement of basic strokes and serifs. The stem, for example, bears only two kinds of serifs, one for upper case, the other for genuine lower case (that is, not scaled-down from upper case). In general, a routine for a primitive incorporates a basic stroke and its few serifs. The present set of 13 routines is listed in Fig. XV-5.

In order to generate outlines, the primitive routines use 3 different types of curve. The most important is the general conic section, a unicursal curve of order two. Figure XV-6 illustrates how an arc of this curve is defined: the fifth parameter required to specify the arc, together with end points and end tangents, reflects the "squareness" of the arc between the limiting cases of blunt straight segment AB and sharp corner ACB. The second curve is a straight line, a quicker way to represent limiting cases of the first type, whenever they are predictable. The third curve is used to draw half-bowls. They are unit circles for the Euclidean norms $(x, y) \rightarrow \left|\alpha|x|^r + \beta|y|^r\right|^{1/r}$, which have also been referred to as superellipses.[1, 9]

JSEP

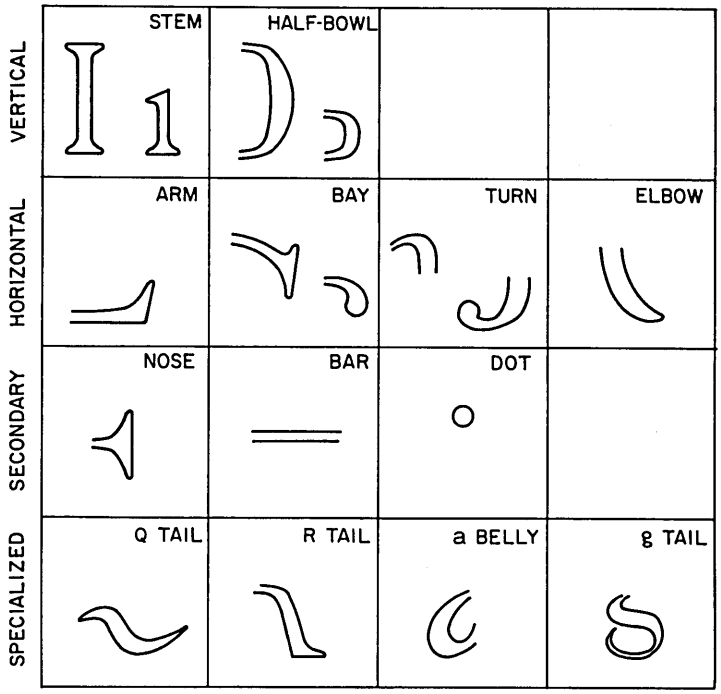
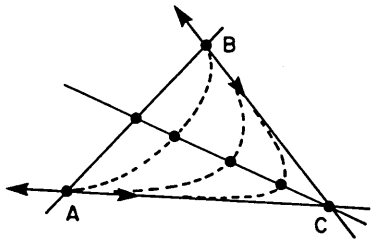


Fig. XV-5. Primitives.

Computation of the third type of curve is slow,[9] and so a simple transformation of the parametric coefficients that allows for their replacement by curves of the first type was made. Such curves have been retained temporarily in our program, however, because they are easier to rotate, which is necessary to account for diagonal stresses in half-bowls. The structure of a primitive routine can now be understood: From the parameters the routine ascertains a ring of break points along the outline of the shape to be drawn, together with sufficient information on the arcs of the curve which approximate the outline between each pair of adjacent break points. Basic subroutines actually draw the arcs, one by one. Although they are invisible to the user, the break points are important insofar as they make for the complexity or simplicity of the routines. Figure XV-7 illustrates the stem.

The code for a character appears to have two parts: a list of values for the special parameters of the letter and a list of primitives used by the letter, and for each part there is a list of values for the parameters. At this point the distinction is made between the different primitives of a given family. There is an average of 10 parameters per primitive but this covers cases from 3 to 30, depending on the shape. The reason for this is that the conciseness of the character code does not result from restricting the variability of the individual shapes but from a conscious desire on the part of the artist to relate the shapes to one another. On the average, there are altogether 27 numerical values per character, with a 7-55 range. This study, however, refers to type fonts,

JSEP



$$x(t) = \frac{a_1 t^2 + a_2 t + a_3}{c_1 t^2 + c_2 t + c_3}$$

$$y(t) = \frac{b_1 t^2 + b_2 t + b_3}{c_1 t^2 + c_2 t + c_3}$$

Fig. XV-6. Defining a unicursal curve of order two.

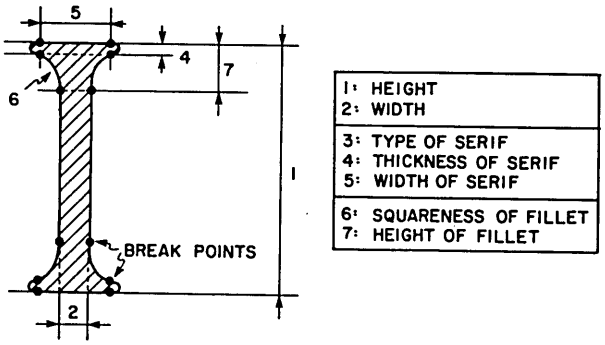


Fig. XV-7. Example of a primitive: uppercase stem stroke.

not to characters per se and, in view of observation 3, all characters of the same font share the same values for the primitive parameters. Hence those values are listed apart from any specific character, as part 1 of the font code. The collection of special character codes, which should be sufficient, comprises part 2 of the font code. There are roughly 300 primitive parameters and 250 letter parameters; those parameters called discrete types are excluded because they indicate an option in a very few cases (for example, the shape of the g may be either classical or much simpler, italic in essence). This would make an average of 10 numeral values per character, a considerable reduction. Observation 3 is not so strong, although it does not ensure exact identity. Rather, it indicates that on the average departures from the common rule are small and infrequent. This calls for a variable-length differential encoding of these departures, which constitutes part 3 of the code. (See Table XV-2.) The actual version of CSD is oriented toward creating new type and does not support the character code just mentioned. Only the values for the primitive parameters are permanently recorded; that is, the first part of the code. The two other parts are typed in an interactive manner each time a character is drawn. Except for these aspects on encoding, the previous description reflects the present CSD program.

JSEP

Table XV-2. A variable-length code.

thick stem:	141, 21, 2, 3, 62, 150, 36	(implicitly taken from part 1 of the font code)
thin stem:	4, 2, 3, 48, 150, 36	
X: ≡	special parameters:	11, 34, 39, 0
	reference to thick stem:	0000000 (bit string)
	to thin stem:	000100
	variable part:	+4 (changes)

To be interpreted as: one change for the 11th parameter of the implicit list of primitive parameters for X, which is the 4th parameter of the primitive "thin stem." The new value for this parameter is: 48+4 = 52.

4. CSD Program Performance

CSD runs on a PDP-9 computer with 32K of core and a disk facility. Apart from the most basic subroutines, CSD is entirely written in Fortran-IV. To demonstrate its capabilities, 4 common type faces were chosen. Their parameters were measured by hand with a ruler, and their values were input to CSD. The shapes that were obtained were displayed on a television screen, the parameters were adjusted interactively, and the final characters were recorded with a flying-spot photo display. Figure XV-8 shows reproductions of Baskerville, Bodoni, Cheltenham Medium, and Times Roman Bold type fonts, together with reductions of the original alphabets, which are approximately 30 cm × 22 cm. The originals, which are enlarged versions of actual type fonts whose point size is not given, were taken from Biegeleisen.[11] Type fonts are not designed merely to display alphabets, so a few examples of text and monograms are shown in Fig. XV-9. The setting is not intended to be perfect. The digital grid on which the characters were computed is 256 × 256. Taken from the generation of Baskerville, the average time per character is approximately 30 s, ranging from 5-85 s. (See Table XV-3.) The speed limit of CSD is probably between 10 and 100 characters per second; significant improvements are possible at the level of the routines, the programming language, and the hardware implementation. To achieve such a limit on a continuous basis would require that all routines be in core at once. Such a demand must be examined in view of the fact that the computer code is approximately 150K long and requires a dynamic loading scheme.

The size of the character code is irrelevant to the actual implementation of CSD, which only remembers part 1 of the code. Nevertheless, it is a key issue for the whole model and the results are summarized in Table XV-4. From estimates on the four type fonts that were matched, on the average we can expect one third of all possible changes

Fig. XV-8. Type fonts. Upper: original. Lower: CSD reproduction. (a) Baskerville, (b) Bodoni, (c) Cheltenham Medium, (d) Times Roman Bold.

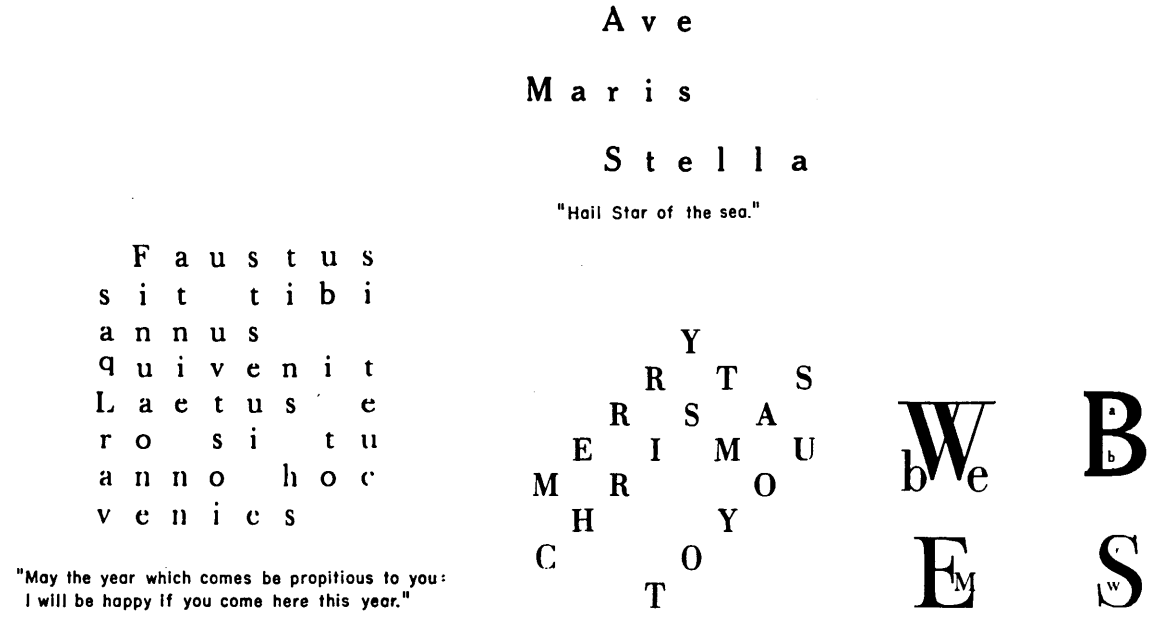


Fig. XV-9. Text and monograms.

JSEP

Table XV-3. Some time measurements.

A:	30 s	a:	15 s
B:	1 min 05 s	b:	40 s
H:	10 s	h:	14 s
I:	5 s	i:	6 s
O:	40 s	o:	20 s
T:	10 s	t:	7 s
W:	1 min 25 s	w:	1 min 20 s

Table XV-4. Space requirement for the character code (8 bits per numerical parameter).

		Num	Bits	Num	Bits	Num	Bits
Shared data basis	discrete type bits	20	20				
	upper case			10	10		
	lower case					10	10
	numerical parameters	310	2480				
	upper case			130	1040		
	lower case					180	1440
Special parameters	discrete type bits	30	30				
	upper case			15	15		
	lower case					15	15
	numerical parameters	240	1920				
	upper case			100	800		
	lower case					140	1120
Position bits		870	870				
	upper case			400	400		
	lower case					470	470
Parameter changes		250	2000				
	upper case			100	800		
	lower case					150	1200
Total (bits)			7320		3065		4255
Average per character			140				
	upper case				120		
	lower case						160

JSEP

to appear in the variable-length part. The final average is 140 bits per character when 8 bits per numerical factor are used. The average was 110 bits when 6 bits were used.

Limited as it is, CSD does illustrate the model that we have described. Paying the price for such a purpose, CSD, with its large computer code and relatively slow output rate, relates the variety of roman printed type fonts to a relative scarcity of choices because of a strong internal structure. At the same time, it provides a very compact character code that is the closest to the human interpretation of the characters.

5. Applications

We have shown that the model has two basic qualities that may lead to real world applications. The first feature is the compactness of the character code. Type manufacturers and printing houses that handle a large set of fonts in a computerized environment might use such a coding scheme to store fonts in as many variations and point sizes as they can take, however infrequent their use may be. The decoding process is still too slow for real-time applications and too complex for small-scale applications. The second feature is that it can be easily understood by a human being with little training. This provides a new way for digitizing old type fonts without scanning the input. The operator may also be a type designer looking for new patterns consistent with the roman style. Besides the advantages shared with any graphic interactive system, CSD helps the operator to respect the constraints with which he is willing to comply, since it models the constraints in the way parameters are shared among letters and primitives.

6. Conclusion

A model for character description is worth applying to automatic character recognition for the following reasons. It is valid over a large class of textual fonts and therefore suitable for the multifont problem for printed matter. The model accounts for the consistency within a font and hence has predictive power, which makes it easier to train the reader. It deals with simple parts of characters, and thus may help the reader to be less sensitive to certain kinds of noise, such as kerning effects, missing parts, and so forth. Future research might deal with the question of whether it is possible to describe a whole family of fonts, in all point sizes, in a way that, without undue distortion, would take into account similarities among different fonts. It might also be considered how missing characters such as digits, italics, and so forth, could be described.

References

1. H. W. Mergler and P. M. Vargo, "One Approach to Computer Assisted Letter Design," J. Typograph. Res. 2, 299-322 (1968).

JSEP

2. L. Rosenblum, "Harmonic System for Computer-Controlled Typesetting," Penrose Annual, Vol. 61, pp. 257-266, 1968.
3. C. Cox, B. Blesser, and M. Eden, "The Application of Type Font Analysis to Automatic Character Recognition," Proc. Second International Joint Conference on Pattern Recognition, August 13-15, 1974, Copenhagen, Denmark, pp. 226-232 (IEEE Cat. No. 74CHO 885-4C).
4. M. Meiss, "The First Alphabetical Treatises in the Renaissance," J. Typograph. Res. 3, 3-30 (1969).
5. A. Durer, On the Just Shaping of Letters (Dover Publications, Inc., New York, reprint 1946).
6. D. M. Anderson, "Cresci and His Capital Alphabet," Visible Language 5, 331-352 (1971).
7. A. J. Frank, "High Fidelity Encoding of Two-level High Resolution Images," IEEE International Conference on Communications, Seattle, Washington, June 11-13, 1973, pp. 26-5 – 26-11.
8. B. Baumgart, "Image Contouring and Comparing," Stanford Artificial Intelligence Laboratory Memo AIM-199 (1973).
9. A. J. Frank, "Parametric Font and Image Definition and Generation," Proc. Fall Joint Computer Conference, American Foundation of Information Processing Societies, Las Vegas, Nevada, November 15-18, 1971, pp. 135-144.
10. P. Coueignoux, "Compression of Type Faces by Contour Encoding," S. M. Thesis, Department of Electrical Engineering, M. I. T., 1973.
11. J. I. Biegeleisen, Art Directors' Book of Type Faces (Arco Publishing Co., New York, 1967).

JSEP