

A Dialogue of Forms:

Letters and Digital Font Design

Debra Anne Adams

A. B. Degree Vassar College
May 1978

Submitted to the Department of Architecture
in partial fulfillment of the requirements of the Degree

MASTER OF SCIENCE IN VISUAL STUDIES
at the **Massachusetts Institute of Technology**

September 1986

Massachusetts Institute of Technology 1986

Signature of Author

Debra Anne Adams
Department of Architecture
August 8 1986
DA Adams

Certified by

Muriel Cooper
Muriel Cooper
Associate Professor of Visual Studies
Thesis Supervisor

Accepted by

Nicholas Negroponte
Chairman
Departmental Committee for Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

AUG 29 1986

LIBRARIES

Rotch

A Dialogue of Forms:

Letters and Digital Font Design

Debra A. Adams

Submitted to the Department of Architecture on August 8 1986
in partial fulfillment of the requirements of the Degree of

MASTER OF SCIENCE IN VISUAL STUDIES
at the **Massachusetts Institute of Technology**

ABSTRACT

A Dialogue of Forms is an investigation of typeface design tools and processes. The aim of this investigation is to develop techniques of deriving letterforms automatically from a subset of letters called the control characters. The control characters are representative letters that contain the primary structural elements, design attributes, and proportional relationships that characterize a typeface. Design information derived from the control characters is used to constrain the design of other letterforms. The lower case letters o, h, v, and p are the control characters studied in this investigation.

The control characters are interactively created and edited by the designer, and stored as sets of primitive parts. These parts are used as building blocks to construct other letters automatically. Knowledge about letterform structure and font design consistency is represented and used to manage the derivation process. Generated designs may be edited by the designer and changes to parts can be propagated.

Automatic letterform derivation can aid the designer by reducing time consuming labor. As a visualization tool, it provides a fast and efficient means of evaluating a design idea.

Thesis Supervisor Muriel Cooper
Title Associate Professor of Visual Studies

To Matthew

TABLE OF CONTENTS

2	Abstract	
5	Introduction	
9	Typeface Design	ONE
18	Letterform Structure and Design	TWO
32	Type and Technology	THREE
43	Digital Font Generation	FOUR
59	Letterform Derivation	FIVE
73	Project Description	SIX
79	Software Design and Implementation	SEVEN
100	Conclusion	
104	Reference Bibliography	
121	Appendix	
156	Acknowledgments	

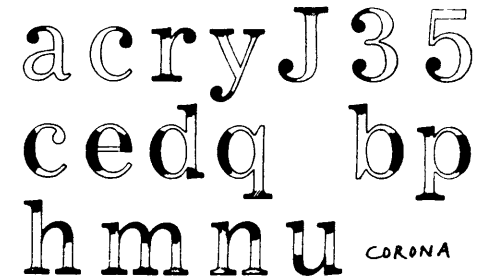
INTRODUCTION

Reduced to simplicity, typeface is a specific set of design ideas used to clothe a basic letterform. It is this set of design ideas which is totally aesthetic or artistic.

Mergenthaler Linotype Company

The task of the typeface designer is to conceive a design idea and apply it consistently to all characters in a font or font family. Conceptually, this process is structured and systematic. Letterforms are visually related in weight, shape, spacing, and alignment. Drawn in consistent fashion, key design elements repeat and blend. (See Figure ia) "Thus it is not a question of designing a group of beautiful letters, but rather designing a beautiful group of letters." [Mergenthaler Linotype Company 1971] (See Figure ib)

A Dialogue of Forms is an investigation of the process and practice of creating typeface designs. The aim of this investigation is to examine techniques of automating the generation of letterforms. It is hypothesized that letters can be automatically derived from a subset of forms called the control characters. The control characters contain the primary structural elements, design attributes, and proportional relationships that characterize a typeface. Typically they are the first letters created by the designer "since their design



a c r y J 3 5
c e d q b p
h m n u CORONA

Figure ia



harmony
harmony

Figure ib

INTRODUCTION

would suggest how the remaining letters and characters should be drawn."

[Mergenthaler Linotype Company 1971] (See Figure ic)

The concept of automatic letterform derivation differs from current font generation systems in the following fundamental way. Current systems require the user to create each individual character shape, character shape primitive, or structural representation in a font. No software exists to automate this process. Batch techniques are primarily applied to the generation of alternate font sizes, weights, and resolutions. To create these additional ranges, one or more complete fonts must be designed and input by the user.

The idea presented here is that the designer can create a subset of letters and the system can be used to automatically generate preliminary designs of the remaining characters. With the use of interactive tools, shape modifications can be incorporated and automatically filtered throughout a font. The designer continues to work back and forth among letters to define subtle typographic details and to create a unique design pattern.

Thus the system proposed in this thesis is not intended to remove the designer from the creative process. As Donald Knuth writes: " ... an enormous amount of subtlety lies behind the seemingly simple letter shapes that we see

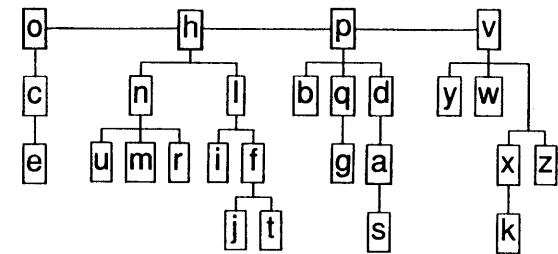


Figure ic

INTRODUCTION

every day, and the designers of high-quality typefaces have done their work so well that we don't notice the underlying complexity." [Knuth 1986] Type design requires extensive skill in letter drawing, expertise in the area of printing, an understanding of the reading process, and an artistic sensitivity to form. No formalized body of rules exists, to date, that can be applied to the systematic production of high-quality, finished typefaces. However, automatic processes can provide the artist with a fast and efficient means of evaluating a design idea and can reduce time-consuming labor.

As a preliminary study, this thesis paper functions as both a survey and an analysis. Chapter One is devoted to a discussion of the type design process and the functional role of consistency and contrast in reading. Chapter Two describes letterform structure and design relationships. The impact of printing materials and processes on design and style is introduced in Chapter Three. This is followed by a description of digital font generation and a review of current design systems and research work related to this thesis in Chapter Four. Chapter Five describes the role of the control characters in the derivation process and general techniques used by designers to create a set of letterforms from the control characters. In Chapter Six, the demonstration

INTRODUCTION

software project that accompanies this paper is introduced. This software was developed to illustrate the derivation concept and to test procedures and representations useful to automatically generating consistent letter designs and to propagating changes to letter contours. The software design and implementation is described in detail in Chapter Seven. The concluding chapter contains a brief software analysis and recommendations for future research.

This investigation is limited to a study of lower case, or miniscule, letters in sans serif typefaces in general and in Helvetica in particular. Helvetica was chosen because it is a highly regularized sans serif design and its letters are conventional forms. Miniscule letters were chosen because they are more differentiated in design than upper case, or majuscule, forms. Upper and lower case letters can be derived according to similar design principles. The control characters used are o, h, v, and p.

It may be easy to think of one letter, but to think of its twenty-five relations which with it form the alphabet and so to mark around them that they will combine in complete harmony and rhythm with each other and with all - that is the difficult thing, the successful doing of which constitutes design.

Frederick W. Goudy

The type designer is concerned with the perceptual requirements of the reader. As such, the designer must develop a precise and microscopic knowledge of the visual effects of letter shapes, massed together, and seen at small sizes. For legible results, letterforms must be identifiable and familiar, clearly contrasted in structure, even in weight and spacing, and harmonious in style.

The 'certainty of decipherment' is an important element in true legibility; and in relation to typography it bears the message that legibility, or ease of reading, is increased by letters that are clearly distinguished from each other and decreased by letters that look too much like each other. [McLean 1980]

Contrasts in letterform structure create variations in word shape when letters are combined. (See Figure 1a) During the reading process, word

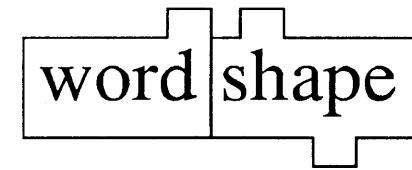


Figure 1a

shape patterns are perceived. Javal, in 1878, concluded that distinguishing letter features predominate along the upper portion of a line of text. [Spencer 1968] (See Figure 1b) Twenty five years later, Messmer postulated that words composed primarily of a variety of contrasting shapes are more legible than those composed of structurally similar forms. [Spencer 1968] In 1940, Tinker differentiated between total word shape and total word structure. The total word structure consists of both the word outline and the pattern produced by its internal configuration of light and dark values. [Spencer 1968]

Whereas contrasts in letterform structure facilitate word identification and recognition, consistencies in style and design ease the flow of reading. "Where the text letters are uniform, the reader is free to give his attention to the sense of the words." [Johnston 1977] Letters are designed to combine and to produce an even impression and tone on the page when set next to one another. (See Figure 1c) No single distinguishing letter feature should dominate or attract the eye. Visual harmony preserves clarity of form.

These typeface designs involve a considerable amount of talent and creative product not only to create a pleasing and effective design of a single letter of type, but also to provide a consistent pattern of design which

for offset litho printing
 for offset litho printing

Figure 1b

NEWS NO. 9
 The history of writing is the history of the human race, since in it are bound severally and together, the development of thought, of expression, of art, of intercommunication and of mechanical invention. It has been said that the invention of writing is more important

PALADIUM
 The history of writing is, in a way, the history of the human race, since in it are bound up, severally and together, the development of thought, of expression, of art, of intercommunication, and of mechanical invention. Indeed, it has been said that the invention of writing is more important than all the victor

AVANT GARDE GOTHIC MEDIUM CONDENSED
 The history of writing is, in a way, the history of the human race, since in it are bound up, severally and together, the development of thought, of expression, of art, of intercommunication, and of mechanical invention. Indeed, it has been said that the invention of writing is more important than all the victories ever won or constitutions

Figure 1c

will enable the various letters to be fitted together in all of the hundreds of thousands of permutations and combinations of twenty six lower case letters, twenty six capital letters as well as all of the additional symbols, punctuation marks and numbers necessary to complete the family of print.
[Mergenthaler Linotype Company 1971]

The task of the designer, then, is to blend the visual contrasts between letters without impairing their legibility. This is achieved through regularity and repetition in design. Letterforms are consistently created in weight, spacing, and alignment. Design features such as the curve axis and stroke endings are structured and repeat. Regularities exist on several levels from general similarities in shape and structure to subtle curve relationships and size proportions. Subregions of each letter image are designed to interact.

The difference between the look of one type and the look of another is the difference between thousands of tiny repeating details that have been carefully orchestrated or arranged and combined by the typeface designer. [Mergenthaler Linotype Company 1971]

When sufficiently varied and sufficiently uniform, letters create an integrated texture and a rhythmic pattern of values.

The Design Process

While formal, written rules exist in calligraphy books for hand drawing consistent alphabets with a brush or pen, no such codified knowledge can be found in the literature on printed forms. (See Figure 1d) Writers on type design refer to the "harmony", "family likeness" , and "unity" of letters in vague fashion, seemingly unable or unwilling to explicitly describe their processes and principles of design.

To the accomplished letterer, there may be guidelines but there are no rules. The overriding consideration is that the result be harmonious and pleasing within the context of the alphabet's intended function. Such a result comes about through subjective judgements rather than through mathematical precision. [Mergenthaler Linotype Company 1971]

The lack of explicit rule description in type design compared to that found in calligraphy can be explained, in part, by differences in the formation and technical production of hand written and type drawn alphabets. Brush and pen letterforms are composed of a series of individual lines, called strokes, drawn by sequential movements of the writing tool in the plane of the writing surface. (See Figure 1e) Each stroke primitive is defined by a distinct hand

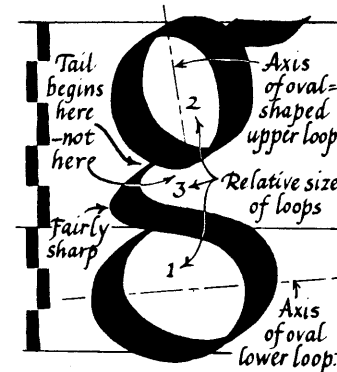


Figure 1d

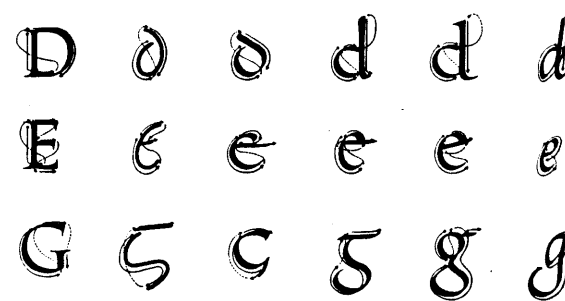


Figure 1e

motion. The pattern and shape of hand movements, called the ductus by Bigelow [Bigelow 1983], describes the underlying letterform structure and sequence of stroke composition.

Repeating stroke primitives are consistent in character due to uniform movements of the writing tool. (See Figure 1f) In printed fonts, these visual consistencies are maintained. However, type drawn letterforms contain subtle variations in contour curvature not found in hand written letters. Whereas brush and pen stroke contours and characteristics are constrained by the movement and use of the writing tool, its angle with respect to the horizontal, its flexibility, and its shape and size, type drawn contours vary independently. Each edge is modifiable and unique. (See Figure 1g) Thus, printed stroke consistencies in weight, shape, and proportion are created by manipulating contour edge features to accomodate the eye. They are consistencies of visual appearance and not of actual physical dimensions. (See Figure 1h)

Designers know, for instance, that there are visual interactions between the elements of a character shape that affect the way it is perceived; they know also what the nature of these interactions are, and that they are governed by certain rules. But they cannot formulate these rules

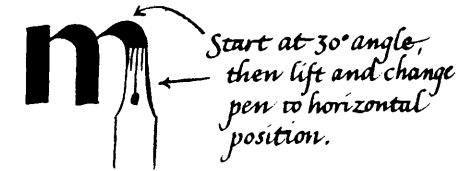


Figure 1f

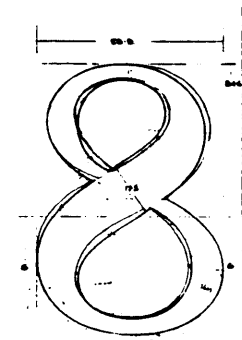


Figure 1g

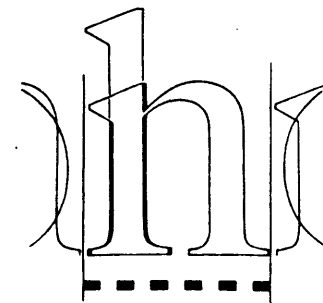


Figure 1h

otherwise than by making shapes that take the effects of the interactions into account.
[Southall 1985]

Through a lengthy process of iterative testing and proofing, the type designer draws letter shapes and "changes them until they look correct."
[Southall 1985] Optical properties of weight, shape, fit, and alignment are modified and refined in relation to one another. As visual contrasts are reconciled, design influences overlap and become interwoven. (See Figure 1i)
Typeface texture and rhythm slowly evolve in the context of words and control strings. "The type designer thinks with images, not about them." [Bigelow 1982]

Frederick Goudy, describing his design process in the book Typologia, writes:

For myself, I usually begin a new type with some definite thought of it's final appearance, though it may be no more than the shape or position of the dot of the lowercase i, a peculiar movement or swell of a curve, or the shape or proportion of a single capital. From such humble beginnings I progress step by step, working back and forth from one letter to another as new subtleties arise, new ideas to incorporate, which may suggest themselves as the forms develop,

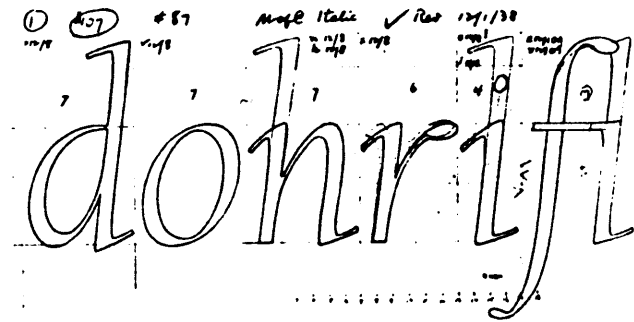


Figure 1i

until finally the whole alphabet seems in harmony - each letter the kin of every other and of all. [Goudy 1977]

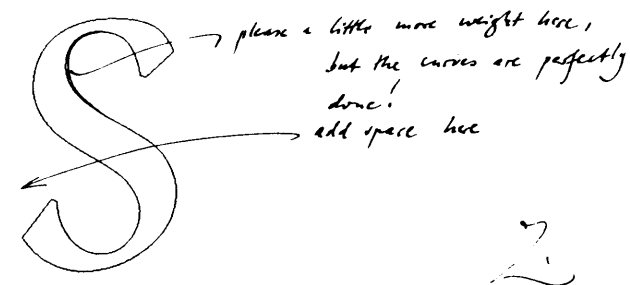
The typeface designer learns through apprenticeship and practice. His knowledge is craft knowledge; "it has become part of the intuitive understanding of the person concerned" [Southall 1985] and cannot necessarily be stated in explicit form.

Traditional lettering artists draw large filled outline contours with pen or pencil on paper or transparent film and build and edit shapes by cutting and pasting pieces of letters together and reworking letter contours. As each character is rendered it is placed in a word or control string to judge its width relationships and to determine its spacing parameters. (See Figure 1j) Set in words and phrases, its integration and rhythm with other letters are viewed, compared, and studied. Its design may cause a change in another letterform or set of letters. These changes are made by redrawing the selected characters, incorporating the editing changes, and again proofing and marking letters for correction. (See Figure 1k) To accurately judge a design, small scale proofs are made or the designer stands back and views the letters through a reducing glass. This process can continue for as long as two years until the character

HAHHHOHRHaHbHeHgHhHiHmHnHoHrHsHuHv
 OAOHOOROaObOeOgOhOiOmOnOoOrOsOuOv
 RARHRORRRaRbReRgRhRiRmRnRoRrRsRuRv
 aAaHaOaRaaabaeagahaiamanaoarasaav
 bAbHbObRbabbbegbbhbibmbnbobrbsbub
 eAeHeOeReaebeeegeheiemeneoeresuev
 gAgHgOgRgagbgeggghgigmngogrgsgugv
 hAhHhOhRhahbbehghhhhhimhnhohrhshuhv
 iAiHiOiRiaibieigihiiiminoirisuiuv
 mAmHmOmRmambmemgmhmimmmnmommsmumv
 nAnHnOnRnanbnengnhninmnnnonmsnunv
 oAoHoOoRoaoeboeogohiomonooorosouov

 rArHrOrRrarbrergrhrirmrortrrsrurv
 sAsHsOsRsasbsesgshsismnsosrsssuv
 uAuHuOuRuaubueuguhuiumunuourusuuv
 vAvHvOvRvavbvevgvhvimmvovrvsvuuv

Figure 1j



15 Figure 1k

set is complete.

In order to appreciate the magnitude of the problem, consider the variability of letterforms that is reflected in a single superfamily of typeface designs. For each modern design one of each of three opposing features must be specified: whether the type is roman or italic, whether it is normal weight or boldface and whether it is serif or san-serif...Taken together, the three features generate eight typeface designs. Furthermore, each type alphabet typically includes characters in 16 different sizes. The total number of glyphs, or individual bit maps, necessary to accommodate a single character for a minimum superfamily of type is therefore 128; the number of glyphs necessary for a complete superfamily, which may include 128 letterforms, is 128^2 , or more than 16,000. [Bigelow 1983]

During the initial rendering process, the designer creates a set of control characters or key letterforms used to define the visual attributes of a typeface. These attributes include the width, height, and alignment relationships, the curve axis, the letter spacing or set width, and the stroke weights and stroke endings. Design information contained in the control characters is mapped from letterform to letterform. Through this sequence of mappings

visual relationships are structured and stylistic consistencies emerge. "It is as though you have to take the qualities of a given 'a' and, so to speak, hold them loosely in the hand as you see how they slip into variants of themselves as you carry them over to another letter." [Hofstadter 1985]

In order to provide a framework for discussing the role of the control characters in the design process, Chapter Two will be used to define the characteristics of letterforms and their design relationships.

In the "Statement of Mergenthaler Linotype Company in Support of the Registerability of the Claim of Copyright in Original Typeface Design", the following definition for "typeface" is given.

As used herein, the term 'typeface' shall mean sets of designs of a) letters and alphabets as such with their accessories such as accents and punctuation marks, and b) numerals and other figurative signs such as conventional signs, symbols and scientific signs, which are intended to provide means for composing texts by any graphic technique. [Mergenthaler Linotype Company 1971]

A "font" is defined by Mergenthaler as:

The type font is merely the assortment of a typeface in a particular size or style for a particular purpose. In any given font, there are usually seventy to ninety or more characters. [Mergenthaler Linotype Company 1971]

The focus of this investigation is on the relationships that exist among letterforms within a type font. In other words, we are interested in what the letters in each horizontal row in Figure 2a have in common and not in what the letters in each vertical column have in common.

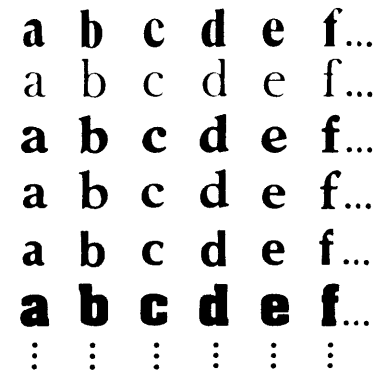


Figure 2a

visual structure and a set of part primitives that distinguish its identity as a unique element of the alphabet. The type designer does not invent new conventional letter structures; he uses those that already exist. "The basic forms of letters are fixed; that is, they have become classic." [Goudy 1977] Reproduced in Figure 2b are conventional forms of the miniscule alphabet. These letters are from the typeface Helvetica.

The structure of a letterform constrains its part configuration, or the spatial relationships among its parts, and their joining characteristics. Within the miniscule alphabet, certain part configurations are valid. For example, the two bowls of a capital B are situated to the right of the stem and horizontally aligned in relation to it. (See Figure 2c) Therefore, sets of rules can be defined to describe the relative positions of each part and their types of linkage. The attributes of each letterform define the horizontal position, orientation, alignment, size, and shape geometry of its part configuration.

Reproduced in Figure 2d are the parts of each letterform, commonly called strokes. As mentioned in Chapter One, the term "strokes" derives from pen lettering and refers to the set of discrete lines drawn with the writing tool to form each letter. In printed fonts, stroke shapes are defined by sets of

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

Figure 2b

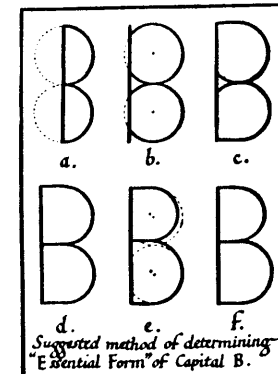


Figure 2c

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

Figure 2d

contour edges hand drawn by the designer or engraved in metal. Parts function as integrated elements within each letter. Depending on the typeface, context, or use, their boundaries may be redefined. Although no standard nomenclature exists for naming the parts of letters, the system developed by Philip Gaskell for labelling the parts of serif letter designs utilizes conventional terms. (See Figure 2e)

Each part has a characteristic set of visual attributes that define its horizontal position, orientation, vertical alignment, shape and size within each letter. The shape boundaries and attributes of a part are constrained by the position, length, direction, curvature, and joining relationships of its contour edges. There are two general part shape types: straight and curved. Straight strokes vary in length, thickness, direction, and slope. Curved strokes vary in length, thickness, direction, and curvature.

Parts that share common shape attributes are visually related and they may be grouped into the part primitive classes illustrated in Figure 2f . Within each class, subclasses of parts can be defined such as the ascender stem subclass or the crossbar subclass. Parts within each subclass share identical or nearly identical sets of shape attributes. They are consistently designed

LETTERFORM STRUCTURE AND DESIGN

TWO

MINUSCULES

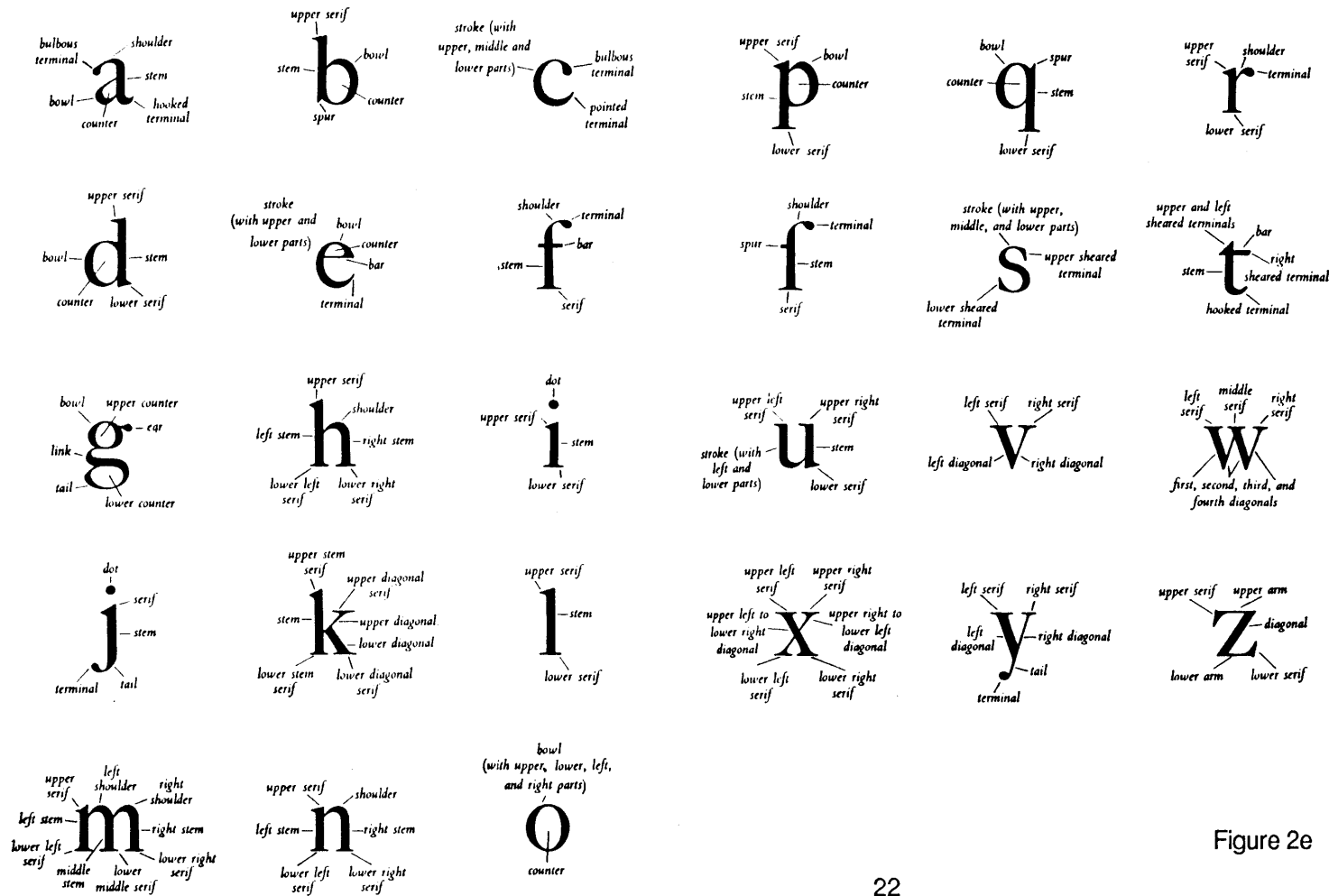


Figure 2e

a b d f g h i j k l
m n p q r t u

k v w x y z

e f t z

a b d g q p

c e o

h n m u

a f j t y

c e o

a g s

throughout a typeface. However, although they may appear visually consistent, repeating part instances often differ in their physical geometry due to the visual interactions within and among letterforms. Therefore, each repeating part instance can be inherently nonuniform in character.

Although differentiated, sets of letterforms share parts in common. Within the miniscule alphabet, four general types of letterforms can be discerned. They are those composed primarily of (1) vertical strokes, (2) curved strokes, (3) vertical and curved strokes, and (4) oblique strokes. (See Figure 2g) These are referred to as the square letters, the round letters, the square and round letters and the oblique letters respectively. Their basic shapes repeat throughout a typeface design.

The control characters are representative letterforms from each letter shape category. They contain the primary design features and proportional relationships that repeat throughout a typeface. Thus the set of control characters is used to establish the design harmonies within and among each category of letters. The primary proportions that characterize a typeface are the letter height to width ratio, the character height to stroke thickness ratio, the letter width ratios, the ascender, xheight, and descender height ratios, and the

square	i l f j r t h n u m
round	c e o s
square/round	a b d g p q
oblique	k v w x y z

Figure 2g

thick to thin stroke weight ratio. The following discussion will be used to introduce letter design relationships.

Width Relationships

The o is the primary letter in a typeface. Its round width determines the width rhythm of the remaining letters, and its width to height ratio determines the major size proportions. Except for m and w, the o and the round letters are generally the widest letters in the miniscule alphabet of a proportionally spaced font. (In sans serif cases this is not always the case.) To appear optically related in width, square letters are more narrow than the rounds. The width of the square and round letters lies between these two. The oblique letters generally appear similar or identical to the square letters in width. The single stroke letters are the most narrow. (See Figure 2h) The width relationships illustrated in Figure 2i are based on classical proportions derived from the Trajan Column inscription in Rome. The width of the square majuscules on the Trajan Column is roughly 4/5 the circular round width.

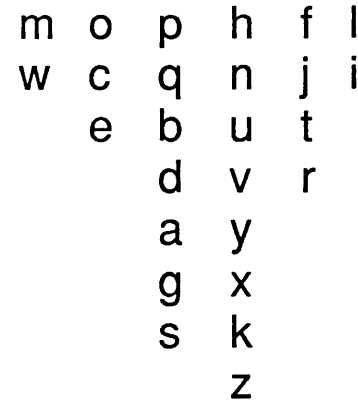


Figure 2h

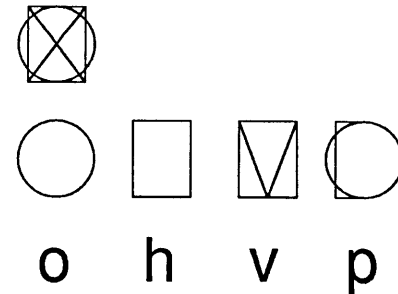


Figure 2i

Height Relationships

The heights of the letters in a typeface are proportionally related to one another. Due to the nature of visual perception and optical illusion, round, square, and diagonal letters of the same geometric height appear unequal. Therefore, the height of the round letters is slightly extended above or below the square heights, and the oblique letters dip slightly lower at their apex to compensate for these visual effects. (See Figure 2j)

Alignment Relationships

Letters are optically aligned along an imaginary horizontal line called the baseline. There are three other primary alignments in the miniscule alphabet. From top to bottom they are the ascender alignment height, the xheight or meanline, and the descender depth. Because of their actual height differences, square, round, and diagonal character alignments differ. Consequently, it is possible to imagine four or more secondary alignment lines for the round and diagonal letters. (See Figure 2k) In addition, the arches in letters such as h, n, or m may have their own alignment value.

The h is used to determine the square ascender and baseline alignment



Figure 2j

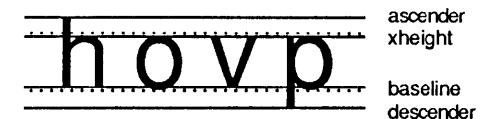


Figure 2k

and the ascender height, the p is used to establish the descender depth and the descender square alignment, the x determines the square xheight and alignment, and the v defines the diagonal alignment height. The o determines the round height and alignment and the square to round height and alignment proportions.

These alignment heights are proportionally related to one another. Typefaces can have a small xheight in relation to the ascenders and descenders or a large xheight. This can have an impact on the legibility of a typeface. At small sizes, the xheight is generally enlarged.

Letterspacing/Set Width

The set width includes the body width of a letter and the spaces designed to its left and right, called the left and right sidebearings. The sidebearings are adjusted to determine character fit. (See Figure 2l) Character fits throughout a typeface are designed to appear optically equal in area. These areas are proportionally related in size to the area enclosed by the positive shape of each letterform, or the counterform. (See Figure 2m)

To illustrate, Figure 2n shows spacing between squares and circles



Figure 2l



Figure 2m

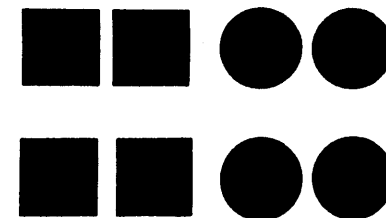


Figure 2n

which are geometrically equal. These areas appear uneven to the eye. Proper adjustment situates the squares further apart to compensate and to appear optically equal to the space between the circles. When letters are substituted for these shapes, as in Figure 2o, the spacing problem can become more complicated, depending on the configuration of square, round, and diagonal strokes in relation to one another. When the letterspacing is narrow, the white areas between letters dominate and attract the eye. Under "normal" reading conditions, counter and letterspace areas appear equal.

As the weight of a typeface increases, the body size increases, the counterspace areas decrease, and the fit between characters becomes tighter. (See Figure 2p) The set width of a letter is also influenced by the presence or lack of serifs, and their length, shape, and positioning on a letterform.

Certain character combinations need to be individually adjusted. This is called kerning. In Figure 2q, the intercharacter spacing between 'T' and 'y' appears too wide and must be reduced by overlapping the side bearings.

Stroke Thickness

Stroke thicknesses are consistently maintained throughout a typeface.

Letterspacing Letterspacing Letterspacing

Figure 2o



Figure 2p

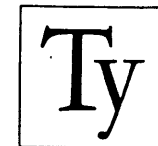


Figure 2q

However, in order to appear optically equal, they actually differ in their physical dimensions. Variations depend primarily on the stroke type and slope. Horizontal straight strokes are thinner in width than vertical straight strokes. Similarly, diagonal stroke weights lie between the horizontal and vertical and vary according to their degree of slope. Curved strokes are the thickest and gradate from thick to thin along an axis of curvature. (See Figure 2r) The curve axis may be oblique or vertical. The degree of thick to thin stroke contrast in a typeface varies and is a significant design feature which can add texture to a design. (See Figure 2s)

Further modifications in stroke weight depend on the density or complexity of a letter (an 'm' with three straight strokes in close proximity will appear too dense or black unless its stroke weights are slightly reduced), its legibility (often the top of the crotch of the 'n' in indented or thinned to accentuate its form), and the spatial location of the strokes in relation to one another (the curve weight and axis on the bowl of 'p' and 'b', for instance, may differ due to the visual interaction produced by the location and alignment of the straight stems in relation to the bowls). (See Figure 2t)

The degree of greyness, or visual weight, of a typeface is a function

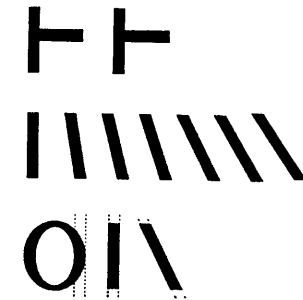


Figure 2r

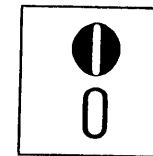


Figure 2s



Figure 2t

of the stroke thickness and its relationship to letterform size. Stroke weights merge with counterspace areas to create the image weight of each letter. Character height and width affect the overall black to white ratio. Tall letters will appear visually thinner than short letters of the same stroke thickness and wide letters will appear less heavy than narrow characters. (See Figure 2u)

Stroke Endings

Serif designs differ in length, shape, degree of contrast, placement, and alignment. Top and bottom serifs often differ in appearance. Serifs contribute to the texture and pattern of a typeface design. (See Figure 2v)

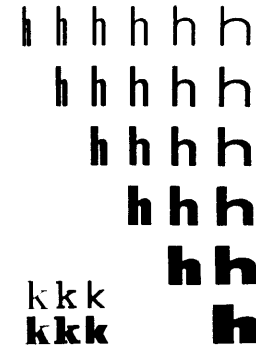


Figure 2u

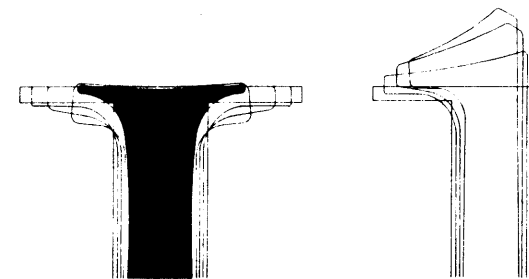


Figure 2v

All technical requirements must be considered and regarded even at the drawing stage. A printing face is the sum of a series of factors which must be fused into harmonious unity if a useful type is to result. To be so designed, a type demands of its designer the knowledge of historical coherence in type development, artistic perception, and an inclusive insight into the technique of typesetting.

Hermann Zapf

In addition to the requirements of legibility, each typeface design must be adapted to the materials and technical processes of printing and type founding in order to be reproduced faithfully and consistently. This relationship between design and technology has altered the design characteristics and proportions of letterforms over time. "The first printers did not realize that the printed form had its own kind of laws and was capable of making its own kind of impact." [Bigelow 1983] As type design moved from its imitative phase into innovation, written letter shapes were reinterpreted as typographic forms. "Proportion, width, weight, and construction were altered independently of the underlying topology of the letter, rather than being partially determined by it as they were in the ductal letter." [Bigelow 1983] (See Figure 3a)

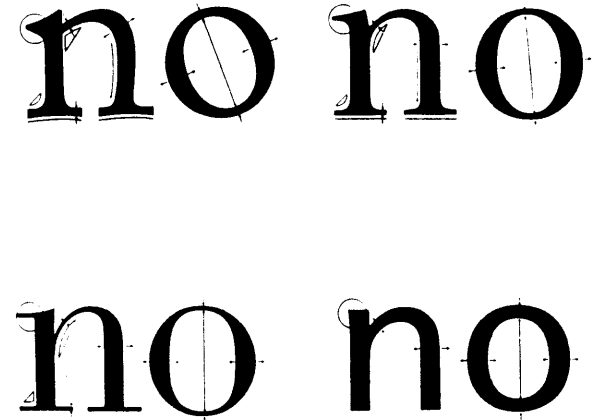


Figure 3a

Reproduced in Figure 3b (next page) are typefaces that illustrate significant design changes that have occurred over the past 500 years. Oldstyle typefaces exhibit the round letters, oblique curve axis, minimal thick and thin stroke contrast, and concave serifs characteristic of manuscript forms. Transitional faces, such as Baskerville, contain a greater degree of thick to thin stroke contrast, shorter and less concave serifs, and the curve stress varies from oblique to vertical. Baskerville's designs were influenced by the introduction of smoother papers. [Ruggles (in preparation)] Copperplate engravings had a significant impact on the design of Modern typefaces. Thin strokes became hairlines and serifs were slightly bracketed or not bracketed at all. The rise of commercial printing during the Industrial Revolution created a demand for typeface designs that could be used for display purposes, periodicals, and newspapers. In the early 1800's, square serif monoline faces were designed. Although many weights and widths of square serif typefaces were eventually produced, the original letterforms were very bold in weight, with minimal contrast in stroke thickness and little serif bracketing. During the nineteenth century, an abundance of decorative, embellished faces were created.

(not shown) Sans serif types appeared in the 1800's and were revived in the

24 point Centaur (Foundry)

ABCDEFGHIJKLMNOPQRSTUVWXYZ&
abcdefghijklmnopqrstuvwxyz 1234567890\$

Oldstyle

24 point Baskerville (Foundry)

ABCDEFGHIJKLMNOPQRSTUVWXYZ&
abcdefghijklmnopqrstuvwxyz
1234567890\$

Transitional

24 point Bodoni Trueface (Ludlow)

ABCDEFGHIJKLMNOPQRSTUVWXYZ&
abcdefghijklmnopqrstuvwxyz
1234567890

Modern

18 point Clarendon Bold (Foundry)

**ABCDEFGHIJKLMNOPQRSTUVWXYZ&
abcdefghijklmnopqrstuvwxyz
1234567890\$**

Slab Serif

24 point Futura Medium (Foundry)

ABCDEFGHIJKLMNOPQRSTUVWXYZ&
abcdefghijklmnopqrstuvwxyz
1234567890\$

Sans Serif

Figure 3b

1920's and 30's in geometric form by the Bauhaus designers. These simplified letterforms represented an innovative break from the traditions of roman type. In the late 1900's, the concept of a typeface "family" composed of several variations of a single design was introduced. Reproduced in Figure 3c is the Univers design program developed by Adrienne Frutiger.

Type Founding and Print Technology

Early printed roman typefaces modelled the letter structures, proportions, and patterns of design found in humanist scripts of the ninth and tenth centuries and their inscriptional origins. Each letterform was engraved, at actual size, on the end of a steel rod called a punch. The punch was used to strike a copper matrix from which a three dimensional rectangular block of metal type was cast in an adjustable mold. The block of type contained a raised letter image, in reverse, on its face. (See Figure 3d) To print a page of text, the pieces were hand composed or set next to one another, prepared with ink, and impressed on paper.

The invention of the adjustable mold by Gutenberg in the mid 1400's made it possible to create uniform and easily replicable pieces of metal type

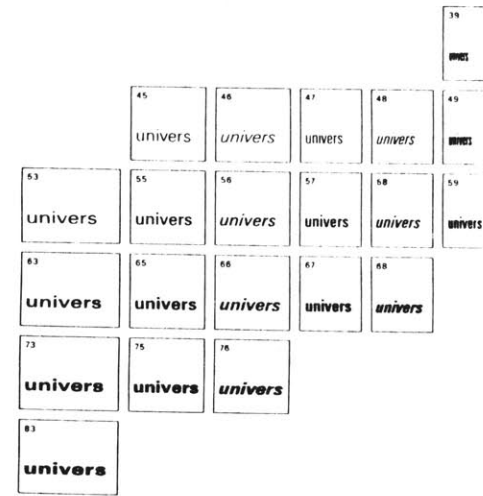


Figure 3c

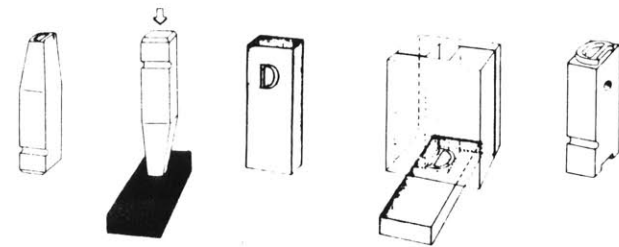


Figure 3d

that could be fit together accurately. Each piece had a constant height and thickness, but was variable in width. Letter designs were constrained within the rectangular face and carefully positioned to achieve optically even letter spacing and proper vertical alignment. Unevenly spaced character combinations were kerned or cast together on a single block of type. (See Figure 3e)

These early typeface designs were crude and irregular forms. The paper used to print text from type had a thick, spongy quality and was "dampened before use to soften its fibers so that the printing ink would adhere to it." [Ruggles (in preparation)] (See Figure 3f)

Hand-made paper of long fibre, used damp and with an elastic back, gave an impression in which the breadth of the actual lines forming the face of the type was uniformly widened, and consequently the hairlines and serifs were broadened out of proportion to the main-strokes, the external corners at the same time becoming rounded. [Legros and Grant 1916 quoted in Ruggles (in preparation)]

Typeface designs were modified to compensate for the effects of ink on paper.

In addition to the requirements of the printing process, the punchcutter and designer had to learn the subtle alterations of letter shape and proportion

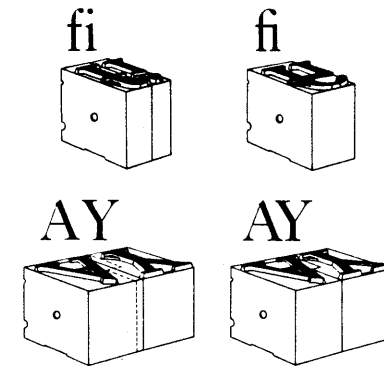


Figure 3e

RQEN
baegn

Figure 3f

that were necessary for proper legibility and consistency over a range of point sizes. (A point is a standard unit of type measurement used to calculate the height of letterforms. There are roughly 72 points to the inch.)

A sense of scale and the adaptation of letters to the various sizes of type so as to make them all as comfortable to the eye as possible is a very important part of the letter-cutter's art. It is a mistake to think that a range of types from great to small can all be made from one set of drawings. I said that before he can begin cutting a letter, a punch-cutter must have the whole fount in his mind's eye; but in fact he must do more. He must conceive a fount that is susceptible of a production in all the various sizes in which type is needed. [Carter 1954 quoted in Johnson (in preparation)]

Letter designs were not simply scaled versions of one another. Ascender, descender, and xheight size relationships, letter widths and shapes, and stroke thicknesses within each font were altered to appear visually consistent. (See Figure 3g) Although some punchcutters worked from scaled drawings, the eye was considered to be the best judge of correct form and proportion.

In 1885, Lynn Boyd Benton issued a U.S. patent for the pantograph machine. This invention ushered in the "era of type manufacture." [Southall

6 POINT
RQEN baegnov

8 POINT
RQEN baegnov

14 POINT
RQEN baegnov

18 POINT
RQEN baegnov

Figure 3g

1985] Although the pantograph was initially used to cut matrices, it was later applied to the cutting of punches. [Ruggles (in preparation)] Mechanical punchcutting was used to mass produce identical type matrices that were required, in large quantities, for the success of hotmetal typesetting systems.

The punchcutting machine was a pantograph with a sharp cutting tool on one end used to cut steel punches by tracing and reducing large metal patterns of letterforms. (See Figure 3h) The metal patterns were created from large scale drawings produced from an existing typeface or print of type, or from an original ink or pencil design. [Warde 1935] Measurements in tenths of thousandths of an inch were marked on the drawings and used to translate "...every detail...into terms of the size of the matrix which is to be struck." [Warde 1935]

The mechanical punchcutting system caused a significant change in type design practice. The emphasis shifted from making to drawing. [Southall 1985]

In hand-cutting the punch can be called the original work of art in the whole process of making type. It is that single and unique object by which one can obtain as many as 500 matrices, each matrix being capable of

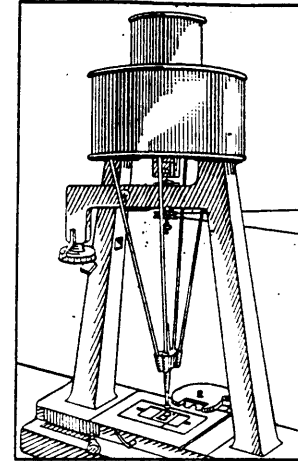


Figure 3h

forming millions of types. But in machine-cutting the unique object is the drawing, from which any number of patterns can be made, each pattern serving for any number of punches of the letter. [Warde 1935]

The need for large scale drawings forced the designer to have to previsualize or anticipate the final appearance of letters scaled down to text size. "In passing to machine production they must, for clear comprehension, first realize that here the thinking-out part of the work is separate from, and altogether precedent to, any actual making..." [Warde 1935]

The success of typesetting systems was also dependent upon the introduction of self-spacing type by Benton in the 1830's. Self spacing type is a method of tabular composition based on a unit system that divides the horizontal width of the em square into even increments. (An em square is a unit of type measurement equal to the square of a given point size of type. An eighteen point em square is eighteen points by eighteen points. (See Figure 3i)) Standard units sizes were eighteen, thirty-six, and fifty-four units to-the-em. The width of each character was calculated to be a certain number of units wide. For example, single stroke characters such as I may be six units wide while M may be eighteen units wide. (See Figure 3j) Self-spacing type was used to



Figure 3i

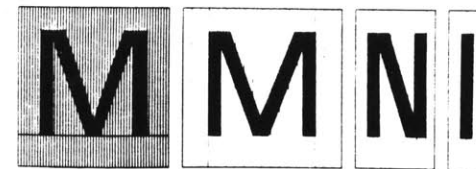


Figure 3j

mechanize the counting of line length and escapement in the Monotype type-casting system.

The Linotype machine, invented by Ottmar Mergenthaler in 1884, cast entire lines of type at once. Text content was input by the operator with a keyboard. With each keystroke, a brass matrix mold of the indicated letter was released and positioned in place until an entire line of matrices was formed. When each matrix was properly spaced, the line was cast in molten lead. To accurately justify a line of text, small wedges were inserted between the matrices to force their separation. (See Figure 3k) A counter was used to measure character widths to determine the available space per line. Each matrix contained two molds to store letters from two different fonts. (See Figure 3l) The characters in these fonts had to be identical or nearly identical in width. This resulted in width distortions in italic letterforms when roman and italic faces were paired. Conventional italic forms were narrower than those designed for the Linotype machine.

In the Monotype machine, developed by Tolbert Lanson in 1887, individual pieces of type were cast to compose a line of text. A perforated paper tape was used to drive the typesetter. In contrast to the Linotype system, the

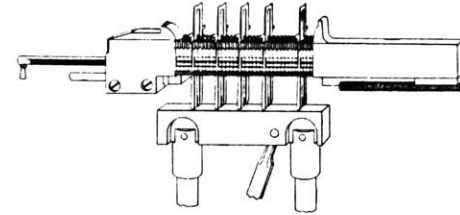


Figure 3k



Figure 3l

typesetter required the use of letterforms designed according to a prespecified width unit system. (See Figure 3m) Consequently traditional letter widths and proportions were altered to accomodate the new technology.

With the advent of photocomposition in the 1950's, typeface design was no longer restricted to the rectangular block of metal type. Drawn letters were photographically reduced and stored on a transparent film strip. The negative film masters were exposed to photosensitive paper or film. Three or four sizes of film masters were typically created by the designer. Alternate character sizes and letter spacing could be created by photographic distortion with the use of a lens system. Therefore deliberate variations in character weight, slope, width, and height could be made from the original set of designs. This flexibility in type manipulation made possible by phototypesetting led to greater typographic freedom and creativity.

In addition, the unit system was refined in second generation phototypesetters. [Ruggles (in preparation)] Unit widths became significantly smaller in size, or higher in resolution, and thus the designer could vary letter widths more freely and make subtle adjustments in spacing. To counteract the effects of light exposure on different parts of the character image, stroke

Unit Value	Row	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	Row		
5	1																1		
6	2	j	f	i	!	:	:	-	j	f	i	!	:	:	-		2		
7	3	c	r	s	e)	('	r	s	t	j	v	°	z		3		
8	4	↓	q	*	b	g	o	?	I	z	c	e	z	s	!	?	4		
9	5	I	9	7	5	3	1	0	.	9	7	5	3	1	0	5	5		
9	6	c	8	6	4	2	8	-	8	6	4	2	8	-	8	6	4	2	6
9	7	x	k	y	d	h	a	x	J	g	o	a	P	F	L	T	7		
10	8	A	f	u	n	.	S	v	y	p	u	n	Q	B	O	E	8		
10	9	D	fl	p	ff	fi	q	k	b	h	d	v	Y	G	R	9	9		
11	10	H	&	J	S	æ	æ	ff	Z	ff	x	U	K	N	10	10			
12	11	O	L	C	F	w	æ	L	P	F	¶	M	Z	Q	G	11	11		
13	12	E	&	Q	V	C	B	T	O	E	A	w	P	T	R	B	12		
14	13	D	A	Y	ff	ff	m	æ	Y	U	G	R	æ	w	V	13			
15	14	K	N	H	ff	ff	X	D	N	K	H	m	&	h	X	U	14		
15	15	æ	æ	¼	¼	½	W	M	-	M	W	%	æ	æ	15	15			
Unit Value	Row	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	Row		

Figure 3m

weight proportions and dense areas, such as join features, were modified.

Third generation phototypesetters incorporate the use of CRT technology and digital methods of storing and processing character data. Character images are scan converted onto the CRT screen on the fly. These images consist of discrete small units composed on a raster grid. As the linear scaling of type became an accepted practice, fewer original sizes were designed, although this technique has not been universally approved of by designers and producers of type. Before the development of digital methods of design and font conversion, typeface designs for CRT phototypesetters were drawn by hand.

With the advent of digital typesetting and the expanded need for digital typeface designs, several systems have been developed to merge the font generation process with computer technology. These systems are used primarily for analog to digital font conversion and font design modification. Analog to digital font conversion is the process of translating existing analog letterform images into digital outline or bitmap format. Outline format represents character shape information as a series of curve control points, connected by straight lines and curve segments that define the contour boundaries of each letter shape. (See Figure 4a) Bitmap descriptions are composed of discrete point pixel coordinates that are either run length encoded or stored as an array of on and off pixels. (See Figure 4b)

Analog source images are scan converted or manually transformed into numerical data. Manual outline processes require the user to mark discrete curve control points along the contour edge of each letter image. (See Figure 4c) Control points are entered and edited by specifying coordinates interactively with a puck or through a programmed listing of coordinate data.

Outline representations of high order continuity are resolution independent. The curve forms typically used are bezier curves, conic sections, or

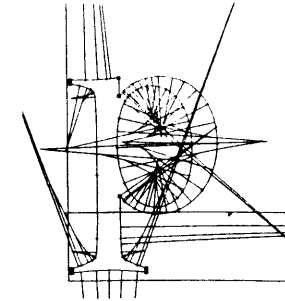


Figure 4a



Figure 4b

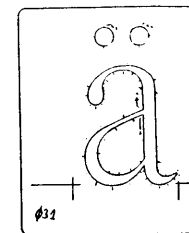


Figure 4c

hermite curves. They can be scaled, rotated, translated, and output to variable resolution devices by altering stroke writing patterns. Bitmap fonts, on the other hand, are resolution dependent and must be created, edited, and stored individually at each desired point size and aspect ratio. Due to the nature of sampled systems, scan conversion algorithms result in quantization error and illegibility, particularly at low resolutions, under twenty lines to the em square. (See Figure 4d on next page) Low resolution bitmaps are designed by hand on paper or with the use of an electronic grid.

The Ikarus system, designed by Peter Karow of URW, incorporates automatic scan conversion correction processing. Inconsistencies in stroke weight, character height, alignment and curve symmetry are normalized and justified without designer intervention. (See Figure 4e) Once adjusted, bitmap designs can be modified with several batch programs. (See Figure 4f) Character heights and widths can be altered independently to expand and condense characters, and letters can be automatically italicized, shaded, contoured, and rounded. Interpolations can be performed to create intermediate weights of a typeface.

Although reducing time and labor by fifty percent over pen and pencil



Figure 4e

Hamburgefons
 Hamburgefons
 Hamburgefons
 Hamburgefons
 Hamburgefons

RRRRRR
RRRRRR

Figure 4f

correction techniques [Flowers 1984], batch produced bitmaps require extensive editing. Faces designed originally for photocomposition are not directly usable. Each character in a digital font must be tailored to the specific requirements of digital output devices. Consequently, skilled lettering artists must edit each letter of each typeface.

An alternate method of correcting bitmap characters is to use the PM Digital Spiral developed by Purdy and MacIntosh. The PM Digital Spiral is an extremely high resolution spiral form or template that can be uncurled and placed along the edge of a bitmap character. A series of curve lengths, angles, and starting points is created and used to correct dropout and smooth aliasing errors. (See Figure 4g)

The Camex Letter Input Processor, adapted for use at Bitstream, Inc., was developed in response to the need for real-time editing and graphic interaction. Outline editing tools include functions to select, insert, delete, move, and constrain individual points graphically. By forming point groups, curve segments and letter parts can be copied, moved, scaled, and rotated to recreate repeating elements, build structurally related letters, and compare similar or identical letterform features. (See Figure 4h)

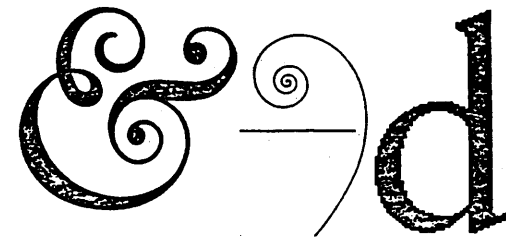
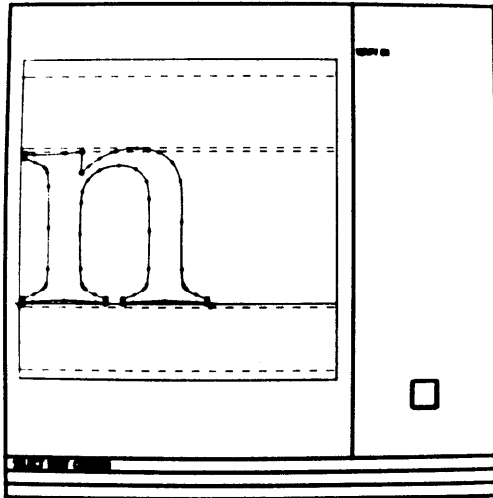
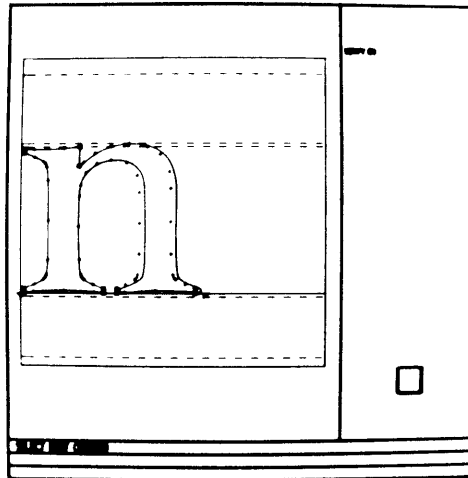


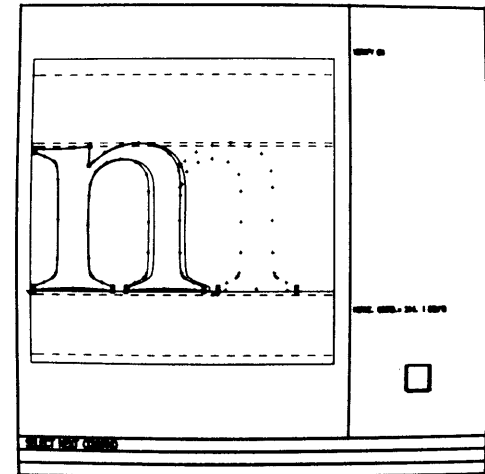
Figure 4g



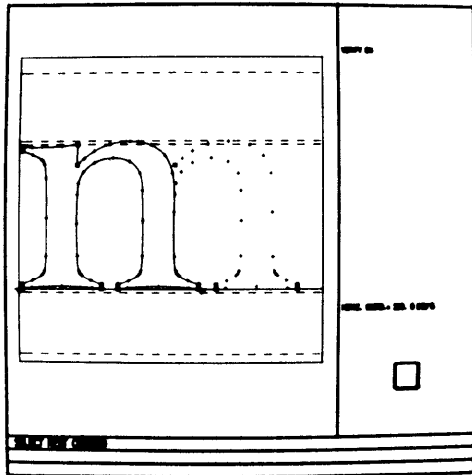
1.



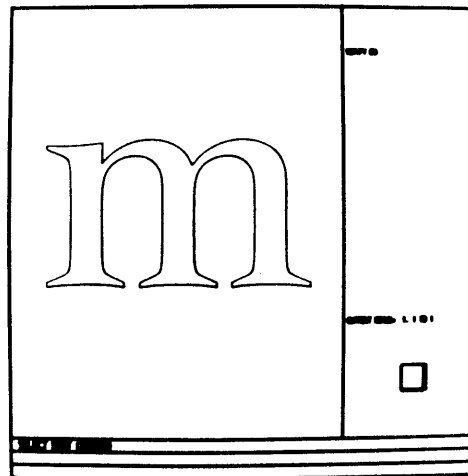
2.



3.



4.



5.

Figure 4h: to create an m from an n

1. begin with n
2. condense the arch of the n
3. copy new arch and position it
4. erase the n
5. the completed m

The outline images from the Camex LIP are input to a Symbolics 3600 Lisp machine where significant design features of letters, referred to as zones, are interactively marked, measured, and named and individual scan lines are adjusted for accurate bitmap reproduction. The zones create an underlying pattern of horizontal and vertical dimensions referred to as a plaid. To automatically generate a series of bitmap fonts, zone values within each letter are constrained to the design features of the control characters in each font. (See Figure 4i)

Typeface Design Systems

Adopting terminology defined by Southall in "Designing new typefaces with Metafont", analog to digital font conversion systems are drafting systems and can be differentiated from systems developed for use in digital typeface design. Drafting systems are used to input and convert already existing typefaces or letter drawings into digital form. Design systems are used to create new typeface images. Generally they provide tools that simulate the use of traditional pen and paper techniques such as sketching and drawing, cutting and pasting, copying, and proofing in an interactive and visual environment intend-

ed to support the fluidity of the creative process. Three design systems will be reviewed here. This discussion will focus on the unique features of each.

ELF is an interactive graphics system developed for typeface design that supports all stages of the design process, from sketch creation to final proofed image generation. Work on ELF began in 1979 by Kindersley and Wiseman at the University of Cambridge. Character images are drawn with a light pen on the display surface and translated by the system into a series of line segments or filled trapezoidal shapes. As the designer modifies images, an internal model of geometrical manipulations is stored and a textual log file is created that contains the sequence of actions performed by the user during each design session. The log file can be replayed or edited to recreate a series of design modifications on viewed images. ELF includes unique techniques for manipulating character spacing based on area computations performed on each letter image to determine its optical center. As the designer edits a letterform feature, the optical center can be recomputed and the "intrinsic width" of each letter recalculated. Character images are automatically updated as the designer proceeds.

IMP, a computer aided design system developed by Carter and Wise-

man on the color Rainbow workstation at the University of Cambridge, takes advantage of windowing facilities to model concurrent design contexts. The designer can move freely among five window areas, each associated with an active level of the design process or problem solving hierarchy. Windows and transparent overlays are created to sketch, smooth, grid, build, edit, copy, compare, and proof letterform images. Characters can be magnified for editing and simultaneously updated in bitmap or outline format in the proof and compare windows. Letters from several different fonts can be selected for viewing. Single bit or grey scale letter images are edited by positioning the cursor and cycling through pixel colors interactively.

At Stanford University, Lynn Ruggles is developing an interactive workstation called Paragon that combines traditional paper-oriented type design techniques and digital processes. Sketches of characters or character primitives can be drawn on the screen and converted into smoothed outline and bitmap representations displayed at varying sizes. The designer works with overlays that function as "translucent sheets of paper" to create and edit each image.

Related Work

The notion of encoding typeface consistencies is not new. Three noteworthy experimental systems have been created to explore the idea of automatically generating and manipulating typeface designs. In contrast to analog to digital font conversion systems and design systems that are used to record character contour shape data in numeric form, each of these systems represents letterform primitives such as individual strokes and letterform structure and each makes use of shape parameters that control letter characteristics such as height, width, stroke weight, and serif designs.

ITSFL, the InTeractive Synthesizer of LetterForms created in 1967 by Mergler and Vargo, was the first computer system developed to produce actual typeface designs. Earlier systems had been applied to the reproduction of outline character images on vector CRT's or with dot matrix plotters using coordinate shape data punched onto batch processed cards. Mergler and Vargo extracted geometric letterform features from enlarged characters and stored them in the computer as straight and curved lines. Design features such as letterform heights, widths, stroke weights, and stroke endings were stored as

parameters and used to modify the geometric data to produce varying typeface designs. ITSLF had both an automatic and a manual mode. The manual mode was used to alter parameter values individually for each letter. In automatic mode, the parameter values for the capital letter 'E' were input and used to calculate the designs of the other letterforms. Mergler and Vargo could generate 24 capital letters with their system. These are illustrated in Figure 4j. They concluded that while it was possible and useful to modify geometric letter designs parametrically, further investigation was necessary.

In 1976, Coueignoux developed an extensive set of rules for describing the consistencies in structure and design within and among Roman printed fonts. His system, CSD, or Character Simulated Design, was used to automatically generate upper and lower case character drawings. Each character was defined by parameters and parameter values. Parts that shared similar or identical sets of parameters were grouped into families of related shapes, i.e. stems, bowls, etc. The common parameters shared by each shape were classed into the following parameter sets: height, thick and thin thickness, horizontal extension, angle, and squareness, and discrete type. Relationships among the parameter values within each parameter family were delineated as rules of

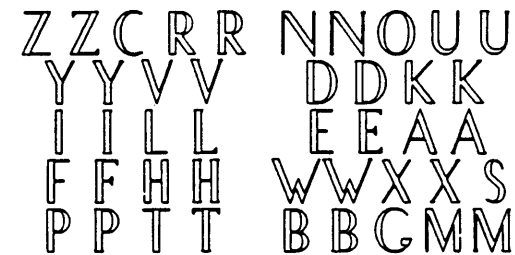


Figure 4j

proportion. The spatial relationships among parts were described by rules of disposition. The user manipulated parameter values to modify the shape of each part.

Coueignoux developed a generative grammar used to automatically synthesize part and letter descriptions, to construct part configurations, and to constrain part joining relationships. To create each letter, parameter values for each family of parameters or for individual parts, and the part locations within each letterform were input in numerical form by the user. Repeating primitive shapes were stored as routines that could be called by each letter procedure. To output a character outline, the primitive routines generated sets of conic curve break points and curved or straight line segments.

In total, forty four letter routines and thirteen part primitive routines were used. The primitives are illustrated in Figure 4k. New parameters could be defined by the user with the use of Coueignoux's grammar. There were roughly 300 part parameters and 250 letter parameters. The number of parameters per primitive varied from 3 to 30 with an average of 10, and the number of values per parameter ranged from 7 to 55 with an average of 27. The parameter values were taken from measurements made on enlarged drawings



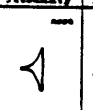

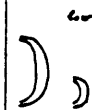
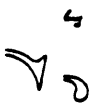


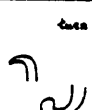



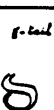
Vertical	Zenogonal	Secondary	Finalize
			
			
			
			

Figure 4k

typeface." [Southall 1985] These parameters describe the height, width, slope, and shape of "virtual" pens and erasers that trace a skeletal letterform input by the designer. (See Figure 4l) Additional parameters define the horizontal and vertical dimensions of letters, letter slope, and a number of serif attributes such as the degree of bracketing, crispness, and length. (See Figure 4m)

The designer creates "symbolic descriptions" of letter shapes by writing programs that specify the pen's motion, the path it travels, and its shape and size. Each letter is drawn with a separate character routine. (See Figure 4n) By manipulating parameter values, it is possible to create a variety of letter shape modifications with each single program. "The designer goes on making changes to the specifications until the marking device produces a shape that has the desired appearance." [Southall 1985]

Realizing the limitations of pen-defined shape parameters for reproducing the subtle variations in contour detail characteristic of printed letterforms, Metafont was modified to include outline drawing routines. By incorporating programs that can express bezier curve control points and slope descriptors in the Metafont language, Knuth was able to retain the pen-meta-

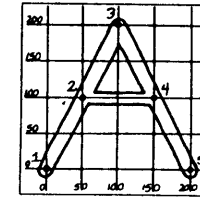


Figure 4l

The x-height and the heights of ascenders and descenders can be independently specified.

A 'slant' parameter transforms the pen motion, as shown in this sentence, but the pen shape remains the same. The degree of slant can be negative as well as positive, if unusual effects are desired. Too much slant leads, of course, to letters that are nearly unreadable. Perhaps the most interesting use of the slant parameter occurs when Computer Modern Italic fonts are generated without any slant.

Figure 4m

```
x1=0; y1=0;
x2=50; y2=100;
x3=100; y3=200;
x4=150; y4=100;
x5=200; y5=0;

penshape=circular;
input pen;

20 draw 1..3;
20 draw 3..5;
15 draw 2..4;
```

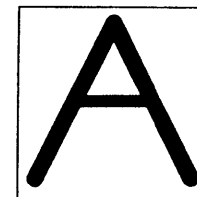


Figure 4n

of Baskerville, Bodoni, Cheltenham Medium, and Times Roman Bold and used to output drawings of each font on a vector screen or with a 200 dot per inch electrostatic printer. At the time of Coueignoux's writing, only the primitive parameter values were saved in the computer. Letter values and lists of parts had to be re-entered. Output characters were photographed and reduced to size.

The most well known system originally developed for typeface design is Metafont created by Donald Knuth at Stanford University. Metafont is a programming language used for making character shape specifications. It is "not a graphic-mode design system in the traditional sense." [Southall 1985] Specifications are issued in numeric and symbolic form and are used to drive a marking device that draws graphic character shape images. [Southall 1985]

Knuth sought to capture the "meta-characteristics" of a typeface or the kernel of design principles used to vary letter drawings throughout a series of related font designs. Image descriptions are produced by setting font wide shape parameters. "In our terms, a meta-typeface is a typeface design in which the stylistic and functional visual attributes of the design have parameters associated with them. Each setting of the parameters defines a different

phor. Contour edges are specified with two pen strokes, each one pixel wide. However, the outline drawing routines do not make use of the concept of "meta-ness", except where it can be applied to changing the point size of letters with linear scaling techniques.

Richard Southall, a designer who worked closely with Knuth and with Metafont, sums up the essence of "a-priori meta-design" in the following quote.

In a meta-design for a typeface, the specification for the character shape incorporates specifications for the changes in the shape that occur as a consequence of changes in the typeface parameters. In a symbolic specification for a character shape, these specifications will be in the form of functions that relate features of the character shape to values of the typeface parameters; and we can describe as *a priori meta-design* a design method in which these functions and their coefficients are specified explicitly by the designer...Doing *a priori* meta-design in a way that ensures the eventual production of technically satisfactory character images for all reasonable combinations of typeface parameter settings requires the same thing that successful symbolic-mode design requires: explicit formulations of the rules that govern the visual interactions between the elements of character shapes. [Southall 1985]

Southall stresses that

...we do not at present have the theoretical basis for predicting the shapes of technically satisfactory typeface characters on which a successful symbolic-mode design system could be built. It is true that once the design of a technically satisfactory typeface has been completed, exact definitions of the shapes of all the characters in it do exist: but these definitions are graphic rather than symbolic, and the routes by which they were arrived at cannot be restated explicitly in algorithmic form. [Southall 1985]

The unique contribution of this thesis is the notion of automatic letterform derivation. As stated in the introduction, it is hypothesized that letter designs can be derived from the set of control characters. In order to understand the role these letters play in the design process and the problem solving activity of the designer, this Chapter will be used to describe the derivation of letterforms and general techniques that can be applied to creating a preliminary set of consistent characters.

Letterform Derivation

Figure 5a on the next page illustrates and lists the part and letter attributes contained in the control characters h, o, v, and p. The o contains the primary round letter characteristics, the h the primary square letter characteristics, the p the round and square letter characteristics combined, and the v the diagonal letter information. Attributes and values extrapolated from the control characters are used to constrain the design of other letterforms.

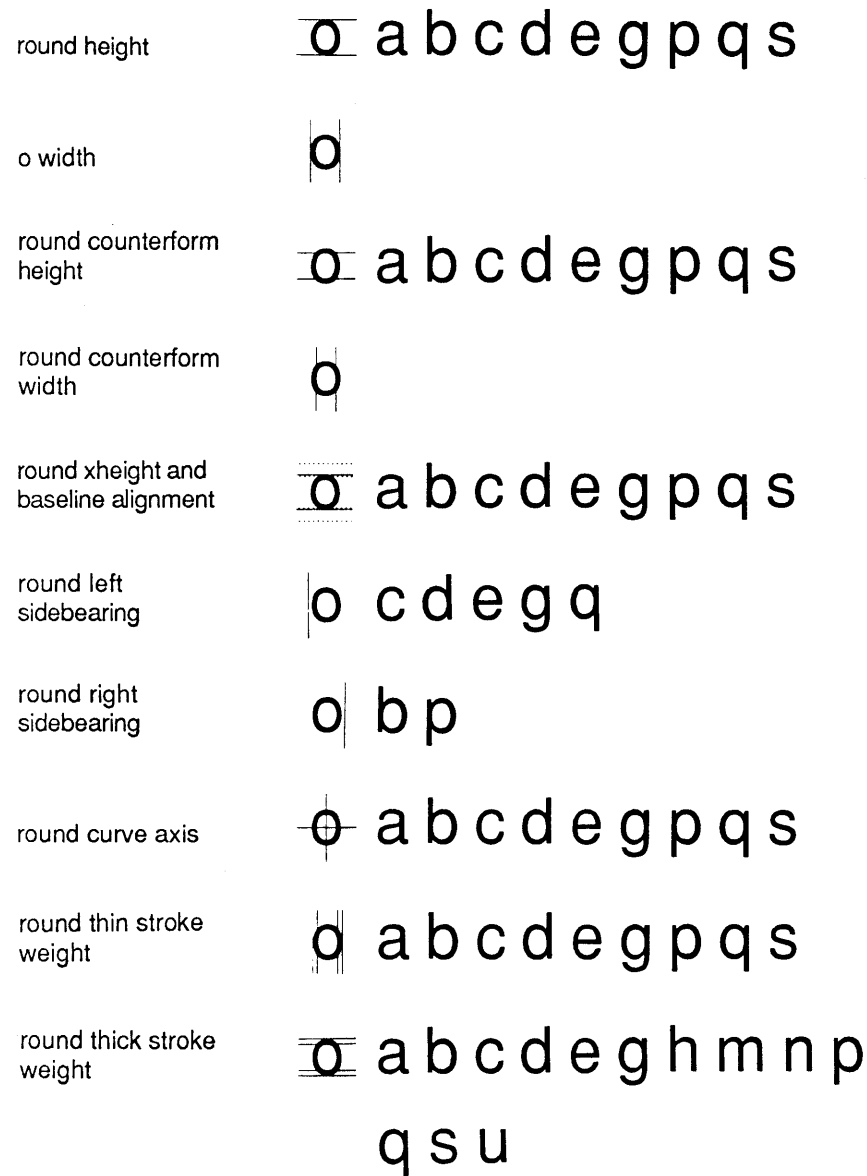
The parts of each control character are illustrated in Figure 5b. As seen in Chapter Two, a part can be defined as a set of composite properties that function together as an independent unit. Parts that contain shape attri-



Figure 5b

LETTERFORM DERIVATION

FIVE



LETTERFORM DERIVATION

square ascender height h b d k l

square width h n u k x z

square counterform height h n u m

square counterform width h n u m

arch height h n m r

square ascender alignment h b d k l

square baseline alignment h b d f i k l m n r u v
w x z

arch alignment h n u m r

FIVE

square left sidebearing | h b i k l m n p r u

square right sidebearing | h d g i l j l m n q u

vertical stroke weight || h a b d f g i j k l m
n p q r t u

LETTERFORM DERIVATION

descender height	p q
square/round width	p a b d g q
square/round counterform width	p b d g q
square xheight alignment	p i j m n q r u v w x y z
square baseline alignment	p q

FIVE

oblique height (xheight)	v i m n r u w x z
v width	v x y
v counterform height	v
v counterform width	v y
oblique left and right sidebearing	v w x y
v stroke weight	v y

butes and attribute values identical or nearly identical to those found in the control characters can be directly derived. These parts are illustrated in Figure 5c on the next page. The remaining parts either do not exist in the initial source subset or are created as modified source part primitives. Although they share attributes and values in common with the control characters, they require more decision making to determine their actual contour shape boundaries and design features. Letters are related to the control characters in the tree illustrated in Figure 5d depending on their degree of similarity in shape or structure, on the number and kind of decisions that need to be made to define their shape attributes and values, and on the complexity of the graphic operations used to create them.

Parts such as the xheight stem contain the same shape attributes as those found in the ascender stem of h, that is, they are both straight strokes, but their height values differ. Likewise, the slope and the thickness of the oblique strokes of the w, x, and z differ from the v. These values cannot be defined without knowing the overall width and design of each letter. The length, and position of horizontal parts such as the crossbars are unknown given the control characters. The slope and joining characteristics of the legs of the k

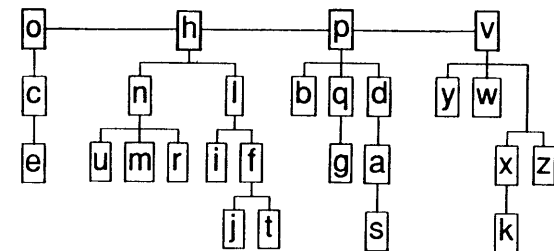


Figure 5d

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

are completely undefined by any letter in the lower case alphabet.

Other parts are similar or stylistically consistent, but not identical, in shape to the parts found in the control characters. Their sets of shape attributes and/or their values differ. For example, the arch of the m can be derived from the n; it has many of the same features such as the thin and thick stroke weight, the top arch alignment value and the counterform height value, but its curve gradation differs due to the fact that its width is condensed. Similarly, parts of the c appear nearly identical to the o, but their actual curve gradation is unknown due to the fact that the top is condensed and the bottom piece is extended for visual balance. Whether or not the terminal angle is oblique or horizontal is also not known given h, o, v, and p. This is also the case in f, a, g, t, s, and y. There is no design information that can be used to determine the slope of the s. The widths of f, j, t, and r are not given.

As more letterforms are created by the designer, unknown design features, shape attributes, and values are defined. This information can be used to create, or solve, other letterforms. For example, once the curvature of the upper terminal of the f is designed, it can be mapped to t and j. Likewise a and g share similar design features. Although there is no required order in which

letters are created, the pattern or sequence does constrain the "solution space" or range and number of design problems that remain to be solved. A path through the font is created from letter to letter as design decisions are made and carried over.

An additional factor that may influence the order in which letters are created is the need to view characters grouped together in words, phrases, paragraphs, and control strings as soon as possible. This gives the designer an idea of the texture and color of the text and its legibility. Figure 5e lists the design attributes Matthew Carter studies in the control characters h, o, v, and p. At the bottom of this list are two sets of letterforms Carter creates in addition based on a) the complexity of their structural form and b) their use or frequency of appearance in text. With thirteen lower case letters and four upper case forms, Carter is able to visualize his design as seen in Figure 5f. Thus a time element is involved.

Heuristics

The operations described below are general procedures that can be used to create a set of rough sans serif letterforms from the control charac-

LETTERFORM DERIVATION

FIVE

outer sides/fit straight h i m
 round o
 straight/round v
 oblique p

stems 1 i
 2 h
 3 m

vertical proportions xheight v h
 round o
 ascender h
 descender p

frequent/curves a t s

inner counters straight h
 round o
 straight/round p
 oblique v

arches/joints to stem h m
 p
 a

weights/stress straight h
 round o s
 straight/round p
 oblique v

serifs horizontal base h i m
 top h i m p
 foot p
 xheight v
 s

structural
h o p i v a m t s

frequent or easy
e n d u l b r y

Figure 5e

LETTERFORM DERIVATION

FIVE

Font 0669 Alisal - Version 2 Thu 13 Feb 1986 9:04 a.m.

OOHOHHHOOnOoOHnHoHoononnnooHH
AAAHAOARAAAbAeAgAhAiAmAnAoArAsAuAv

HAHHHOHRHaHbHeHgHhHiHmHnHoHrHsHuHv
OAOHOOORoaObOeOgOhOiOmOnOoOrOsOuOv
RARHRORRRaRbReRgRhRiRmRnRoRrRsRuRv
aAaHaOaRaaabaeagahaiamanaoarasaauav
bAbHbObRbabbbebgbbibmbnbobrbsubv
eAeHeOeReabeeegeheiemeneoereseuev
gAgHgOgRgagbgeggghgigmngogrgsgugv
hAhHhOhRhahbheghhhhhmhnhohrhshuhv
iAiHiOiRiaibieigihiiiminioirisuiiv
mAmHmOmRmambmemgmhminmmnmomrmsmumv
nAnHnOnRnanbnengnhninmnnnonrnsnunv
oAoHoOoRoaoeogohoiomonooorosouov

rArHrOrRrarbrgrhrirmrnrorrsvuv
sAsHsOsRasbsesgshsismnsosrsvuv
uAuHuOuRuabueuguhuiumunuourusuuuv
vAvHvOvRvavbvevgvhvnmvovrvsvuv

AHORabeghimnorsuv

BITWOCKY - an original composition by Regis McCarter

Amerigo roams murmurous seas his ambergris seahorses see
Rogue samurai as gruesome as a robber sagamore vamooses
Osage maharishi measures his massive samovar ambiguous
As sahib or memsahib rummage umbrageous Hamburg rooms agree
Our mauve irises gamboge mimosas brush a sham summerhouse
Rose shrubs high over grass submerge some bugs or grubs
Habsburg margrave goes overseas his bimbo houri huge bosom
Herbiverous moose grim beaver rare marabou bogus grebe arrive
Origami horseshoes embarrass our average vigorous greaser
Remember somber suburbia borough barbarism garbage horror
Ambushes mob messenger as sober shamus absorbs rough sourmash
A vague gossamer miasma hovers over morose Omaha reservoirs
Obsessive he hogs mushroom gumbo various sesame mousses give
Him serious regressive seborrhea rash his bum behavior ushers
Aggressive hearse remorse as grave rabies virus erases his visage

Amerigo roams murmurous seas his ambergris seahorses see
Rogue samurai as gruesome as a robber sagamore vamooses
Osage maharishi measures his massive samovar ambiguous
As sahib or memsahib rummage umbrageous Hamburg rooms agree
Our mauve irises gamboge mimosas brush a sham summerhouse
Rose shrubs high over grass submerge some bugs or grubs
Habsburg margrave goes overseas his bimbo houri huge bosom
Herbiverous moose grim beaver rare marabou bogus grebe arrive
Origami horseshoes embarrass our average vigorous greaser
Remember somber suburbia borough barbarism garbage horror
Ambushes mob messenger as sober shamus absorbs rough sourmash
A vague gossamer miasma hovers over morose Omaha reservoirs
Obsessive he hogs mushroom gumbo various sesame mousses give
Him serious regressive seborrhea rash his bum behavior ushers
Aggressive hearse remorse as grave rabies virus erases his visage

Figure 5f

ters h, o, v, and p. Related letters can be generated through a series of transformations. Each transformation consists of a set of operations applied to a source letterform or part to derive a destination character. These operations differ in number and kind depending on the degree of similarity between two forms. Letters can derive shape information from multiple sources or from a single source letterform. In addition, each letter can contain its own "slots" of information.

As seen in the preceding discussion, some letters can be more explicitly described in relation to the control characters than others. Therefore these procedures vary in their degree of specificity. They do not represent practices employed by each and every designer nor can they be used to generate a final set of letter drawings.

Letters such as q, b, and d and n and l are relatively easy to solve. Most of their attribute information is given and in sans serif faces their forms will not tend to vary greatly from the control character shapes. Q can be created by duplicating the descender shape in p and inverting the bowl. B and d contain the ascender stem of h, as does l. (See Figure 5g) Common variations of the b form are shown in Figure 5h. As mentioned, the height of the

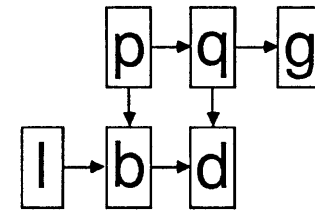


Figure 5g



Figure 5h

xheight stems can be determined using the square xheight alignment value found in p or v for n and i. The dot shape of i and j are generally round or rectangular shapes and are top aligned at the ascender alignment height.

U contains parts found in the n but they are rotated 180 degrees. (See Figure 5i) Its shape will generally not be exactly identical to the n due to its orientation and visual balance. An alternate u form is shown in Figure 5j. An initial form of m can be generated by modifying the n arch width. Although their curvature and stroke weight appear related there are subtle differences in their final form. Attention must also be paid to the joining characteristics of the two arches. In addition, the vertical stem weights of m will tend to be thinned to appear visually consistent. R can be created by modelling the n arch curvature, but in Helvetica its shape is based on b. The terminal shape of r needs to be determined as does its width. R's tend to vary in sans serif faces more often than n and u. (See Figure 5k)

As seen, the c can be created from the o, but decisions are required to define its curvature, its width, and the length and characteristics of its terminal endings. (See Figure 5l) The top right shape of e will tend to be extended slightly in comparison to the c to join the crossbar. The bottom terminal of e

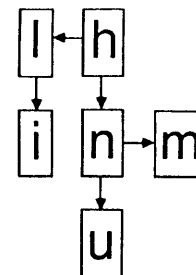


Figure 5i



Figure 5j



Figure 5k



Figure 5l

the bottom portion of the g can vary although it tends to be visually related to the a hat. The bowl of the g is generally condensed in the vertical direction due to the presence of the tail and the visual complexity of its design.

F, j, and t often have related terminal features. (See Figure 5s) Figure 5t shows an exception in the typeface Futura. Generally f and t are similar in width. Their horizontal crossbars are aligned at the xheight square alignment and they will be identical or nearly identical in thickness to the crossbar in e.

The oblique letter category contains the most differentiated shapes. V, y, and w are related in structure. The y can be created by extending the right stroke of v, although its terminal shape is undefined. (See Figure 5u) The w can be treated as two v forms that are joined, although its strokes are more oblique and therefore thinner in width. (See Figure 5v) X can be created by determining the slope of its legs in relation to its overall width. Its stroke weight is also a function of slope. The bottom legs of the x will generally extend beyond the top arms for balance. The slope of the diagonal stroke in z can also be defined in relation to the letter width. Its horizontal stroke weights are known, although its join shaping can vary. (Figure 5w)

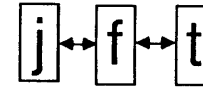


Figure 5s



Figure 5t



Figure 5u

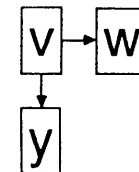


Figure 5v

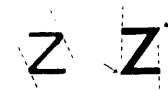


Figure 5w

will appear visually similar to c although it may be extended further to the right due to its visual interaction with the crossbar. (See Figure 5m) The height of the crossbar can vary and its slope can change. Although s is a round letter, it is often more related to the a in its top and bottom curvature, its bowl forms, and its stroke weights in faces where a double story a is present. (See Figure 5n) The slope of the diagonal is s differs from typeface to typeface and differs in light and bold faces. (See Figure 5o) The bottom stroke of s will tend to be extended to the left and the top stroke will be indented for visual balance. Generally, the top bowl shape will be smaller or more condensed than the bottom bowl shape.

The two common forms of a are shown in Figure 5p. Single story a's can be derived from q. Double story a's are related to d in structure, but are highly variable in shape. Although its top hat shape may model o, there are few constraints on its curvature. The bowl form is unpredictable as are its terminal endings given h, o, v, and p. (See Figure 5q) There are also two common forms of g. (See Figure 5r) However, in sans serif faces, the former is more prevalent. This g form can be partially determined using the q but its descender stem depth and the shape of its bottom curve are not known. The shape of

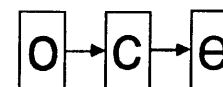


Figure 5m

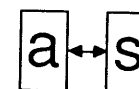


Figure 5n



Figure 5o



Figure 5p



Figure 5q



71 Figure 5r

The system presented in this thesis is a tool for digital typeface design. As such, it is intended to be used to create new typeface images. Like other design systems, it provides interactive tools and a visual design environment. The designer works directly in graphic mode to create and modify letterforms. Letters can be interactively edited and viewed on more than one level. Contour curves can be altered and parts can be translated, inverted, reflected, aligned, and joined to form higher level part groups. The display screen is divided into several areas for proofing and comparing letters individually and in text strings.

At the same time, this system is also an attempt to integrate automation into the design process. Software is used to automatically derive letterforms and to propagate design changes. However, the task of automating the derivation process is compounded by the diversity of typeface styles and font dependent variables. Therefore a highly regular and small set of letter shapes was tested in order to determine representations and operations useful for automating the initial construction of letterforms and for generating a set of consistent designs. H, n, u, and m were chosen because they contain sets of repeating primitives that differ in orientation, alignment, shape, and size. H is

the control character used to derive n, u, m, i, and l. With the addition of p, the letters b, d, and q can also be created. The control character images are from the typeface Bell Centennial designed by Matthew Carter. (See Figure 6a)

Representation

Letterforms are represented, at the lowest level, as discrete point coordinates which, when connected with cubic spline curve segments, describe the contour boundaries of each letter shape. Outline representation was chosen because it provides a means of reproducing the subtle and complex curves that characterize most typeface designs, and because it is relatively easy to perform geometric transformations such as scaling, rotation, and translation at varying resolutions. Outlines do not, however, adequately represent letterform structure. In order to create letters as built forms, it is necessary to extrapolate or define, at a higher level, their topological features. These features include the structural components or parts, their means of combination, and their spatial configuration.

The parts of each letter were identified in Chapter Two. They are



Figure 6a

treated as modular and connected primitives that function as building blocks. Part attributes describe the position, orientation, slope, alignment, size, and shape of each part. Letter attributes define the position, orientation, alignment, size, and shape geometry of each part configuration. The shape attributes of parts and letters are defined by their contour edges. Part contours can be subdivided into individual edges or sets of edges in order to reference a particular feature such as a join or to manipulate a design attribute such as height or stroke weight. Similarly, parts can be grouped together to form higher level structural units that are duplicated in exact or slightly modified fashion in structurally related letters.

Each part is classified according to its type and associated with a set of shape specifications and procedures used to generate its contour description. Part objects are stored in a part library organized into part type classes and subclasses and letter objects are stored in a letterform library consisting of letter classes. Parts are copied, manipulated, or created by the computer to generate and build each letterform. Letters are constructed with the use of rules and procedures that specify each component part type and its position, orientation, and alignment within each letter. Part shape outlines are joined

and recontoured.

Letterform Generation and Propagation

The control characters contain the initial source attributes and values used to constrain the design of other letterforms. Characters such as h and l that contain identical part shape attributes can be created with duplicate part objects. Other letters such as b, d, and q contain identical or similar bowl shapes that differ in orientation. These parts can be copied and rotated or inverted to create each letter. To modify the shape, curve gradation, and size of existing parts, contour edge features need to be manipulated. These features include the curve slope and size descriptors. For example, to generate the arch of an m the n arch can be manipulated by scaling its top and bottom curve contour edges in x independently of one another. These two edges are modified independently in order to maintain their width relationship when they are joined with the right stem. Therefore, when altering or creating part objects, constraints on certain design features have to be satisfied in order to derive consistent designs.

To create part shapes that are not defined by the designer, contour

edge descriptions must be generated. This requires more explicit attribute representation and a complete set of shape specifications and procedures. For example, the slope of the oblique strokes of an x may be specified by determining x's width. Its stroke weights can be defined as a function of the oblique to vertical stroke thickness ratio. Therefore, with each new design of the control characters, a unique x will be created. Creating these parts has not been tested in this software.

In the current implementation what can be referred to as "one way" letter transformations are implemented. In other words, procedures have been written to modify an existing n or h arch to create an m but not to generate an n or h given an m. These procedures could be referred to as "two way" transformations and, to extend the system's versatility, they could be included. At present, the order in which letters are created can, and does, influence the generation process.

Rather than editing each shape graphically or in a program, the designer can automatically propagate changes to part and letter contours throughout a font to all letters that contain like parts or to user specified letters. Thus curve relationships can be recreated. Changes can also remain local to the ed-

ited letter, thereby preserving the inherent nonuniformity of repeating part primitives.



The software package described in this Chapter is called "**abcdefg**" (pronounced ab-kuh-def-gee), or "**a better constraint driven environment for font generation**". Although it is not truly a constraint driven system to date, constraint representation software has been written by Rick Poyner, an undergraduate student working on this project. The abcdefg package represents a testbed and provides a foundation for the future integration of Rick's work. All demonstration software is written in the C language.

Hardware Environment

Abcdefg was developed on an IBM XT personal computer with 512 k of core memory and two 10 megabyte hard disks running under the MS-DOS operating system. The display unit includes a high resolution red, green, and blue color monitor and an experimental graphics board that supports an 8-bit frame buffer with a visible area of 640 x 480 pixels and an invisible area of 640 x 336 pixels and graphics functions. The display architecture is called YODA and was developed by IBM. The primary input device is an optical mouse with three buttons.

Screen Layout

The screen space is divided into four working areas and one area devoted to menu display. (See Figure 7a) In the central area, a 360 x 360 pixel area called the Em Square is provided for creating and manipulating letter shape images at large sizes. Four vertical alignment lines are displayed in the Em Square: the descender line, the baseline, the xheight line, and the ascender line. The left and right side bearing lines are also displayed. The Scaled Letter space is a 90 x 90 pixel area used to display a scaled version of the current letter or part being created or edited in the Em Square. This letter is displayed at one quarter of its original size. The Text area along the bottom of the Em Square is an additional viewing space provided for scaled text input. This space is 230 x 94 pixels. The space labelled View is used to display part and letter libraries. Part and letter library images are also scaled to one quarter of their original size. The View space is a 182 x 365 area.

User Interface

The user can freely move to any area of the screen and view and edit letter images on several levels as mentioned. Three working modes are

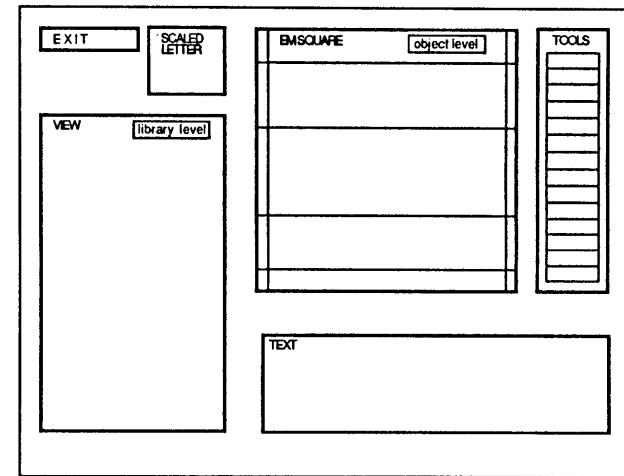


Figure 7a

available. They are: the interactive mode, the automatic mode, and the propagation mode. Modes are selected using a pop up menu that is displayed at the cursor location. A flow chart is shown in Figure 7b.

Interactive Mode:

In interactive mode, letterforms are created and edited graphically and can be viewed on all levels of the letter primitive hierarchy within the Em Square. Curve points can be moved, aligned, inserted, and deleted, spline curve segments can be interactively modified, and parts, part groups, and letterforms can be scaled, moved, inverted, and rotated. Individual parts can be aligned and joined. These techniques model traditional cut and paste practices. The default mode allows the user to directly select and move displayed level objects. Selected objects are highlighted. All editing and creating functions are performed with the middle mouse button.

Level changes are made by selecting the level at which objects are to be displayed. This is accomplished by toggling the level tag situated along the top of the Em Square area. The switch is toggled using the left and right, or up and down, mouse buttons respectively. (See Figure 7c) As the levels change,

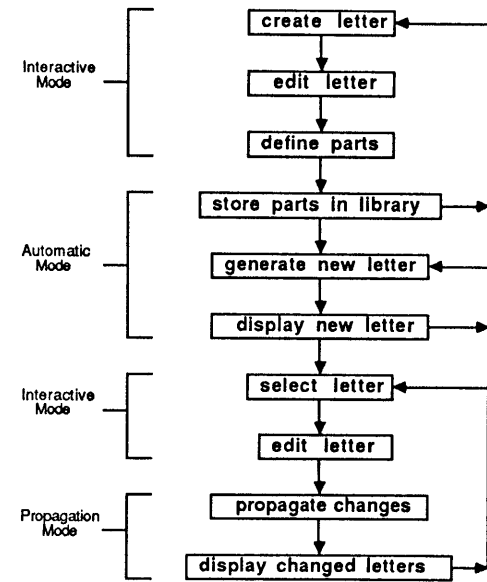


Figure 7b

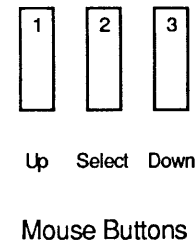


Figure 7c

points, curve segments, part, part group, and letter images are displayed along with the appropriate set of menu buttons listing the tools available at each level. To change functions, the user simply selects the desired menu buttons. Menu buttons are highlighted to ensure proper visual feedback. Menu functions are selected using the middle mouse button.

At any time during the editing or creating of objects, a scaled version of the active image can be displayed. This scaled image is updated when the user moves the cursor into the Scaled Letter area and presses the middle mouse button. This image is positioned within the Scaled Area relative to the origin point of the box to correspond to the position of the large image in the Em Square. (See Figure 7d)

Text is input by moving the cursor into the Text space and also pressing the middle mouse button. Letters are displayed as the user types at the keyboard. Each letter is properly spaced within its given set width.

Part and letter library images are displayed in the View area of the screen. Letters are listed in alphabetical order, and parts are organized according to type. Alternate versions of the same part or letter are displayed together. As each new letter or part is created, image libraries are automa-

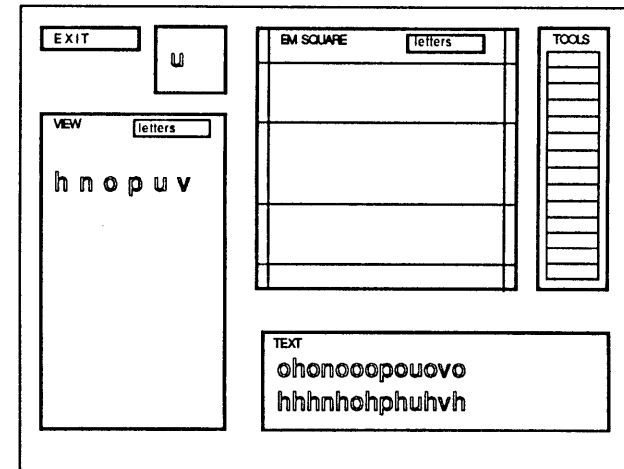


Figure 7d

tically updated and displayed. As in the Em Square, the user toggles a level tag situated along the upper right edge of the View space with the middle mouse button to alternate between part and letter libraries. Library images can be selected for editing and propagation.

Side bearing and vertical alignment lines can be interactively moved.

Automatic Mode:

In automatic mode the user selects letters to be generated. As each letter is created, it is displayed in the View area where it can be selected for editing. Automatic processes are discussed below.

Propagation Mode:

Propagation of changes to curve contours can proceed globally to all letters that contain duplicates of the modified part or to user specified letters. To propagate a change globally, the global button is selected on the pop up menu. User specified letters are selected from existing designs displayed in the letter library. As changes are made, each letter and/or new part is displayed in the View area.

Data Structures

Letterforms:

Each individual letterform is stored as a data structure that contains a list of pointers to its part instances and a pointer to the list of spline curve points that describe its contour shape, called an object list or object. (See objects) (See Figure 7e) In addition, the height, width, top and bottom alignment values, and left and right sidebearings are stored.

```

struct _letters
{
    struct _parts      *partlist;
    struct _objects   *object;
    int               height, width;
    int               top_align, bottom_align;
    int               left_sidebearing,
                    right_sidebearing;
}

```

Parts:

The part structure also contains an object pointer to its list of contour curve points. In addition a type or name variable that corresponds to the class the part belongs to is stored along with height, width, orientation, rotation, top

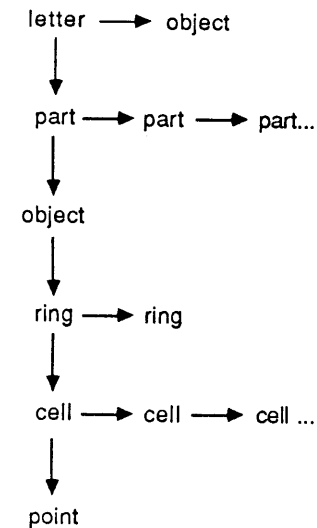


Figure 7e

and bottom alignment, and origin x and y attribute variables. A master part pointer is also stored. (Master parts are explained in the section entitled Defining Parts)

```
struct _parts
{
    struct _parts    *masterpart;
    struct _objects *object;
    int              type;
    int              height, width;
    int              orientation, rotation;
    int              top_align, bottom_align;
    int              origin_x, origin_y;
}
```

Objects:

Each part and letter structure contains a pointer to an object. An object is one or more two way linked lists of cubic spline curve control points used to draw the spline curve contour boundary of each part or letter. Each list is called a ring. Letters such as n have one ring or one continuous edge; letters such as b have two rings, one used to enclose the inner counterform and one to enclose the outer edge. Rings are lists of cells. A cell stores a pointer to a point coordinate structure called a point. Ring, cell, and point structures are

briefly described below. Most of the operations performed on parts and letters, in this software, are operations on object lists.

```
struct _objects
{
    struct _rings    *ring[2];
    int              numrings;

    int              offset_x, offset_y;
    int              extent_x, extent_y;
}
```

Rings:

The ring structure contains a pointer to the first cell in the curve point list and the number of cells in the list.

```
struct _rings
{
    struct _cells    *first_cell;
    int              numcells;
}
```

Cells:

A cell contains a next and previous pointer and a pointer to a point structure.

```
struct _cells
{
    struct _points    *point;
    struct _cells     *next, *previous;
}
```

Points:

The point structure is used to store the x and y point coordinates and the type of control point. There are two possible control point types: straight and curved. The structure also contains a `crd_flag` used to indicate whether or not the coordinates are absolute or relative, an `if_changed` flag used to determine if a coordinate has been changed, and an `if_joined` flag used during the letter construction process.

```
struct _points
{
    int          coord_x, coord_y;
    char         type;
    int          crd_flag;
```

```

        int          if_changed, if_joined;
    }

```

Letter Library:

The letter library is used to store all letter images. It is organized into classes and instances. Theoretically, there would be a letter class for each different letter in a font or typeface but in this software there are only eleven: h, n, u, m, i, l, p, d, b, and q. Each letter class contains letter instances, or different versions of a letter that the designer makes and wants to save temporarily. (See Figure 7f)

```

struct _letterlibrary
{
    struct _letterclasses  h_class, n_class, m_class, u_class,
                          i_class, l_class, p_class, d_class,
                          b_class, q_class;
}

```

Letter Classes:

Each letter class structure has a pointer to its list of letters and a number of letters variable.

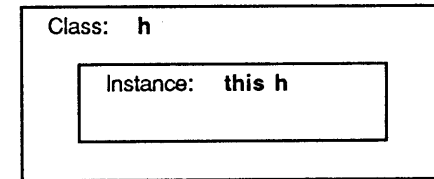


Figure 7f


```

struct_letterclasses
{
    struct_letters    *letterlist;
    int              num_letters;
}
    
```

Part Library:

The part library is organized into master part classes, subclasses, and instances. Three class structures are explicitly stored: the stem class, the bowl class and the arch class. These correspond to the part primitive families or part types discussed in Chapter Two. Each subclass represents a particular kind of bowl, stem, or arch. For instance, there is a subclass of ascender stems and a subclass of xheight stems within the stem class. The ascender stem class contains all ascender stem master part instances. Parts stored in the library are master parts. (See Figure 7g)

```

struct_partlibrary
{
    struct_partclasses    stem, bowl, arch;
}
    
```

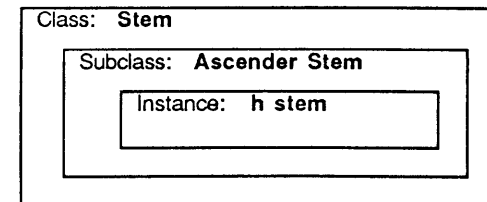


Figure 7g

Part Classes:

Each part class data structure contains an array of pointers to part subclass data structures and its number of subclasses.

```
struct _partclasses
{
    struct _partsubclasses *subclass[10]
    int subclass_number;
}
```

Part Subclasses:

A part subclass data structure stores a pointer to a list of master parts, the number of master parts, and a name variable.

```
struct _partsubclasses
{
    struct _parts *masterpartlist;
    int name;
    int masterpart_number;
}
```

Vertical Alignments:

The square, round, and arch alignment letter values are stored in the

following data structure. These values are referenced when positioning parts and letters.

```
struct _alignments
{
    int          ascender_square, ascender_round;
    int          xheight_square, xheight_round,
                xheight_arch;
    int          baseline_square, baseline_round,
                baseline_arch;
    int          descender_square,
                descender_round;
}
```

Sidebearings:

Square, round, and oblique letter sidebearing values are stored. Each letter can also have its own left and right sidebearing values.

```
struct _sidebearings
{
    int          square_sidebearing,
                round_sidebearing,
                oblique_sidebearing;
}
```

Driver Program

The driver program is the main control loop. It is used to determine and maintain the current cursor location and state, the current workspace, the current object, the current working mode, the current tool, and the current working levels in the View and Em Square workspaces.

Creating Letterforms

The system provides standard outline generation tools for creating letter shapes. These tools include techniques for marking cubic spline curve control points along the contour edge of each letter image. To create a letter, the user interactively selects points on the screen. (See Figure 7h) These points are inserted into an object list. There are two possible spline point types: straight and curved. When a point is created, a straight or curved point marker is displayed. Once the points are entered, an object can be drawn on the screen as an entire closed contour or individual curve segments can be drawn or erased. A sample letter h is shown in Figure 7i. The spline curves pass directly through the straight and curved point markers thus facilitating visualization and ease of curve manipulation. Entered points are edited until the

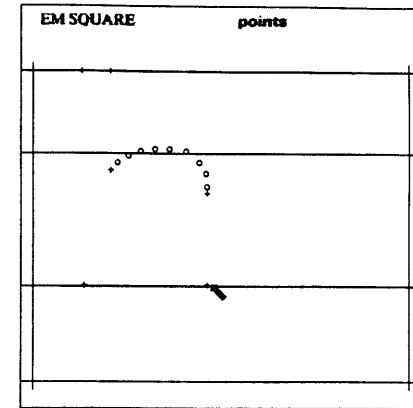


Figure 7h

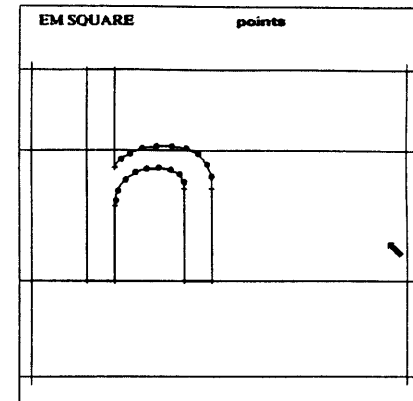


Figure 7i

character shape appears correct.

An alternate method of creating the control characters allows the designer to draw part primitives, position and align them in relation to one another, and recontour them to form higher level structural units or letters.

When a letterform is saved, its height, width, and left and right side-bearing values are calculated. Letter and part objects exist within an imaginary box that extends from the lower left origin point to the upper right coordinate. Contour points are stored relative to the x and y origin. Each letterform is then stored as a set of part primitives. Stored letters can be saved in files and loaded from files. The original set of control characters from the Bell Centennial Typeface were input with these tools.

Defining Parts

Due to the fact that this software does not include feature recognition tools that could be used to extrapolate a letter's parts given its contour shape, the user interactively specifies part boundaries. (See Figure 7j) To define a letter's parts, control points that delimit vertical, horizontal, curved, and oblique shape primitives are selected. These primitives must correspond to the

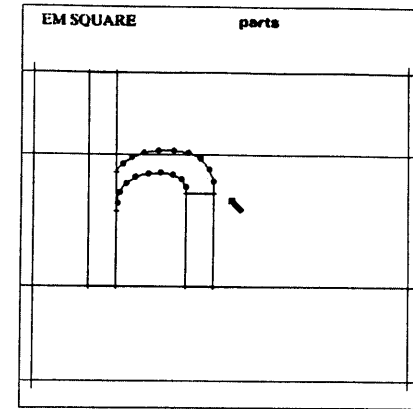


Figure 7j

parts specified by the system to construct other letterforms during the automatic construction process. At present, these points must lie along a horizontal or vertical part boundary in order to facilitate proper recontouring once the pieces are joined together. After the part boundaries are specified, software is used to break up each letter contour list into part object instances.

These parts are stored in the part library as master parts. Master parts are copied by the computer and used to create other letterforms. They are permanent data records and cannot be altered, although they may be deleted from the library by the system or by the user. Part instances used to create letters automatically are copied from master parts. These instances are independent data objects, i.e. the xheight stems used to build the n and u are not the same object. This is because the designer may wish to modify the u stem and not the n.

Editing Letterforms

As mentioned, letterforms can be interactively manipulated on all levels of the object hierarchy. Objects to be edited are first selected and then altered.

Points

- Insert Point: Insert point on a contour edge between two selected points.
- Delete Point: Delete a point.
- Edit Type: Change the type of point from square to round or vice versa.
- Move Point: Interactively move a selected point on the screen.
- Align Point: Points can be aligned to one another in x or y or can be aligned to a selected vertical alignment line.

Lines

- Move: Interactively manipulate a spline curve segment by selecting a curve point and moving it.

Parts

- Move: Interactively translate a part on the screen.
- Scale: Scale an object in x and/or y.
- Align: Align two parts in relation to one another. Parts can be left, right, top, and bottom aligned.
- Rotate: Rotate an object by a given angle.

Mirror: Invert a part horizontally or vertically.
Join: Two selected parts can be joined to one another in x or y and recontoured to form a single part or part group.

Part group and letter manipulation functions are the same as those used to manipulate parts.

Automatic Letter Generation

To automatically generate a letterform, the following basic procedure is used. Within each letter function, the type of each component part is pre-specified. Instance copies of the letter's appropriate master parts are made by the part manager. These part instances are rotated and/or inverted to their proper orientation and sent to the letter construction procedure where they are positioned and joined. (See Figure 7k) If the correct master parts do not exist, they are created by the part manager. For example, to create an n, an xheight stem is needed. If the xheight stem does not exist in the library, the part manager copies the ascender stem master part and it is scaled to produce the xheight stem. This new part is sent to the master part library and copied, along with the arch and right short stem, to create the n. (See Figure 7l)

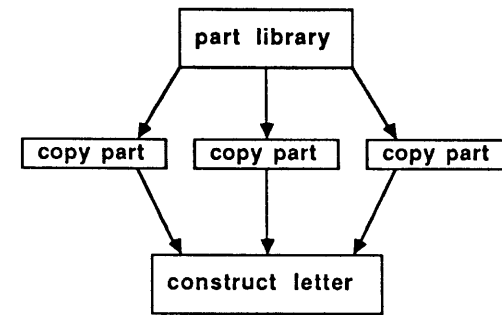


Figure 7k

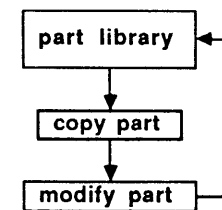


Figure 7l

Currently, if a part cannot be created given the existing set of source parts, the user is asked to make it. A sample n and u are reproduced in Figure 7m.

The part manager calls all of the procedures that are used to generate and modify parts and maintains all derivation and propagation data. This data includes information that specifies all source and destination letters and parts. To date, the procedures include operations to create the xheight stem and the arch of the m and rotation procedures to create the d and q.

Automatic Letter Construction

To build a letterform, part primitives are aligned and positioned in relation to one another and joined. Part and part group objects can be left, right, top, and bottom aligned, individually moved to a round, square, arch or apex y alignment value, or horizontally aligned in relation to a left or right side bearing. To join two parts, one object is positioned to the left, right, top, or bottom of another object and together they are recontoured to form a higher level structural unit. Parts are joined together until a complete letter is formed. The part configuration, or letter, is then positioned horizontally in relation to the left side bearing and vertically aligned. The right side bearing value is then

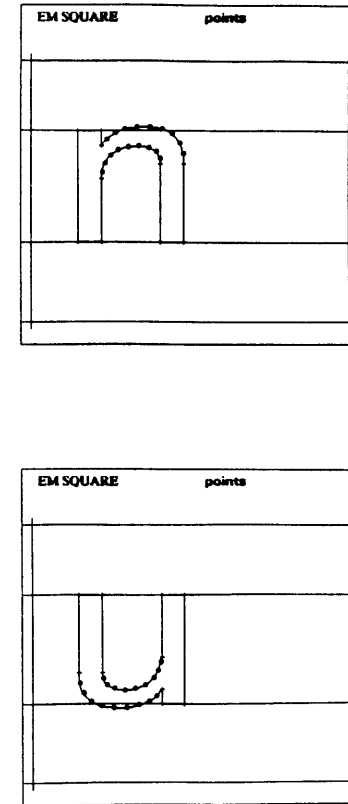


Figure 7m

assigned and the letter is displayed at scaled size in the View area of the screen.

For example, to create an n, its arch and left right stem are left aligned. The arch is then positioned on top of the right stem and these two parts are joined. This part group is then positioned to the right of the left stem and joined to it to form the n. Therefore, each letter schema contains a pre-defined set of procedures and expects to be sent the correct set of parts.

The recontouring procedures create a single object list of contour points by appending two part object lists. One procedure is used to join two objects in x, or a left and right object; the other procedure is used to join two objects in y, or a top and bottom object. These procedures are called ObjGlue_X and ObjGlue_Y and they are described in the Appendix. This new list is a display list that contains pointers to the point structures of each part object. Therefore, when the user edits the constructed letterform contour, part coordinates are automatically updated. This technique was used to facilitate change propagation to part contours.

Propagating Changes to Letters

As mentioned, when a given letterform is edited, changes to its part contours are automatically made. If a part shape has been changed, it becomes a new part instance and it is copied to the master part library. To propagate a change, the new master part is copied and sent to the construction procedure of each letter to be updated. The letters are rebuilt using the substituted part(s). (See Figure 7n) Changes in the curvature of the n arch are illustrated in Figure 7o. These changes are automatically updated to the u shown in Figure 7p. The part manager maintains records of all derivation and propagation paths, i.e. all source and destination letters and parts, and performs all substitution functions.

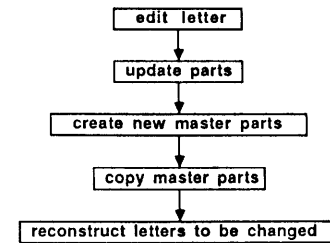


Figure 7n

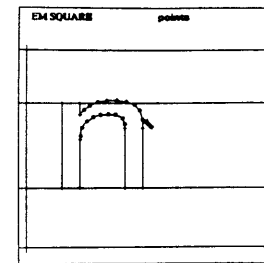


Figure 7o

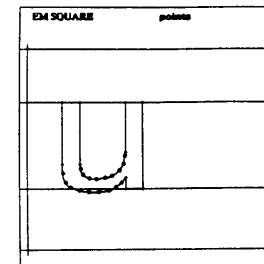


Figure 7p

CONCLUSION

The software presented in the preceding Chapter can be used to generate letterforms from a subset of character shapes. These character shapes are created by the designer and stored as part primitives that are copied and manipulated by the computer to construct other letterforms. With the use of procedures, parts are rotated or inverted, aligned, and positioned in relation to one another. Rules are used to constrain their horizontal and vertical alignment and rotation or inversion values. In addition, each letter can have its own value slots. In the current implementation of abcdefg, the part manager can duplicate part shapes and modify them. However, to date, these modifications are limited to altering the height of vertical stems and the width of arches. More complex shape manipulation procedures need to be tested.

While this schema is useful for constructing letters given a set of part shapes created by the designer or manipulated by the system, the actual generation of shape contours by the computer requires further investigation. At the lowest level, contour points must be plotted in a stylistically consistent fashion. At a higher level, design attributes such as the thickness of strokes and their alignment and height values need to be constrained. And, at a higher level

CONCLUSION

still, the attributes of shapes must be specified. In short, the system must know not only *how* to draw the shape but *what* to draw. [Montalvo 1985]

Interesting work is being done in this area by Montalvo. Montalvo is developing a declarative model of shape attribute representation. Her goal is to use this model to recognize, manipulate, and generate visual objects. In Montalvo's scheme, objects are represented as sets of low level visual properties that are related to one another to form higher level symbolic descriptions.

"The recognition and generation of visual objects from symbolic descriptions are two sides of the same coin." [Montalvo 1985] With the use of Montalvo's vocabulary, diagrammatic conversations can be established between the computer and the user. Whether or not this vocabulary will ultimately be applicable to generating complex shapes such as letterforms remains to be seen.

In the domain of analog to digital font conversion, the recognition/graphics problem is being addressed at Bitstream, Inc. in Cambridge, Massachusetts. As noted in Chapter Four, significant design features of each letter are represented by an underlying grid or pattern that corresponds to areas of the shape, called zones, that are to be constrained in relation to other letters. Zone

CONCLUSION

representation can be used to model varying contour curve shapes and to manipulate the design features within each letter independently of one another. Zone patterns are stored for each typeface. Although in its current implementation zone areas are specified by the user, work is proceeding towards automating the recognition of one dimensional features, or edges, and two dimensional shapes such as parts. If reversed and applied to font generation, zone patterns could be automatically created for each letter. Unknown shapes can be defined in relation to existing letter zone descriptions in another typeface or within the font itself.

Within the specific context of this thesis investigation, one solution to the lack of design information that can be applied to the derivation of letterforms is to expand the initial source subset of letters to include all or more of the basic part primitives with which letters can be created. Another possibility is to store pre-defined primitives and manipulate their shape in relation to the control characters created by the designer.

The addition of feature recognition software to the abcdefg package would be beneficial for automating the derivation process. At present, primitive procedures are used to reference contour points when modifications are

CONCLUSION

made. Individual contour edges or curve segments cannot be located easily.

Parts are defined by the user. Most important, however, is the fact that shape attribute information cannot be extrapolated from the control characters without the intervention of the designer. Widths, heights, and alignments are known, but design features such as the curve axis cannot be found. If the goal is to reduce the need for shape specification, feature recognition coupled with a generative grammar or procedures and a descriptive vocabulary could be useful.

The constraint representation software to be used in the abcdefg package is a highly simplified version of a constraint system. Boxes and wires can be created and used to constrain defined objects or values such as set widths, alignments, and stroke weights and to propagate value changes. In an ideal design environment, a network of two way constraints among all letter attributes and values could be implemented to both derive and propagate design information, thereby allowing the designer to begin with his own chosen set of control characters, to modify letters at any decision node in the tree, and to establish his own design relationships.

REFERENCE BIBLIOGRAPHY

Anti - Aliasing

- Bender, Walter "Anti-Aliasing for Broadcast Compatible Television", Society for Information Display 1984 Symposium Digest Display, June, 1984.
- Cannon, Don L. and Gerald Luecke Understanding Communications Systems, Texas Instruments, Dallas, 1980.
- Crow, Franklin "The Aliasing Problem in Computer - Generated Shaded Images", Communications of the ACM, Vol. 20 (11), November 1977, pp. 799 - 805.
- Crow, Franklin "A Comparison of Antialiasing Techniques", IEEE Computer Graphics Applications, Vol. 1 (1), January, 1981, pp. 40 - 48.
- Crow, Franklin "Computational Issues in Rendering Anti - Aliased Detail", IEEE Computer Graphics and Applications, January, 1981, pp. 238 - 244.
- Crow, Franklin "The Use of Grayscale for Improved Raster Display of Vectors and Characters", Computer Graphics, Vol. 12, No. 2, July, 1978, pp. 1 - 5.
- Fishkin, Kenneth, P. and Brian Barsky "A Family of New Algorithms for Soft Filling", Computer Graphics, Vol. 18, No. 13, July, 1978, pp. 235 - 244.
- Gupta, Satish and Robert F. Sproull "Filtering Edges for Gray - Scale Displays", Computer Graphics, Vol. 15, No. 3, August, 1981, pp. 7 - 15.

REFERENCE BIBLIOGRAPHY

- Kajiya, J. and M. Ullner "Filtering High Quality Text for Display on Raster Scan Devices", Computer Graphics, Vol. 15, No. 3, August, 1981, pp. 7 - 15.
- Kobayashi, Sandra C. Optimization Algorithms for Grayscale Fonts, B. S. Thesis, Massachusetts Institute of Technology, 1980.
- Leler, William J. "Human Vision, Anti - Aliasing and the Cheap 4000 Line Display", Computer Graphics, Vol. 14, No. 3, July, 1980, pp. 308 - 313.
- Schmandt, Christopher "Soft Typography", Architecture Machine Group, Massachusetts Institute of Technology, Cambridge, MA, 1980.
- Warnock, John and Douglas K. Wyatt "A Device Independent Graphics Imaging Model for Use with Raster Devices", Computer Graphics, Vol. 14, No. 3, July, 1982, pp. 313 - 319.
- Warnock, John E. "The Display of Characters Using Gray Level Sample Arrays", Computer Graphics, Vol. 14, No. 3, July, 1980, pp. 302 - 307.
- Whitted, Turner "The Causes of Aliasing in Computer Generated Images", Siggraph 1982 Advanced Image Synthesis Seminar, pp. 1 - 7.

Character Recognition

- Cox III, C.H.,B.A. Blesser, "The Graphical Context of Printed Characters", 105

REFERENCE BIBLIOGRAPHY

- and M. Eden Visible Language XII 4, Autumn, 1978, pp. 430 - 442.
- Skillman, R., C. Cox, T. "A Bibliography in Character Recognition: Tech-
Kuklinski, J. Ventura, niques for Describing Characters", Visible Lang-
M. Eden, and B. Blesser uage VIII 2, Spring, 1984, pp. 151 - 166.
- Skillman, R., C. Cox, T. "Character Recognition Based on Phenomenolog-
Kuklinski, J. Ventura, ical Attributes", Visible Language VII 3, Sum-
and M. Eden mer, 1973, pp. 209 - 223.
- Troxel, Donald E. "Automated Reading of the Printed Page",
Visible Language II, Spring, 1971, pp. 125 -
144.
- Watt, W.C. "What is the Proper Characterization of the
Alphabet", Visible Language IX 4, Autumn 1975,
pp. 293 - 327.

Constraint Systems

- Borning, Alan Thinglab: A Constraint-Oriented Simulation
Laboratory, PhD. Thesis, Stanford University,
1979.
- Gross, Mark D. Design As Exploring Constraints, PhD. Thesis,
Massachusetts Institute of Technology, 1985.

REFERENCE BIBLIOGRAPHY

Curve Representation

- Kochanek, Doris and
Richard H. Bartels "Interpolating Splines with Local Tension, Continuity, and Bias Control", Computer Graphics, Vol. 18, No. 3, July, 1984, pp. 33 - 41.
- Pavilidis, Theo "Curve Fitting with Conic Splines", ACM Transactions on Graphics, Vol. 2, No. 1, January, 1983, pp. 1 - 31.
- Plass, Michael and
Maureen Stone "Curve - Fitting with Piecewise Parametric Curves", ACM, Computer Graphics, Vol. 17, No. 3, July, 1983.
- Pratt, Vaughan "Techniques for Conic Splines", Siggraph '85, Vol. 19, No. 3, 1985, pp. 151 - 159.
- Prosser, Colin J. and
Allstair C. Kilgour "An Integer Method for the Graphical Output of Conic Sections", ACM Transactions on Graphics, Vol. 2, No. 3, July, 1983, pp. 182 - 191.

Digital Font Generation Systems

- Bigelow, Charles "Type Design Principles and Automated Design Aids", Source Unknown, pp. 115 - 137.
- Carter, K. A. "The Rainbow Workstation in Brief", Computer Laboratory, University of Cambridge, May, 1984.
- Carter, K. A. "Imp - a System for Computer - Aided Typeface 107

REFERENCE BIBLIOGRAPHY

- Design", Computer Laboratory, University of Cambridge, May 1984.
- Carter, K. A. and E. Wiseman "Designing in Fives", Computer Laboratory, N. University of Cambridge, May, 1984.
- Flowers, Jim "Digital Type Manufacture; An Interactive Approach", Computer, May, 1984, pp. 40 - 48.
- Hobby, John and Gu Guoan "A Chinese Metafont", Department of Computer Science, Stanford University, Stanford, California, July, 1983.
- Hofstadter, Douglas R. "Metafont, Metamathematics, and Metaphysics", Visible Language XVI 4, 1982, pp. 309 - 338.
- Hofstadter, Douglas R. "A Reply from Douglas Hofstadter", Visible Language XVII 4, 1983, pp. 413 - 416.
- Huggins, Cleo and Mike Sheridan "Exercise Overview: Ikarus and Signus", Fifth ATypI Working Seminar paper, Summer 1983.
- Knuth, Donald E. "The Concept of a Metafont", Visible Language XVI 1, Winter 1982, pp. 3 - 27.
- Knuth, Donald "Metafont Workbook", Fifth ATypI Working Seminar paper, Stanford University, Stanford, California, Summer, 1983.
- Knuth, Donald The METAFONTbook, Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.
- Knuth, Donald TEX and Metafont, American Mathematical Society and Digital Press, 1979.

REFERENCE BIBLIOGRAPHY

- Ruggles, Lynn "Letterform Design Systems", Technical Report STAN - CS - 83 - 971, Computer Science Department, Stanford University, 1983.
- Ruggles, Lynn Paragon, An Interactive, Extensible Typeface Design Environment, PhD. Thesis, University of Massachusetts, (in preparation).
- Sanders, John "Intelligent Fonts for Asian Scripts", National Print Quality Seminar Proceedings, 1982, pp. 45 - 56.
- Siegel, David The Euler Project at Stanford, Department of Computer Science, Stanford, California, 1985.
- Southall, Richard "Designing New Typefaces With Metafont", Technical Report STAN - CS - 85 - 1074, Computer Science Department, Stanford University, September, 1985.
- Trombley, John E. "Font Creation: A Case Study in Low Cost CAD Applications", National Print Quality Seminar Proceedings, 1982, pp. 129 - 135.
- Unternehmensberatung, Rubow, and Weber "Ikarus - System: Computer Controlled Font Production for Photocomp, Crt, and Lasercomp", Hamburg, Germany, date unknown.

Digital Type

- Bigelow, Charles "Book Review: Kindersley, Optical Letter Spacing for New Printing Systems", Visible Lang-

REFERENCE BIBLIOGRAPHY

- uage XI 3, Summer, 1977, pp. 325 - 329.
- Bigelow, Charles and Donald Day "Digital Typography", Scientific American, Volume 249, Number 2, August, 1983, pp. 106 - 119.
- Bigelow, Charles "The Principles of Digital Type: Quality Type Low, Medium, and High Resolution Printers", Seybold Report, Vol. 11, No. 11, February 8, 1982, pp. 11.3 - 11.23.
- Bigelow, Charles "Technology and the Aesthetics of Type", Seybold Report, Vol. 10, No. 24, August 24, 1981, pp. 24.3 - 24.16.
- Buyers, Stephen "Digital Fonts, RIP's, Video and Outline", Lasers in Graphics/Electronic Publishing in the 80's, 1983, pp. 320 - 329.
- Buyers, Stephen "Developing a Type Library for the Eighties", National Print Quality Seminar Proceedings, 1982, pp. 37-43.
- Bond, Liz "Digital Font Development for Typographic Print Quality", National Print Quality Seminar Proceedings, 1982, pp. 25 - 35.
- Casey, R.G., T.D. Freidman, and K.Y. Wong "Automatic Scaling of Digital Print Fonts", IBMJ Research Development, Vol. 26, No. 6, November, 1982, pp. 657 - 666.
- Carter, Harry "Optical Scale in Type Founding", Printing Historical Society, Bulletin 13, September, 1984, pp. 144 - 148.

REFERENCE BIBLIOGRAPHY

- Johnson, Bridget A Model for Automatic Optical Scaling of Type Designs for Conventional and Digital Technology, M. S. Thesis, School of Printing, Rochester Institute of Technology, (in preparation).
- Latham, John "Developing Digital Type Fonts at Monotype", National Print Quality Seminar Proceedings, 1982.
- Naiman, Avi C. High Quality Text for Raster Displays, M. S. Thesis, Department of Computer Science, University of Toronto, January, 1985.
- Parker, Mike "Digitizing Fonts for Low and Medium Resolution Output Machines", National Print Quality Seminar Proceedings, 1982.
- Seybold, John "Digitized Type: The New Letterforms", Seybold Report, Vol. 8, No. 24, August 27, 1979, pp. 24.3 - 24.17.
- Stewart, Doug "The Delicate Art of Digital Type", Popular Computing, January, 1985, pp. 99 - 102.

Knowledge-Based Systems

- Allen, Elizabeth "Yaps: A Production Rule System Meets Objects", AAAI - 83, 1983, pp. 5 - 7.
- Badler, Norman I. and
Timothy W. Finn "Computer Graphics and Expert Systems", IEEE CG and A, 1985, pp. 25 - 28.

REFERENCE BIBLIOGRAPHY

- Barr, Airon and Feigenbaum (eds.) The Handbook of Artificial Intelligence, Volumes I, II, and III, HeurisTech Press, Stanford, California, 1981.
- Duda, R.O., P.E. Hart, N.J. Nilsson, and G.L. Sutherland "Semantic Network Representations in Rule-Based Interface Systems", Pattern - Directed Interface Systems, (D.A. Waterman and F. Hayes-Roth, eds.), Academic Press, Inc., 1978, pp. 203 - 221.
- Feiner, Steven "APEX: An Experiment in the Automated Creation of Pictorial Explanations", IEEE CG and A, 1985, pp. 29 - 37.
- Flemming, Ulrich "A Generative Expert System for the Design of Building Layouts", Carnegie - Mellon University, Pittsburgh, Pennsylvania, September, 1985.
- Harmon, Paul and Expert Systems, John Wiley and Sons, Inc., New York, 1985.
- Hayes - Roth, Frederick, Donald Waterman, and Douglas B. Lenat Building Expert Systems, Addison - Wesley Publishing Company, Inc., 1983.
- Hofstadter, Douglas Godel, Escher, Bach: An Eternal Golden Braid, Vintage Books, New York, 1979.
- Hofstadter, Douglas R. Metamagical Themas: Questing for the Essence of Mind and Pattern, Basic Books, Inc., Publishers, New York, 1985.
- Hull, Jonathan J. "Word Shape Analysis in a Knowledge - Based System for Reading Text", IEEE, 1985, pp.

REFERENCE BIBLIOGRAPHY

114 - 119.

Montalvo, Fanya S. Diagram Understanding: The Intersection of
Computer Vision and Graphics, A.I. Memo 873,
Massachusetts Institute of Technology, 1985.

Nilsson, Nils J. Principles of Artificial Intelligence, Tioga
Publishing Company, Palo Alto, California,
1980.

Legibility

Foster, Jeremy J. "Locating Legibility Research: A Guide for the
Graphic Designer", Visible Language X, 1976,
pp. 257 - 268.

Romano, Frank "Perception and Legibility Issues", National
Print Quality Seminar Proceedings, 1982.

Spencer, Harold The Visible Word, Royal College of Art, London,
1968.

Printing History

Seybold, Johnathan W. The World of Digital Typesetting, Seybold
Publications, Inc., Media, Pennsylvania, 1984.

Seybold, Johnathan W. Modern Photocomposition Systems, Seybold
Publications, Inc., Media, Pennsylvania, 1979.

REFERENCE BIBLIOGRAPHY

Steinberg, S.H. Five Hundred Years of Printing, Penguin Books, Ltd., Middlesex, England, 1974.

Updike, D.B. History of Printing Types, Volumes I and II, Dover Publications, Inc., New York, 1980.

Shape Grammars

Knight, T.W. "The Generation of Hepplewhite - Style Chair - Back Designs", Environment and Planning B, Vol. 7, 1980, pp. 227 - 238.

Knight, T.W. "The Forty-One Steps", Environment and Planning B, Vol. 8, 1981, pp. 97 - 114.

Koning, H. and
J. Eizenberg "The Language of the Prairie: Frank Lloyd Wright's Prairie Houses", Environment and Planning B, Vol. 8, 1981, pp. 295 - 323.

Stiny, G. and
W.J. Mitchell "The Grammar of Paradise: On the Generation of Mughul Gardens", Environment and Planning B, Vol. 7, 1980, pp. 209 - 226.

Typeface Classification

Gaskell, Philip "A Nomenclature for the Letterforms of Roman Type", Visible Language XI, Winter 1976, pp. 41 - 51.

REFERENCE BIBLIOGRAPHY

- Jaspert, W.P., W.T. Berry,
and A.F. Johnson The Encyclopedia of Typefaces, Blandford Press,
Dorset, 1983.
- Lawson, Alexander Printing Types, Beacon Press, Boston, 1971.
- Sutton, James and
Alan Bartram An Atlas of Typeforms, Lund Humphries,
London, 1968.
- Weinberger, Norman S. Encyclopedia of Comparative Letterforms, Art
Directions Book Company, New York, 1965.
- Perfect, Christopher and
Gordon Rookledge Rookledge's International International
Typefinder, Sarema Press, 1983.
- V and M Typographical, Inc. The Type Specimen Book, Van Nostrand Reinhold
Company, 1974.

Typeface Design and Lettering

- Anderson, Donald M. The Art of Written Forms: The Theory and
Practice of Calligraphy, Holt, Rinehart, and
Winston, Inc., New York, 1969.
- Benson, John Howard and
Arthur Graham Carey The Elements Of Lettering, McGraw-Hill Book
Company, Inc., New York, 1950.
- Brown, Frank Chouteau Letters and Lettering, Bates and Guild Company,
Boston, 1902.
- Camp, Ann Pen Lettering, Taplinger Publishing Co., Inc.,
New York, 1980.

REFERENCE BIBLIOGRAPHY

- Carter, Matthew "Bell Centennial", Type and Technology Monograph #1, The Center for Design and Typography, The Cooper Union, New York, 1982.
- Deighton, Harold The Art of Lettering, B. T. Batsford Ltd., London, 1947.
- Degering, Hermann Lettering, Pentalic Corporation, New York, 1965.
- Dwiggins, W. A. Mss. by WAD, The Typophiles, New York, 1947.
- Frutiger, Adrian Type Sign Symbol, ABC Edition, Zurich, 1980.
- Gardner, William Alphabet at Work, St. Martin Press, New York, 1982.
- Gill, Eric An Essay on Typography, J.M. Dent and Sons, Ltd., 1931.
- Goudy, Fredrick W. Typologia. University of California Press, Berkley, California, 1977.
- Gray, Nicolette "Slab-Serif Type Design in England: 1815 - 1845, Journal of the Printing Historical Society, No. 15, 1980/81, pp. 1 - 28.
- Hess, Stanley The Modification of Letterforms, Art Direction Book Company, New York, 1981.
- Hutchinson, James Letters, Van Nostrand Reinhold Company, New York, 1983.
- Level, Jeffery "What Type Isn't - Letterforms in the Present, Past and Future Tenses", Ligature, Vol. 3, No. 116

REFERENCE BIBLIOGRAPHY

- 1, January, 1985, pp. 4 - 6.
- Lindergren, Erik An ABC Book, Grafisk Studio, Askin, Sweden, 1976.
- Morisson, Stanley Letterforms, The Typophiles, New York, 1968.
- Mergenthaler Linotype
Company "Statement of Mergenthaler Linotype Company
in Support of the Registerability of the Claim of
Copyright In Original Typeface Design", Mergen-
thaler Linotype Company, 1971.
- Romano, Frank The TypEncyclopedia, R. R. Bowker, Company,
New York, 1984.
- Wanner, Gustaf Adolf "The Haas Typefoundry Looks Back on 400
Years", TMY, 1980.
- Zapf, Herman About Alphabets. MIT Press, Cambridge, MA,
1978.
- Zapf, Herman Manuale Typographicum, The M.I.T. Press, Cam-
bridge, Massachusetts, 1970.

Type Founding Technology

- Bullen, Henry Lewis "Linn Boyd Benton - The Man and His Work", The
Inland Printer, Oct. 1922, pp. 60 - 64.
- Carter, Harry G. "Letter Design and Typecutting", Percy Smith
Memorial Lecture, Journal of the Royal Society
of Arts, October 1, 1954.

REFERENCE BIBLIOGRAPHY

- Koch, Paul "The Making of Printing Types", The Dolphin Number One, Limited Editions Club, New York, 1933, pp. 24 - 57.
- Warde, Beatrice "Cutting Types for The Machines: A Layman's Account", The Dolphin Number Two, New York: Limited Editions Club, 1935.
- Typography**
- Arnheim, Rudolf "Spatial Aspects of Graphical Expression", Visible Language XII 2, Spring, 1978.
- Dair, Carl Design with Type, University of Toronto Press, Toronto, 1982.
- Damen, Herman "Excerpt: Towards a Three-Dimensional Poetry", Visible Language XI, Winter, 1976, pp. 41 - 51.
- Frank, Harry "Typographs", Visible Language VII 3, Summer 1973, pp. 109 - 208.
- Gerstner, Karl Designing Programmes, Arthur Niggli, Ltd., Jeuten, Switzerland, 1968.
- Gerstner, Karl Compendium for Literates, R. Weber AC, Heiden, Switzerland, 1974.
- Glenn, Bernice T. "Alphabet and Text in Presentation Graphics", Computer Graphics World, January, 1984, pp. 10 - 22.

REFERENCE BIBLIOGRAPHY

- Gray, Nicolete Lettering as Drawing, Taplinger Publishing Company, New York, 1971.
- Hatherly, Ann "The Reinvention of Reading", Visible Language XI 3, 1977, pp. 307 - 319.
- Hellerstein, Neil S. "Paul Claudel and Guillaume Apollinaire as Visual Poets: Ideogrammes Occidentox and Calligrammes", Visible Language XI 3, 1979, pp. 245 - 270.
- Huey, Edmond Burke The Psychology and Pedagogy of Reading, The M.I.T. Press, Cambridge, Massachusetts, 1979.
- Kim, Scott Inversions: A Catalog of Calligraphic Cartwheels, McGraw Hill, Peterborough, New Hampshire, 1981.
- Marcus, Aaron "An Introduction to the Visual Syntax of Concrete Poetry", Visible Language VIII 4, pp. 333 - 360.
- McLean, Ruari The Thames and Hudson Manual of Typography, Thames and Hudson, Ltd., London, 1980.
- Rogers, Bruce Paragraphs on Printing, Dover Publications, Inc., New York, 1979.
- Shafran, Joan The Poetry Generator, M. S. Thesis, Department of Architecture, Massachusetts Institute of Technology, 1980.
- Shoboda, John "The Uses of Space in Music Notation ", Visible Language XV 1, pp. 86 - 110.

REFERENCE BIBLIOGRAPHY

- Spencer, Herbert Pioneers of Modern Typography, The MIT Press, Cambridge, Massachusetts, 1983.
- Tschichold, Jan "Assymetric Typography", Reinhold Publishing Co., New York, 1935.
- Twyman, Michael "Typography Without Words", Visible Language XVI, pp. 5 - 12.

Vision

- Arnheim, Rudolf Art and Visual Perception, University of California, Press Berkeley, California, 1974.
- Cornsweet, Tom N. Visual Perception, Academic Press, New York, 1970.

APPENDIX

Function Description Glossary

cellp	Cell pointer.
cpmp	Child pmdata pointer.
cursx	Current cursor x coordinate, in the vis coordinate system.
cursy	Current cursor y coordinate, in the vis coordinate system.
EM_SQ	Name of the EM_SQ workspace area.
INV_EM_SQ	In contrast to the EM_SQ which is a workspace, the INV_EM_SQ is a pmd. It lives in the invisible frame buffer.
invis	Invisible frame buffer.
level	Refers to the level of a letter that is being operated on or displayed.
objarrayp	Object array pointer.
objectp	Object pointer.
pmd	YODA pixel map data structure. Pmd's are rectangular pixel map display areas in both the visible and invisible frame buffers.
pmd_name	A pointer to a YODA pmd.
pmdatap	Pmdata pointer.
ppmp	Parent pmdata pointer.
rcursx	The cursx coordinate recalculated relative to a workspace area.
rcursy	The cursy coordinate recalculated relative to a workspace area.
ringp	Ring pointer.
sbp	Sd_bear pointer.
SCALED	Name of the SCALED workspace area.
seen_arrayp	Object array pointer. This is an array of objects visible in the EM_SQ workspace area.
TEXT	Name of the TEXT workspace area.
VIEW	Name of the VIEW workspace area.
valp	V_align pointer.

APPENDIX

vis	Visible frame buffer.
wkspp	Workspace pointer.
workmode	Refers to the working mode: create, generate, propagate, proof.
workspace	A working area of the screen. (See section on workspaces under the Screen Functions heading.)

Main Program

The main program is used for initialization of the display and video lookup table, YODA fonts, the INV_EM_SQ, and control character part structures. The Driver() program is then called. When control is returned from the Driver(), main calls a cleanup function to close all YODA fonts.

Driver

int	Driver() arguments: none	ndriver4.c
-----	------------------------------------	------------

The driver program is the main control loop. It is used to determine and maintain the current cursor location and state, the current workspace, the current object, the current workmode, the current tool, and the current working levels in the VIEW and EM_SQ workspaces.

This function first tests to see if a mouse button has been pressed. If so, it determines the current workspace. If the middle button is pressed, select an object or tool; else if the first or third button is pressed, change levels in the VIEW workspace or in the EM_SQ workspace. The first button

APPENDIX

switches down a level; the third button switches up a level.

Level Down Select Level Up

Selection Functions

int	WkspSelect() arguments: cursx, cursy	ndriver4.c
	Returns the current workspace.	
int	CpmSelect() arguments: current_workspace, rcursx, rcursy	ndriver4.c
	Returns the current cpm.	
int	SelectInit() arguments: cursx, cursy, current_workspace, current_workmode	ndriver4.c
	Recalculates the cursor coordinates relative to the current_workspace. Calls object and tool selection functions, and calls ToolExecute().	
int	LevelInit() arguments: current_workspace, mouse_button, current_level, cursx, cursy	ndriver4.c
	Determines if the level has been changed in the VIEW or EM_SQ workspace. Initializes the new level and returns the current	

APPENDIX

level.

int	LevelDisp() arguments: current_workspace, levelup_or_leveldown	ndriver4.c
	Displays the new level label. If the level change is in the EM_SQ workspace, displays the current level of the object, i.e. its parts, lines, or points. If the level change has occurred in the VIEW workspace, calls the library display function.	
int	ToolSelect() arguments: rcursx, rcursy	ndriver4.c
	Tool selection function. Initializes the current tool. Calls ToolHlt() to highlight the selected tool.	
int	ToolHlt() arguments: current_tool	ndriver4.c
	Highlights the current_tool.	
int	ToolExecute() arguments: current_tool, current_workmode, current_level	ndriver4.c
	Given the current_workmode and current_level, calls the appropriate object editing or creating function.	

APPENDIX

Screen Objects and Functions

Workspaces

The screen is divided into six working areas called workspaces. Each workspace is stored as an individual data structure that contains pointers to structures called `_pm_data` structures. A `_pm_data` structure contains the screen objects within each workspace area. There are two types of `_pm_data` pointer names stored with each workspace. Parent `_pm_data` pointers, called `ppm`, and an array of children `_pm_data` pointers, called `cpm`. The parent `_pm_data` structure contains all the information associated with the workspace, and the children `_pm_data` structures contain all the information associated with any pixel maps located within a workspace. Each `_pm_data` structure contains pointers to 2 Yoda pixel maps, a parent `pmd_name` and the `pmd_name` associated with the `_pm_data` structure itself. For example, the parent `pmd_name` for the `EM_SQ` workspace is "&vis". Other `_pm_data` structure information includes pixel map dimensions, display color, highlight display color, text label strings, and the current x and y display starting point for text strings.

Workspace pointers are stored as an array. Array indices are named. These names are used throughout these appendices to refer to workspaces.

0	EM_SQ
1	VIEW
2	TEXT
3	SCALED
4	CONTROLS
5	TOOLS

Each workspace area is labelled on the screen. Label names are: "EM SQUARE", "VIEW", "TEXT", "SCALED", and "TOOLS".

APPENDIX

EM_SQ workspace:

The EM_SQ workspace is the area in which full-size letters and letter primitive objects are created and edited. It is a 360 x 360 pixel area. Four vertical alignment lines are displayed in the EM_SQ: the descender line, baseline, xheight line, and ascender line. Two side bearing lines are also displayed: a right side bearing and a left side bearing.

There is one cpm pixel map in the EM_SQ workspace. This is situated at the upper right corner of the workspace area. It is used to change letter object levels. There are five labels associated with this cpm. They are: "points", "lines", "parts", "partgroups", and "letterforms".

VIEW workspace:

The VIEW workspace is used to display part and letter libraries. Parts and letters are scaled to one quarter of their original size for display. Part and letter objects are displayed in horizontal rows.

This workspace contains one cpm area. This is situated at the upper right corner of the VIEW workspace. It is used to display libraries and to switch between part and letter libraries. There are two labels associated with this area. They are "parts" and "letterforms".

A copy of the EM_SQ pmd is stored in the invisible frame buffer and used for BitBlit purposes. This is a pmd, not a workspace, called INV_EM_SQ.

TEXT workspace:

The TEXT workspace is used to display scaled letters in text strings. The scaled letters are one quarter of their full size.

There are no cpm's in this workspace.

APPENDIX

SCALED workspace:

The SCALED workspace is used to display a scaled version of the current letter or part object being displayed, created, or edited in the EM_SQ workspace. This letter is displayed at one quarter of its full size.

There are no cpm's in this workspace.

CONTROLS workspace:

This workspace area is not used.

TOOLS workspace:

The TOOL workspace is used for displaying available tools and functions for creating and modifying letter objects and primitives. These tools are displayed as menu buttons in a long vertical column.

Thirteen tool buttons can be displayed., i.e. there are 13 cpm's associated with this area.

Workspace Structure

```
typedef      struct _wk_space      *wk_space
             int                    name;
             pmdata                  ppm;          /*parent pmdata*/
             pmdata                  cpm[13];      /*children pmdata*/
             int                      cpmnum;
```

```
PixelMapData Structure          /* Not Yoda Pixel Map Structure */
typedef      struct _pm_data      *pmdata;
```

APPENDIX

```
pmd      *name, p_name;
int      origx, origy, ux, uy, extx, exty;
int      height, width;
int      color;
int      highlightflag;
int      highlight_color;
char     *label[6];
int      labelnum;
int      curlabnum;
dblcoord curpt_x, curpt_y; /* for text */
```

Workspace Functions

int	WSAllocAllPtrs() arguments: none	screen02.c
	Allocates and initializes all wksp's, ppmp's, cpmp's, valp's and sbp's.	
int	PmInitAll() arguments: none	screen02.c
	Specifies all data for all workspace _pm_data structures.	
int	PmInit() arguments: pmdatap, pmd_name, origin_x, origin_y, extent_x, extent_y, color, if_highlighted, highlight_color, if_active	screen02.c
	Initializes all data in a pmdata structure.	

APPENDIX

int	LabInitAll() arguments: none	screen02.c
	Initializes all labels and label current points.	
int	LabDisplay() arguments: pmdatap, curlabelnum, font_control_block, color, pmd_name	screen02.c
	Displays a text label.	
int	DisplayAll() arguments: none	screen02.c
	Displays all screen pmd's, vertical alignment lines, side bearing lines, and text labels.	
int	PmdCreate() arguments: pmdatap	screen02.c
	Creates a YODA pmd.	
int	PmdDisplay() arguments: pmdatap	screen02.c
	Displays a YODA pmd.	
int	ClearEm() arguments: none	

APPENDIX

Clears the EM_SQ screen area and redraws alignment and side bearing lines.

int **InvEmRestore()** mmvsppt08..c
arguments: none

Clears the INV_EM_SQ pmd and redraws all alignment and side bearing lines.

Vertical Alignment Line Display Structure

```
typedef        struct _v_align        *v_align
int            origx, origy, endx, endy, extx,
               exty;
int            width;
int            yoffset;
int            color;
pmd            *pmd_name;
int            if_highlight;
int            highlight_color;
```

Vertical Alignment Line Functions

int **VallnitAll()** screen02.c
arguments: none

Specifies all data for all `_v_align` structures.

int **Vallnit()** screen02.c
arguments: pmdatap, valp, origin_x, origin_y, end_x, end_y,
 color, if_highlighted, highlight_color, if_active

APPENDIX

Initializes all data in a `_v_align` structure.

int	ValDisplay() arguments: <code>valp, pmd_name</code>	screen02.c
-----	---	------------

Displays a vertical alignment line.

int	ValpSelect() arguments: <code>rcurs_x, rcurs_y</code>	valpedit.c
-----	---	------------

Selection function for vertical alignment line. Cursor coordinates are recalculated relative to the EM_SQ pmd before entering this function.

int	ValpEdit() arguments: <code>vertical_align_line</code>	valpedit.c
-----	--	------------

Interactive loop for moving an alignment line vertically.

Sidebearing Display Structure

typedef	struct _sd_bear	<code>*sd_bear</code>
	int	<code>origx, origy, endx, endy, extx,</code>
		<code>exty;</code>
	int	<code>width;</code>
	int	<code>xoffset;</code>
	int	<code>color;</code>
	pmd	<code>*pmd_name;</code>
	int	<code>if_highlight;</code>
	int	<code>highlight_color;</code>

APPENDIX

Sidebearing Functions

int	SBInitAll() arguments: none	screen02.c
	Specifies all data for all sd_bear data structures.	
int	SBInit() arguments: pmdatap, sd_bearp, origin_x, origin_y, end_x, end_y, color, if_highlighted, highlight_color, if_active	screen02.c
	Initializes all data in a sd_bear structure.	
int	SBDisplay() arguments: sd_bearp, pmd_name	screen02.c
	Displays a side bearing line.	
int	SBSelect() arguments: rcurs_x, rcurs_y	valpedit.c
	Selection function for side bearing lines. Cursor coordinates are recalculated relative to the EM_SQ pmd before being sent in.	
int	SBEdit() arguments: side_bearing_line	valpedit.c
	Interactive loop for moving a side bearing line horizontally.	

APPENDIX

Pop Up Menu

A pop up menu is used to change working modes. This menu is displayed at the current cursor location.

Pop Up Menu Functions

int	MenuDisplay() arguments: none	menu.c
	Interactive loop for displaying the pop up menu and for menu item selection.	
int	ItemSelect() arguments: cur_tabx, cur_taby, morigx, morigy, oy, y_distance, left_margin, right_margin, numitems, extent_x, height, menu_extent_x	menu.c
	Function for selecting a menu item. There are four menu items to correspond to the four working modes.	
int	ItemHit() arguments: item	menu.c
	Highlights a menu item.	

APPENDIX

Letter and Part Structures and Functions

Letters

Letterform Library Structure

```
struct _letterlibrary
struct _letterclasses    h_class, n_class, m_class, u_class,
                           i_class, l_class, p_class, d_class,
                           b_class, q_class;
```

Letter Class Structure

```
struct _letterclasses
struct _letters         *letterlist;
int                       num_letters;
```

Letterform Structure

```
struct _letters
struct _parts           *partlist;
struct _objects        *object;
int                       height, width;
int                       top_align, bottom_align;
int                       left_sidebearing, right_sidebearing;
```

APPENDIX

Parts

PartLibrary Structure

struct _partlib	partlib
struct _partclass	stem, bowl, arch;

Part Class Structure

typedef	struct _partclass	*partclasses
	partsubcl	subclass[10];
	int	numsubs;

Part Subclass Structure

typedef	struct _partsubcl	*partsubcl
	int	name;
	_parts	masterpart;
	int	numparts;

Part Structure

typedef	struct _part	*parts
	objects	object;
	int	type;
	struct _part	*next, *previous;
	struct _part	*master;
	int	height, width;
	int	orientation, rotation;
	int	topalign, botalign;
	int	originx, originy;
	int	x[100], y[100], t[100];

APPENDIX

Part Functions

parts	PartInit() arguments: none	partdef.c
	Allocates space for a part structure and initializes all pointers to NULL and all integers to 0 or -1.	
int	MasterPartDef() arguments: none	partdef.c
	Initializes all part structures. Reads part data from part files. Assigns part classes, subclasses, and master part objects and part names. Also initializes square and round alignment values.	
parts	PartCopy() arguments: from_partp	partdef.c
	Copies from_part. Returns part copy.	
parts	MasterPartCopy() arguments: partp	partdef.c
	Copies a master part and returns copy.	
objects	ShowPart() arguments: partp, iflines, ifpoints	showp.c
	Displays a part object on the screen in the EM_SQ workspace area. If iflines is TRUE display the outline of the part; if ifpoints is TRUE, display the points.	

APPENDIX

int	PartHeight_Get() arguments: partp	rparts3.c
	Calculates the height of a part object.	
int	PartHeight_Edit() arguments: partp, newheight	rparts3.c
	Searches part object point y coordinates to locate those points that need to be reset to change height of a part. Changes y values of these points to the newheight.	
int	PartAlign_X() arguments: partp, align_value	rparts3.c
	Aligns a part object to a given x alignment value.	
int	PartAlign_Y() arguments: partp, align_value	rparts3.c
	Aligns a part object to a give y alignment value.	
int	DrawOffPart() arguments: partp, offset_x, offset_y	rparts3.c
	Displays a part object in the EM_SQ workspace area at a given offset x and offset y coordinate.	
int	ResetEmPartCoords() arguments: partp	rparts3.c

APPENDIX

Recalculates the offset, extent, and relative coordinates of a part object.

int **RelPartCrdsCalc()** rparts3.c
arguments: partp

Calculates the relative coords of a part object.

Objects

Objarray Structure

```
typedef        struct _objarrays        *objarrays  
              objects                    object[40];  
              int                        number;
```

Objarray Functions

objarrays **ObjArrInit()** crobj03.c
arguments: none

objarray Allocates space for an objarray structure and initializes the
number to zero.

int **ObjArrayPut()** crobj03.c
arguments: objarrayp, objectp

Inserts the object at the end of the object array. Tests to see if the object already exists in the array. If TRUE, the object is

APPENDIX

not inserted.

int **ObjArrDraw()** mvobj01.c
 arguments: objarrp, pmd_name, color

 Draws the objects in an objarray in a given pmd and in a given color.

Object Structure

```
typedef    struct _objects           *object
          char                         *name;
          rings                        ring[MAXRINGS];
          struct _objects             *up[MAXUP], *down[MAXDOWN];
          int                         extentx, extenty;
          int                         offsetx, offsety;
          int                         numdown, numup;
          int                         numrings;
```

Object Functions

objects **ObjInit()** crobj03.c
 arguments: none

 Allocates space for an object structure and initializes all pointers in structure to NULL and all integers to zero.

int **WriteToFile()** store2.c
 arguments: objecpt

 Writes an object list of coordinate values to files.

APPENDIX

objects	ReadObj() arguments: none	store2.c	
	Reads object coordinate values from file, and creates an object.		
objects	ObjCreate() arguments: none	crobj03.c	
	Interactive loop for creating a two way list of cubic spline curve control points. Object curve control points are interactively input by the user. Straight or curved point markers are displayed. When the cursor loop is exited, a cubic spline curve outline is drawn through the points. The object offset and extent is calculated, and the point coordinates are reset relative to the offset. The x, y offset is determined relative to the origin point of the EM_SQ pixel map.		
int	ObjSave() arguments: objectp	demo1.c	
	Saves changes to an edited object.		
objects	ObjSelect() arguments: seen_arrayp, cursx, cursy	select05.c	
	Tests to see which object in the seen_array the cursor is inside and closest to the center of.		
int	ObjDraw() arguments: objectp	demo1.c	
	Display an object on the screen.		

APPENDIX

int	ObjRedraw() arguments: objectp	gravity2.c
	Redraws an object on the screen. So as not to erase other objects on the screen, the seen_array objects in the EM_SQ workspace are drawn in the INV_EM_SQ and used to erase the EM_SQ pmd before redrawing.	
objects	ObjCopy() arguments: from_object	trans03.c
	Copies an object.	
int	ObjMove() arguments: objectp, seen_arrayp	mvobj01.c
	Interactive loop for moving an object on the screen. In order to not erase other objects, the seen_array objects are drawn in the INV_EM_SQ and BitBlt to the screen.	
objects	ObjMirror_V() arguments: objectp	opers.c
	Vertical mirror transformation of an object. Repositions object to original x object offset with XTrans().	
objects	ObjMirror_H() arguments: objectp	opers.c
	Horizontal mirror transformation of an object. Repositions object to original y object offset with YTrans().	

APPENDIX

int	ObjRotate() arguments: ringp, angle, centerx, centery	trans03.c
	Uses a matrix transformation to rotate the points of an object given a ring list of cells, an angle, and a center of rotation x and y.	
objects	ObjRotation() arguments: objectp, degrees	opers.c
	A second object rotation function. This does not use a translation matrix and is more accurate.	
int	ObjScale() arguments: ringp, centerx, centery, scalex, scaley	trans03.c
	Uses a matrix transformation to scale the points of an object given a ring list of cells, a center x and y and an x and y scale factor.	
objects	ObjScale_X() arguments: objectp, scale_factor	opers.c
	Scales object points in x by a given scale factor.	
objects	ObjScale_Y() arguments: objectp, scale_factor	opers.c
	Scale object points in y by a given scale factor.	

APPENDIX

objects	ObjTrans_X() arguments: objectp, translate_value	opers.c
	Translates object points in x by a given translation value.	
objects	ObjTrans_Y() arguments: objectp, translate_value	opers.c
	Translates object points in y by a given translation value.	
objects	LeftAlign() arguments: static_object, move_object	align.c
	Aligns the left of move_object to the left of static_object.	
objects	RightAlign() arguments: static_object, move_object	align.c
	Aligns the right of move_object to the right of static_object.	
objects	TopAlign() arguments: static_object, move_object	align.c
	Aligns the top of move_object to the top of static_object.	
objects	BottomAlign() arguments: static_object, move_object	align.c
	Aligns the bottom of move_object to the bottom of static_object.	

APPENDIX

objects	SquareAlign() arguments: objectp, vert_align_y	align.c
	Aligns the bottom of an object to a given vertical alignment line y value.	
objects	LeftOf() arguments: right_objectp, left_objectp	opers.c
	Translates left_object and positions it to the left of right_object. Recontours the two objects to create a single object with ObjGlue_X(). Returns the new joined object. Syntax: to the left of right_object put left_object.	
objects	RightOf() arguments: left_object, right_object	opers.c
	Translates right_object and positions it to the right of left_object. Recontours the two objects to create a single object with ObjGlue_X(). Returns the new joined object. Syntax: to the right of left_object put right_object.	
objects	BottomOf() arguments: top_object, bottom_object	opers.c
	Translates the bottom_object and positions it at the bottom of top_object. Recontours the two objects to create a single object with ObjGlue_Y(). Returns the new joined object. Syntax: at the bottom of top_object put bottom_object.	

APPENDIX

objects	TopOf() arguments: bottom_object, top_object	opers.c
	Translates the top_object and positions it on top of bottom_object. Recontours the two objects to create a single object with ObjGlue_Y(). Returns the new joined object. Syntax: on the top of bottom_object put top_object.	
int	ObjGlue_X() arguments: left_objectp, right_objectp	glutemp2.c
	Join two objects by appending their lists of points to create a single list. This new list is created by finding the upper left x cell of the left object, initializing it as the first point in the new list, and then adding each next point until the upper right cell of the left object is reached. The cells of the right_object are then appended to this list, beginning with the upper left cell of the right_object, and continuing until the lower left cell of the right_object is reached. The remainder of the list of cells belonging to the left_object is then appended, beginning with the lower right cell until the original starting cell of the left_object is reached. This list is a list of cell points to be displayed on the screen, each referred to as a dcell. This list contains pointers to the cpt pointers of each part object. Therefore, part coordinates are automatically updated each time a letter or part group contour is manipulated.	
int	ObjGlue_Y() arguments: top_objectp, bottom_objectp	glutemp2.c
	Same as ObjGlueX() except that this function is used to recontour a top and bottom object. The following sequence of points is searched for and appended: the upper right cell of top_object	

APPENDIX

to the lower right cell of top_object, the upper right cell of bottom_object to the upper left cell of bottom object, to the lower left cell of top_object, to the origin point.

objects	CrHostObj() arguments: objectp	glutemp2.c
	Creates a pixel map in the invisible frame buffer, draws an object, and BitBl'ts the object to the host memory. A pmd is created in host.	
objects	ScaledObjCreate() arguments: objectp	objdisp1.c
	Scales a copy of object and calculates and sets its scaled offset and extent. This function is used for creating scaled objects to display in the SCALED, VIEW, and TEXT workspace areas.	
int	ScaledObjDisplay() arguments: objectp	objdisp1.c
	Displays a scaled object outline in the SCALED workspace area.	
int	ObjOverlap() arguments: objectp, test_object	mvobj01.c
	Tests to see if object and test_object overlap.	
int	ObjDims() arguments: objectp, *ox, *oy, *ux, *uy	mvobj01.c

APPENDIX

	Returns the origin x and y and the upper x or y coordinate values of an object.	
cells	UpRightPtGet() arguments: objectp	glue4.c
	Returns the upper right cell of an object.	
cells	UpLeftPtGet() arguments: objectp	glue4.c
	Returns the upper left cell of an object.	
cells	LowRightPtGet() arguments: objectp	glue4.c
	Returns the lower right cell of an object.	
cells	LowLeftPtGet() arguments: objectp	glue4.c
	Returns the low left cell of an object.	
int	RecalcOffExt() arguments: objectp	rparts3.c
	Recalculates object offset and extent and resets object point coordinates relative to these new values.	

APPENDIX

int	Obj_Max_Min_Get arguments: objectp, *lowx, *lowy, *highx, *highy	rparts3.c
	Returns the low and high x and y coordinate values of an object.	
int	Obj_Min_X() arguments: objectp	opers.c
	Returns the lowest x value in an object list of points.	
int	Obj_Min_Y() arguments: objectp	opers.c
	Returns the lowest y value in an object list of points.	
int	Obj_Max_X() arguments: objectp	opers.c
	Returns the highest x value in an object list of points.	
int	Obj_Max_Y() arguments: objectp	opers.c
	Returns the highest y value in an object list of points.	
int	ObjBltnit() arguments: offsetx, offsety, extentx, extenty, *visbltpmdname, *vispmdname, *invbltpmdname, *invpmdname	mvobj01.c

APPENDIX

Creates a pmd in the EM_SQ workspace pmd and in the INV_EM_SQ pmd of the same size for BitBlit.

Ring Structure

```
typedef struct _rings rings
        cells          firstcell;
        int             numcells;
```

Ring Functions

```
rings RingInit()                                crobj03.c
arguments: none
```

Allocates space for a ring structure and initializes all pointers to NULL and all integers to zero.

```
int RingTransform()                             crobj03.c
arguments: transformation_matrix, ringp
```

Applies the transformation_matrix to the ring of cell point coordinates.

Cell Structure

```
typedef struct _cells cells
        struct _cells *previous, *next;
        cpt;
```

Cell Functions

APPENDIX

cells	CellInit() arguments: none	crobj03.c
	Allocates space for a cell structure and initializes all pointers to NULL and all integers to zero.	
cells	DCellInit() arguments: objectp, ctp	glutemp2.c
	Allocates space for a dcell, and initializes its ctp to the incoming ctp.	
int	CellInsert() arguments: cellp, previous_cellp, objectp, numcells	crobj03.c
	Inserts a cell into a two way list of cells. Can insert a cell after any given previous_cell. If the previous_cell is NULL, the cell is inserted as the firstcell in the ring list.	
cells	CellLstEndGet() arguments: ringp	crobj03.c
	Returns the last cell in a one way ring list of cells.	
cells	CellGet() arguments: cellp, numcells	mvspt08.c
	Returns a cell that is a certain number of cells away from a given cell (cellp).	

APPENDIX

int	CellArrGet_X() arguments: objectp, xvalue, cell_array, *xcellnum Returns an array of cells that contain a given xvalue.	glue4.c
int	CellArrGet_Y() arguments: objectp, yvalue, cell_array, *ycellnum Returns an array of cells that contain a given yvalue.	glue4.c
cells	SelectPoint() arguments: objectp, rcursx, rcursy Tests to see if a cell point has been selected. If no point selected , this function returns NULL, else it returns the cell.	select05.c
int	PtsDraw() arguments: objectp Display the points of an object on the screen in the EM_SQ workspace.	rparts3.c
int	DrawInvPoints() arguments: objectp, inv_pmd_name Draws the points of an object in a pmd in the INV_EMSQ.	glutemp2.c
int	DrawInvPoint() arguments: coordx, coordy, type, pmdname Display a point of an object in any pmd.	rparts3.c

APPENDIX

int	DrawPoint() arguments: coordx, coordy, type	gravity2.c
	Draws the marker for a given point on the screen.	
int	ErasePoint() arguments: coordx, coordy, type	gravity2.c
	Erases the marker for a point on the screen.	
int	MovePoint() arguments: objectp, cellp	gravity2.c
	Interactive loop for moving a selected point on the screen.	
int	CellMove() arguments: seen_arrayp, objectp, cellp	mvsppt08.c
	Interactive loop for graphically manipulating a cubic spline curve segment on the screen. So as not to erase the visible objects on the screen, the seen array of objects is drawn in the INV_EM_SQ area and BitBlt to the screen prior to redrawing the cubic spline curve.	
int	PtAlign() arguments: objectp, x_or_y	gravity2.c
	Interactive loop for aligning points on the screen. Align the x or y coordinate of any number of points to the x or y coordinate of a reference point. The first point selected is the reference point. The syntax is: align to this point, that point, that point	

APPENDIX

and that point, etc.

int	PtGravity() arguments: objectp, alignment_line_y	gravity2.c
	Interactive loop for aligning the y coordinates of any number of points to any vertical alignment line.	
int	PtTypeChange() arguments: cellp	gravity2.c
	Resets the type of point in a cell. Sets an if_changed flag to TRUE.	
int	PtTypeEdit() arguments: objectp	gravity2.c
	Interactive loop for editing the type of a selected point. Erases the old marker type and displays the new one.	

Cpt Structure

typedef	struct _crds	*cpts;
	int	coordx, coordy;
	char	type;
	int	crdflag;
	int	ifchanged;
	int	ifjoined;

APPENDIX

Cpt Functions

int	CellCrdInit() arguments: cellp, rcursx, rcursy	crobj03.c
	Sets cpt coordinates to x and y. X and y are in EM_SQ coordinates.	
int	CoordConvert() arguments: objectp, oldoffsetx, oldoffsety	gravity2.c
	Recalculates object point coordinates relative to an oldoffsetx and oldoffsety to newoffsetx and newoffsety.	
int	RelCrdsCalc() arguments: objectp	crobj03.c
	Resets coordinates of object curve control points relative to the object offset. Sets the relative coordinate flag to TRUE.	
int	RelXCrdCalc() arguments: objectp, x	opers.c
	Subtracts x from all x coordinates.	
int	RelYCrdsCalc() arguments: objectp, y	opers.c
	Subtracts y from all y coordinates.	

APPENDIX

int **ToEmCrds()** `opers.c`
arguments: objectp

Resets all object coordinate values to EM_SQ workspace coordinate values.

ACKNOWLEDGMENTS

I would like to thank IBM Corporation for sponsoring this research project. Muriel Cooper and Ron MacNeil have been supportive advisors. Thesis Committee members and readers Patrick Purcell, Steve Benton, Lois Craig, Henry Lieberman, Matthew Carter and Fanya Montalvo provided much needed help and direction. Bitstream, Inc. furnished software assistance, technical information, training, and guidance.

To all my friends, I can't thank you enough. Without your constant emotional support and contributions of time and energy I may not have made it. I would like to give special mention to Joan Shafran, Rob Haines, Pam and Paul Paternoster, Bridget Johnson, David Levy, Eckhard Felsmann, Jim Davis, Wendy Katz, Susan Wascher, Liz Rosenzweig, Eileen Baird, Lin-Lin Mao, Alka Badshah, Johnathan Linowes, Tim Shea, Jim Puccio, Allison Druin, Tom Amari and Russ Sassnett. Thanks to Bridget for rare historical references. Jim Davis has been an excellent editor and our discussions have helped to clarify my thinking. Jim is to be credited with the "abcdefg" software package title. Joan Shafran, Susan Wascher, and Allison Druin donated valuable graphic assistance. Steven Paul, Phil Apley, and Larry Oppenburg spent long hours talking with me.

ACKNOWLEDGMENTS

Steven contributed code to the project and helped with the software design. Lynn Ruggles responded to many long distance phone calls. Richard Southall helped me to understand Metafont. Thanks to Dorothy Dehn and to my family for their belief in me. And, finally, I wish to thank the undergraduate students who worked with me on this long project: Shih-Lin Liu, Paul Chernock, Terry Fong, Rick Poyner, John DeRoo, Chris Lombardi, Bosco So, and Dan Lin.