PARAGON, AN INTERACTIVE, EXTENSIBLE, ENVIRONMENT FOR TYPEFACE DESIGN

A Dissertation Presented
By
Lynn Elizabeth Ruggles

COINS Technical Report 87-134

PARAGON,
AN INTERACTIVE, EXTENSIBLE, ENVIRONMENT FOR TYPEFACE DESIGN

A Dissertation Presented
By
Lynn Elizabeth Ruggles

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1987

Department of Computer and Information Science

This research was supported in part by the National Science Foundation under grant IST-820-1926 and by the Systems Development Foundation.

PARAGON, AN INTERACTIVE, EXTENSIBLE ENVIRONMENT FOR TYPEFACE DESIGN

A Dissertation Presented

By

Lynn Elizabeth Ruggles

Approved as to style and content:

[illegible]
David McDonald, Chairperson of Committee

[illegible]
Allen Hanson, Member

[illegible]
Sandy Hill, Member

Charles A. Bigelow
Charles A. Bigelow, Member

[illegible]
W. Richards Adrion, Chairperson
Dept. of Computer and Information Science

For my parents

Acknowledgements

I owe a debt of gratitude to the following people for their support, encouragement, and advice, given above and beyond the call of duty over the last *n* years:

- Special thanks to Donald Knuth for his invitation to join the Digital Typography Group at Stanford University, for his unending support and encouragement while I was here, and for the bread on my table provided through his TEX grant.

- Special thanks also to Chuck Bigelow for his elucidation of matters both typographical and philosophical. His enjoyment and knowledge of letterforms in all their varied shapes has been a resource that no library could match.

- The members of the Digital Type Group were a perpetual source of inspiration, entertainment and croissants. My thanks to the team: Carol Twombly, Dan Mills, Dave Siegel, Dave Fuchs, and Cleo (who?) Huggins.

- For discussions on the art of type design, I am most grateful to Richard Southall, Matthew Carter and Gerard Unger.

- For her gentle polishing touch, my thanks to Ann Rosener whose editing (and nit-picking) changed all my *which*'s into *that*'s.

- Thanks also to Renée Stephens for her patience and persistent good cheer in getting me through the intricacies of long-distance paperwork.

- And, for seeing me through it all, my thanks to Andy Cromarty.

Lynn Elizabeth Ruggles
September, 1987

ABSTRACT

Paragon, an Interactive, Extensible, Environment for Typeface Design

September 1987

Lynn Elizabeth Ruggles, B.A. University of Virginia
M.S., University of Massachusetts
Ph.D., University of Massachusetts
Directed by: Professor David McDonald

Typefaces have been designed and used for over 600 years. As new technology was developed, new methods were designed to cope with the changes in materials and techniques. The current century has seen the development of raster output devices in the form of laser printers, CRT typesetters, and bitmapped graphics displays. Systems used to generate digital type designs for these devices generally fall into the category of copying systems rather than design systems. Little work has been done to provide the type designer with a system intended to be used for the design of new typefaces.

This thesis discusses a typeface design system which has been created to fill this gap. The system, named Paragon, is an interactive, extensible, typeface design environment implemented on an interactive graphics workstation. It is an attempt to integrate the traditional typeface design environment with the capabilities of interactive computer graphics. It enhances the traditional environment by providing functions to aid the design process which are difficult to do by hand, but easy to do within a computer system. The design system itself is extensible in that a designer who has little if any knowledge of computers or programming can create new commands by using the the primitive operations that are provided and can then incorporate these commands into the working environment.

TABLE OF CONTENTS

List of Figures

Time keeps on slipping, slipping, slipping, into the future....
The Steve Miller Band, 1968

Chapter I

Introduction

The work of a copyist can present only the infelicities and mannerisms of the former worker, and will lack entirely the human stamp of life and variety, and the expression of joy so evident in the old work. No art can live by the continued reviving or reproduction of the achievements of the past. To what source, then, or to what quarter shall we turn for suggestions for the new type creations?

Frederic W. Goudy, introduction to *Type Design* by D. McMurtrie

Paragon is an interactive, extensible, typeface design system implemented on an interactive graphics workstation. It is an attempt to integrate the traditional typeface design environment with the capabilities of interactive computer graphics. It gives a type designer sitting at a workstation the ability to design typefaces directly: there is no provision for entering existing designs into the system, nor are there any features which enable a user to do bit-editing of raster characters. All type rendering is done interactively using a pointing device and a bit-mapped graphics screen. Features which are specific to type design and display have been incorporated into the design environment. The system is extensible in that a designer who has little if any knowledge of computers or programming can create new commands by combining the primitive operations that are provided and can then incorporate these new operations into the working environment.

The system has been named *Paragon*, after an early name for a type font size of approximately 20 points. Prior to the adoption of the *point system* in the late-nineteenth century, type sizes were identified by names which were not assigned with any attempt at precision. Type sizes were known by names such as Pearl, Ruby, Agate, Nonpareil, Minion, Brevier, Galliard, Bourgeois, Long Primer, Small Pica, English, Great Primer, Canon, and Gros-Canon, with each name denoting a different size. The names and sizes varied from country to country and came from many sources; some took their names from the book in which the type first appeared (e.g. Canon, Primer); others took their names from the name of another typeface of similar design. Some took their names from common names of the day, such as Galliard, which was originally the name of a dance popular at the time the typeface was designed.

This thesis has been divided into six chapters and two appendices. Chapter 2 covers the history of the type design process up to the advent of computer-controlled typesetters. Chapter 3 continues the discussion with a review of type conversion systems that were developed to take advantage of high speed processors, graphics hardware, and digital display devices. Chapter 4 describes what we mean by an *interactive, extensible design environment*, and enumerates the features which are necessary for a production type-design system. Chapter 5 describes the implementation in detail and provides specific examples of both the interactivity of the system and its extensibility. It also summarizes the successes and limitations of the system and discusses where these might be optimized or eliminated. Chapter 6 concludes the thesis with a brief summary of what we have accomplished. Appendix A provides a lexicon of typographic terms, and Appendix B contains an explanatory description of each of the functions provided with the system.

Chapter II

The Evolution of Typeface Design

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 typecasting.

Hermann Zapf, *About Alphabets*

Many changes have taken place in the design of typefaces because of technological advancements in the printing industry. As new printing machines were introduced, new typeface designs were crafted to take advantage of improvements or changes in the printing methodology. Some developments in the printing trade led to changes in typeface designs because of improved methods of printing. Others influenced the designs because they modified the characteristics of materials used by the printing industry. Other inventions prompted changes in the design process itself.

As is usual when there is technological change, processes that were indirectly dependent on the new equipment did not change immediately upon announcement of the new developments. This delay resulted in inefficiency in the design process until the new technology's capabilities and limitations were explored. When the characteristics of the new technology were understood, the design methodology changed to make the design process conform to easier and faster methods of production.

This chapter will give an overview of the metamorphosis of typeface design from its early beginning as a trade practiced by artisans working with simple hand tools up to the development of computer fonts for phototypesetters. Some typefaces will be mentioned as examples, but this is not meant to be an exhaustive survey of all typeface designs that have been influenced by technology nor of all of the technology that has led to changes in typeface design. Since much of the actual operations of printing and typesetting equipment has been covered in other publications, only a brief mention of it will be made here; a more detailed explanation of the machines or the manufacturing methods can be had by consulting the references.

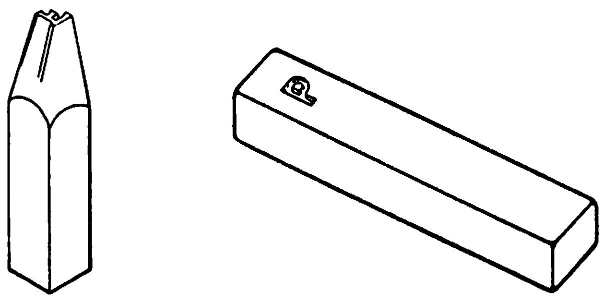


Figure 1. Punch and matrix for use with a handmould for casting metal type.

Early type design

Johannes Gutenberg was the first person to fully develop the idea of letters cast on the end of a piece of metal, now referred to as *type*, and a means of casting these pieces in a uniform and easily replicated manner. Although not much detail is known about his early experiments, he had developed the technique sufficiently by 1450 to be producing enough type to print an edition of the Bible. The process of *casting type* or *type-founding* required several steps for each individual character. The first step was to carve a letter shape on the end of a bar of steel, called a punch, which was then impressed into a bar of copper to form a matrix, leaving an impression of the letter in the copper (Figure 1). This matrix was inserted into a hand mould into which molten metal (mostly lead) was poured. The metal filled in the impression in the copper matrix, so that when the metal cooled and was removed from the mould, there was a raised character shape, or *face*, on the end of the piece of type. After the hardened metal was removed from the mould the edges were squared off . A punch and a matrix had to be made for each character from which many individual types were cast. When a sufficient amount of type had been cast, they were *set* or lined up side by side to form words. Many lines of type were stacked one below another to form a page of text which was then printed on a printing press [KOCH33, DROS85].

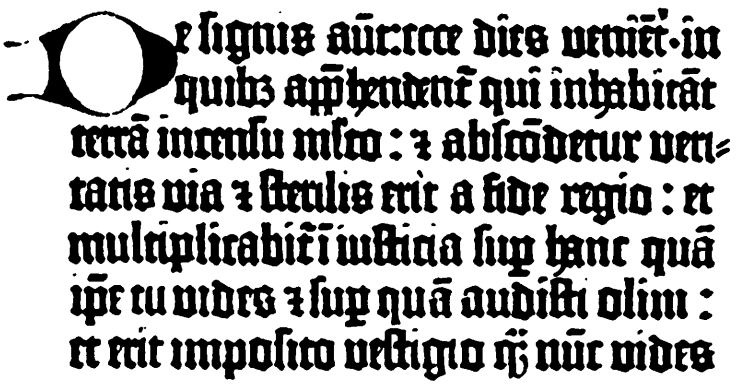
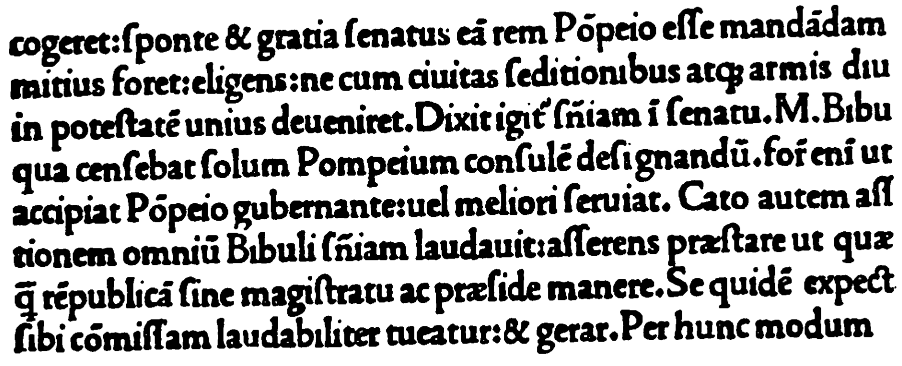

e ſignis auctore dies veniēt·in
quibꝫ apphendent qui inhabitāt
terrā incenſu multo: ⁊ abſcōdetur veri=
tatis via ⁊ ſterilis erit a fide regio: et
multiplicabit̄ iniuſticia ſup hanc quā
ipſe tu vides ⁊ ſup quā audiſti olim:
et erit impoſito veſtigio q̄ nūc vides

Figure 2. Typeface from printed Bible, 1455.

Gutenberg modeled his typefaces on the characters written by the scribes working in Germany (Figure 2). He made no attempt to design characters which were adapted to the new means of producing them, he simply tried to model handwritten forms in a sculpted metal form. It was not until 1470 that Nicholas Jenson, working in Italy, carved the first typeface on the end of a hard steel surface, taking advantage of the new technology by creating delicately crafted letters rather than imitations of the calligraphic forms of an earlier era (Figure 3).



cogeret:ſponte & gratia ſenatus eã rem Põpeio eſſe mandãdam
mitius foret:eligens:ne cum ciuitas ſeditionibus atq; armis diu
in poteſtatẽ unius deueniret. Dixit igit̃ ſñiam ĩ ſenatu. M. Bibu
qua cenſebat ſolum Pompeium conſulẽ deſignandũ. fořeni ut
accipiat Põpeio gubernante:uel meliori ſeruiat. Cato autem aſſ
tionem omniũ Bibuli ſñiam laudauit:aſſerens præſtare ut quæ
q̃ rẽpublicã ſine magiſtratu ac præſide manere. Se quidẽ expect
ſibi cõmiſſam laudabiliter tueatur:& gerar. Per hunc modum

Figure 3. Typeface designed by Nicholas Jenson, 1470.

The success of Gutenberg's invention created a need for the design of characters that were constrained to fit within a small rectangular box (the surface of a piece of metal type). These pieces of type had to fit tightly against one another to form a line of text. Occasionally, in the italic fonts, a character would overhang its box and overlap the following character (or *kern*), but these sorts were the exception rather than the rule (Figure 4).

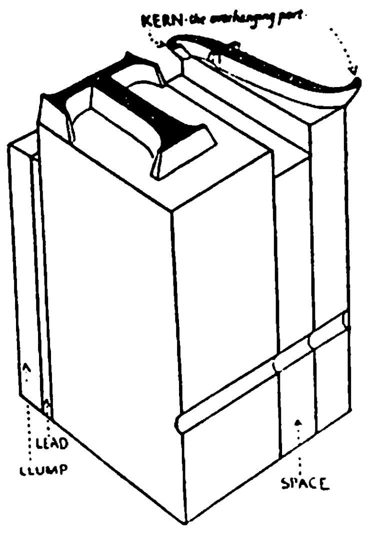


Figure 4. Kerned type.

The characters not only had to be designed to fit within a box, they also had to be placed within the boundaries of the box so that a row of characters would be spaced evenly when the the boxes were lined up. As Stanley Morison wrote, 'The letters will be so placed upon their respective bodies as to make up into words with no unlovely gaps and disturbing spaces.' Since the shapes of the letters varied, great care was taken in the design of the type, the determination of the appropriate width of the box, and the position of each character within the box so that the illusion of equal spacing was achieved. This process required good visual judgement; if the letters were placed exactly the same distance apart, the letter spacing would appear uneven, with adjacent curved letters appearing farther apart than adjacent straight characters [BLUM35, TSCH66, DOWD66].

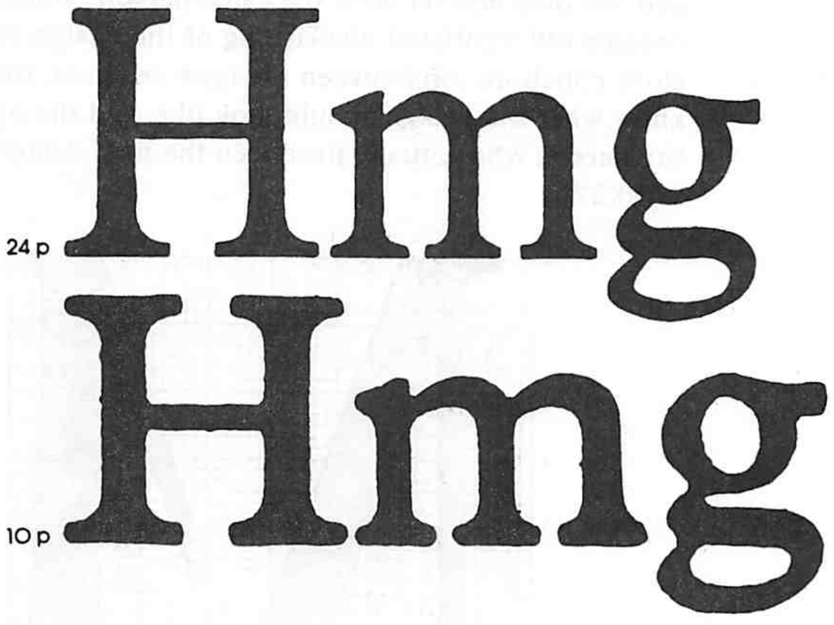


Figure 5. Difference in character shapes for 24 point typeface (above) and 10 point typeface (below). The smaller typeface is bolder, shorter, and wider, but when it is printed at its true size, it appears to be the same design as the larger size.

The letter shapes in each size of type varied slightly to compensate for optical distortion in the perception of small shapes by the human eye. The smaller types required shorter ascenders and descenders and had to be cut wider and in bolder form than their larger counterparts to give the appearance of being the same design (Figure 5). Early type designs were drawn in only one size, and it was up to the punchcutter to use his skill to interpret the design at the various sizes that were to be cut [MOXO58, KOCH33]. In the sixteenth–eighteenth centuries, most punchcutters cut designs in a complete range of sizes. The finished fonts were not identical but their shapes conformed to the spirit of the original letter. It took about six months to cut a complete set of punches and to strike and justify the matrices for just one font [POLL71].

In the early years of the printing industry, the designer and the punchcutter were the same person. When the tasks became differentiated, the cutting of the design required a close collaboration between the type designer, the person who knew what the design should look like, and the punchcutter, the person who actually produced the final design [CART54, VANK57].

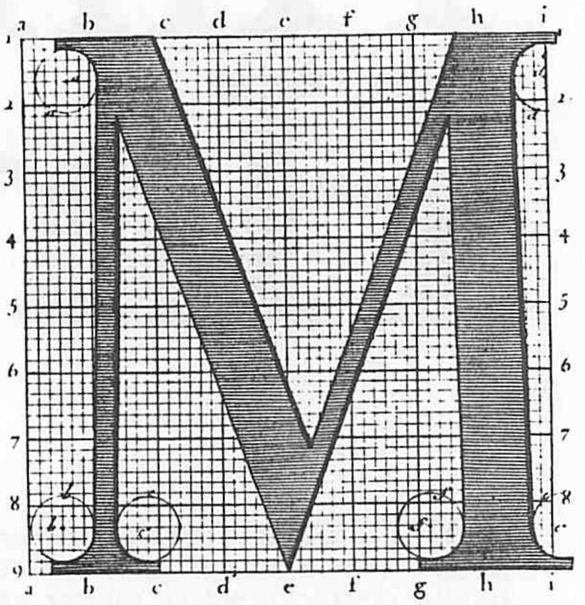


Figure 6. The 'ideal' M, designed by the Académie des Sciences, 1702.

Some attempts were made to legislate type manufacture and design but most were unsuccessful. The most notable attempt was made in France in 1692, when Louis XIV sanctioned the creation of a new set of types to be used exclusively by the Imprimerie Royale, then the royal printing house and part of the Louvre. A commission of experts was appointed by the Académie des Sciences to study the design of the perfect Roman letter. Under the guidance of its chairman, Jacques Jaugeon, a number of elaborate mathematical designs were made which were drawn with the aid of rulers and compasses (Figure 6). Each letter fit within a grid composed of 2,304 squares (48 x 48) and was drawn using a ruler and compasses. When a decision was made to turn these ideal letters into a font, the punchcutter Philippe Grandjean was employed to engrave the punches. He used the designs merely as suggestions, however, and relied more on his own experience and trained eyes than on the carefully drawn characters in determining the final shapes of the punches. The type designer Pierre Fournier later commented, 'Are so many squares need to make an O, which is round, and so many circles to make other letters which are square?' [FOUR30]. Despite a decree issued by the Imprimerie Royale forbidding the copying of the type, it was so popular that many other typefaces were modeled on its design although no other designer went through the elaborate design process employed by the royal commission. The new design technique, if it can be called that, was not adopted by practicing type designers.

Changes in materials

The paper that was used with early printing presses was of a thicker, spongier quality than paper used today. Because of its toughness and stiffness, it had to be dampened before use to soften the fibers so that the printing ink would adhere to it. It was also necessary for typefaces to conform to certain standards of design so that they would work well with the available techniques and materials [CAFL83]. According to Legros and Grant, writing in 1916, 'Handmade 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. One has only to examine old prints with the microscope to see this; under a suitable power the circumjacent surplus ink appears as a band, almost detached from the edge of the actual impression of the type itself. This defect contributed in a rather marked degree to legibility, for it tended, as has been said, to thicken the hairlines and thus render more pronounced the difference between the less dissimilar letters. The highly glazed papers of today, of short fiber, containing much sizing and mineral matter, are not adopted for printing from such irregular surfaces; their want of flexibility requires a hard and true backing, and hence increased accuracy in the printing surface in order to obtain a uniformly sharp impression.'

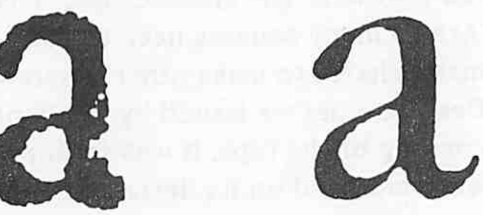


Figure 7. Comparison of letter printed on laid paper (left) and wove paper (right).

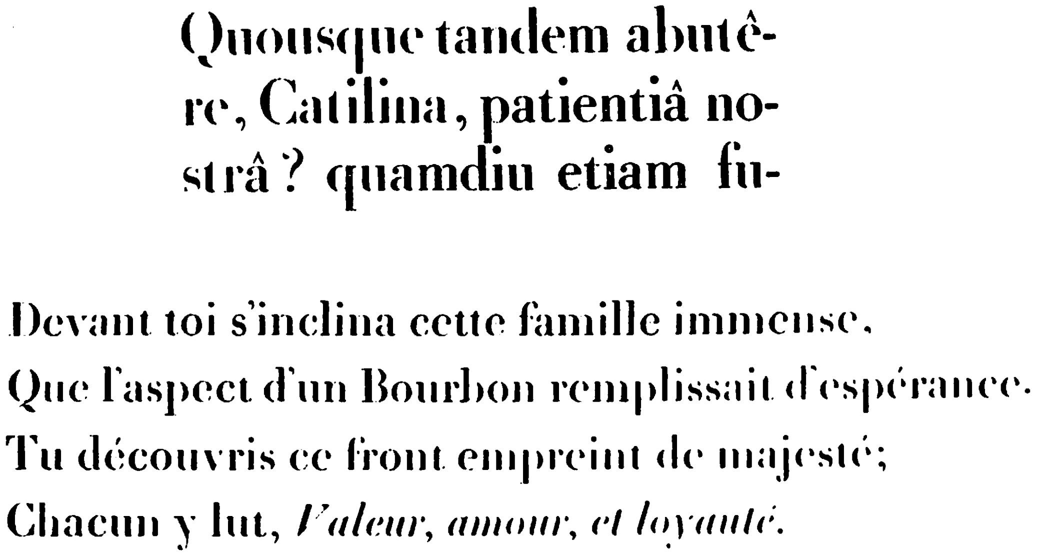
The design of a typeface by John Baskerville in the mid-eighteenth century (which was first offered for sale in 1758) was preceded by the production of new paper-making equipment which could produce *wove paper*, made on a mould containing a woven wire mesh that resulted in a finer grain and a smoother surface than the *laid paper*, made on a mould with wires laid parallel to each other and used by earlier printers (Figure 7). Another refinement at this time was the process of pressing the dampened paper between hot copper plates after it was printed. This procedure smoothed the paper even more to create a silky, shiny surface. Because the process of printing on the smooth paper rendered typefaces much more clearly than on the coarser laid paper used by other printers, the shapes of

Baskerville's types were rounder and more finely drawn than other contemporary types [WROT38, JOHN30]. The delicately drawn serifs and the contrast between thick and thin strokes could be reproduced because the paper provided a sufficiently smooth surface for the details of the face to be perceived (Figure 8). Despite the care with which Baskerville fitted his types to the technology, they were often disliked (Ben Franklin was a notable exception), the claim being made that the shiny paper he used was too glossy and dazzling to the eyes and the types too light and delicate to be read easily.

900 Qualia nunc hominum producit corpora tellus.
Ille, manu raptum trepida, torquebat in hoſtem,
Altior inſurgens, et curſu concitus heros.
Sed neque currentem ſe, nec cognoſcit euntem,
Tollentemve manu, ſaxumque immane moventem.

Figure 8. Sample of Baskerville typeface.

Another technical achievement that served to provide a more precise impression on wove paper was the development of a new printing press made entirely of iron. This press was produced around 1783 by an Englishman, Earl Stanhope. The construction of the press was essentially the same as its predecessors which were made of wood, but due to the finer machining of the parts of the press and the addition of compound levers instead of a screw mechanism, the printer could exert a much firmer control over the pressure required to print a sheet of paper. In addition, the press could print two pages at the same time, whereas earlier presses required that a separate impression be made for each page. It was this invention, coupled with the new paper-making technique, that provided the impetus for the design of typefaces by Bodoni and Didot which were much more finely cut than previous designs (Figure 9). The improvement in technology led to a change in what could be achieved and this resulted in new type designs


Quousque tandem abutê-
re, Catilina, patientiâ no-
strâ? quamdiu etiam fu-

Devant toi s'inclina cette famille immense,
Que l'aspect d'un Bourbon remplissait d'espérance.
Tu découvris ce front empreint de majesté;
Chacun y lut, *Valeur, amour, et loyauté.*

Figure 9. Sample of Bodoni typeface (above) and Didot typeface (below).

whose characteristics would not have been noticeable with the older processes.

It was at this time that proposals were made to standardize type sizes through a measurement based on the *point system.* Up until this time, type designers had no standard measurement for specifying the height of the letters in a typeface. Each designer and each manufacturer had his own standards and names for sizes which did not necessarily correspond to other manufacturer's names or measurements. The point system was initially proposed by Pierre Simon Fournier in his book *Manuel Typographique* published in 1737. He proposed a system of 12 lines per inch with 6 points per line. Type height would be measured in points. His proposal was re-introduced by Didot around 1770 who suggested that the measurement be based on the *pied du roi,* or the French government standard foot which was slightly larger than the English foot.

The development of the point system was an attempt to promote some order upon the chaos of type sizes generated by type foundries all over Europe, but it was not generally adopted in most countries until the late eighteenth century. At that time, the United States and England adopted a measurement based on the English foot, now called the *standard* point system, with 6 points per *pica* and 12 picas per inch (approximately 72 points per inch). The French adopted the same unit divisions, but based their measurements on the (larger) French foot; their measurements are made in *Didot points* and *ciceros*.

This attempt to standardize the printing trade was adopted well after printing and its associated technology had been around for several hundred years. Its acceptance was strengthened by the development of new typesetting machinery which relied on very precise measurements.

Standardization of type designs

The mid-nineteenth century gave rise to many inventions related to automated typecasting and typesetting. One problem that had to be solved before automatic typesetters could be successful was that of spacing a line of text evenly. When a line of text was to be *justified* or set so that both the left and right edge of the text were flush with the margin of the page, the words within a line had to be spread out so that the space between them was evenly distributed. To do this successfully when setting type by hand, printers employed very small spacing increments which were sometimes as small as one point thick (1/72 of an inch). The problems involved in doing this mechanically were very complex and thus hindered the development of a mechanical means of typesetting.

As an aid speeding up the time it took to hand-set text, Linn Boyd Benton, working in the United States, and Alois Auer, in Vienna, each devised similar spacing systems around 1830 [LEGR16, MENG54]. Benton's system was called *self-spacing* type. All characters were to be designed according to a unit grid which was a multiple of a small unit size. Any combination of characters could be made up to a multiple of the em (a measure equal to the body size of a font; for example, in a 12-point font, the em is equal to 12 points) by the addition of spaces which were also made up in unit sizes.

The idea of a unit grid for use in the design of characters was not adopted by the printing community at that time because the limited number of set sizes did not allow for the design of characters which were sufficiently close in design to those in popular use. It was, however, accepted for use with automated typesetters which were developed about 50 years later. At that time, use of self-spaced type was one of the crucial factors in the development of equipment that could set and justify large quantities of type automatically. To justify a line of text, spacing material had to be added to a line to fill it out to the desired width. The spaces had to be distributed evenly across the line so that the line filled the full column width. Because the widths of the letters were not standardized, the spaces that needed to be added to a line could vary in increments of a single point. The adoption of unit spacing simply increased the size of the smallest unit. All letters had to be designed to fit some multiple of this unit, and spacing was available in multiples of the unit size. The reduction in the number of spacing units that were needed made the task of justifying text more manageable, and thus able to be done by machines. The increase in typesetting speed that resulted from this standardization was more important to the printing industry than the reduced legibility of the typefaces that were used with the new machines*.

The mechanisms of the first typewriters (in the 1830s) were designed to use metal type (such as that used with printing presses) to produce an impression of letters on a page. Spacing of text on a page was difficult to control, though, so these machines were not very successful. Later models used a version of Benton's self-spacing type. A counter kept track of how wide a character was and the carriage was advanced according to the number of units in the letter that was typed. Eventually, monospaced characters, or characters that were all the same width, were used since they solved the problems associated with spacing a line of text. Since all the units were equal, it was easier to keep count of the number of letters already typed on a line, the amount of space left in the line, and the unit size of the next letter in the line. Additionally, the correction of mistyped characters was facilitated since corrections could be made

* This change in design is carried over into the design of typefaces for the current generation of digital printers in that the widths of digital characters are a multiple of a fixed pixel size. The pixel size varies according to the resolution of the output device, but in all cases, each character's width is constrained to fit within a unit grid. In addition, the space between characters must be some multiple of the pixel size.

within the same space as the original error. The only drawback to this solution was that all characters had to have the same width, a constraint which resulted in designs with exaggērated serifs on narrow characters to stretch them out, and wide characters that were so condensed that there was little space between their stems. Despite the reduced legibility of these typefaces, they were accepted by the general public because of the overwhelming convenience of the writing machine.

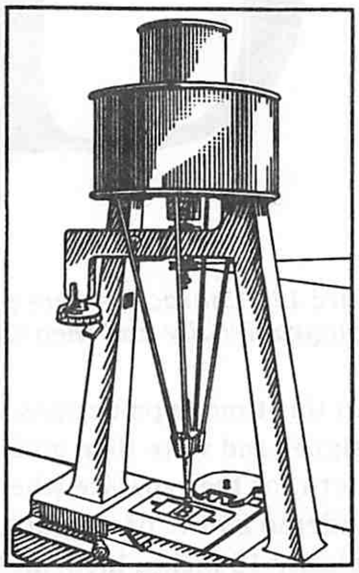


Figure 10. Pantograph machine invented by Linn Boyd Benton in 1885.

Benton's patent of the pantograph machine, in 1885, led to radical changes in the process of designing new typefaces and the production of metal type. The pantograph consisted of two parts: a stylus which was used to trace around a large pattern, and a very fine drill which was controlled by the stylus and used to cut the pattern in a reduced size on a small piece of metal (Figure 10).

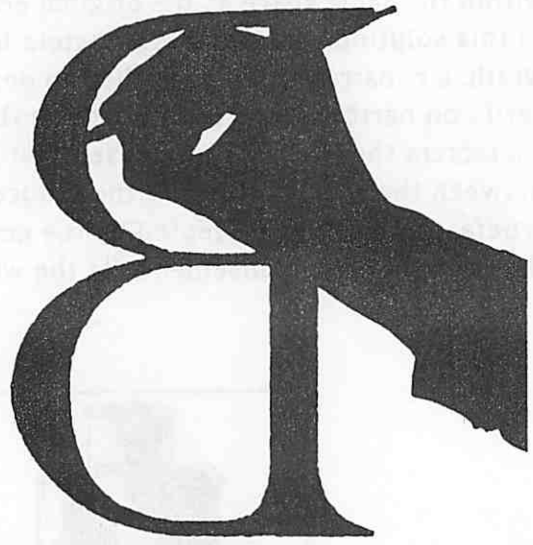


Figure 11. Characters were projected and enlarged photographically and then traced.

Up to this time, type designs were draw in in a small size by a designer and were then given to a punchcutter to be used as a pattern for the typeface when it was cut in metal. Designs to be rendered by the pantograph had to be drawn in a large size (typically 10 inches high) and had to be very precise. These enlarged type designs were produced either through tracing photographic enlargements or by obtaining original designs drawn in a large size (Figure 11). As described by Jan van Krimpen [VANK57], these drawings went through three stages before they were cut into moulds. The first stage was done in pencil so that it could be easily erased and corrected, and as a result the design was not very exact. The next or middle stage was also in pencil but the lines were more accurately drawn. The final stage was in carbon ink and consisted of filled-in character designs. This final drawing had, according to van Krimpen, lost most of its interest. It was too mechanical and precise and showed the least amount of the designer's intentions. Van Krimpen felt that the dehumanized type designs which resulted from the mechanical punchcutting did not result from the mechanization of the punchcutting task, but from the mechanized drawings which had lost all semblance of the sprightliness of the original and thus heralded the movement of type design from a creative process to an engineering one [VANK72].

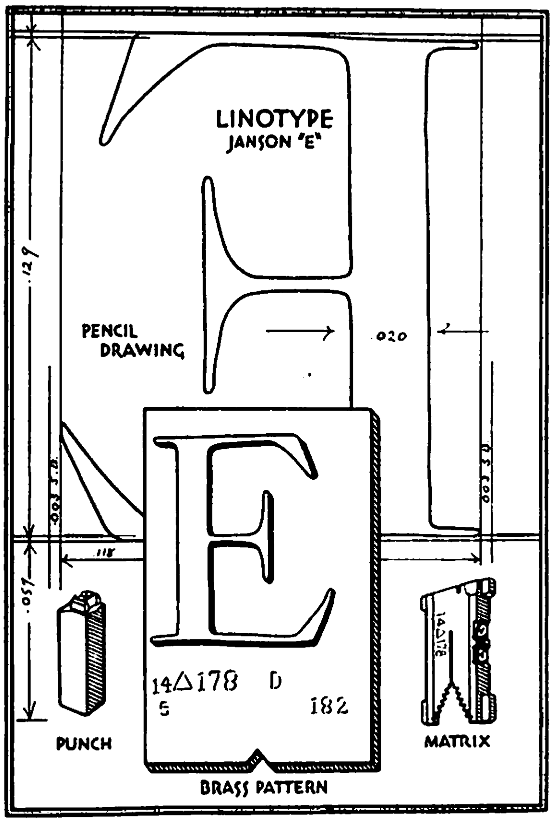


Figure 12. Character drawn for use with pantograph machine (top), brass matrix of the character (bottom center), punch cut by the pantograph (bottom left), and matrix cut by the pantograph (bottom right).

After the designer had created an accurate freehand drawing, it was then given to a drawing office whose draughtsmen redrew the letter using straight edges and French curves. Once the drawings were finished, they were photographically reduced to exact type size and positioned into words and lines of text. Adjustments were made to the drawings until an acceptable design was achieved. The designs were then made into patterns for each of the various sizes that were to be cut by the pantograph machine (Figure 12). All the measurements for the design were precisely calculated, and all characters were drawn so that components such as the stem width, size of the serifs, and the x-height corresponded to the measurements. These drawings had to be very accurate, as the pantograph had a resolution of .0015".

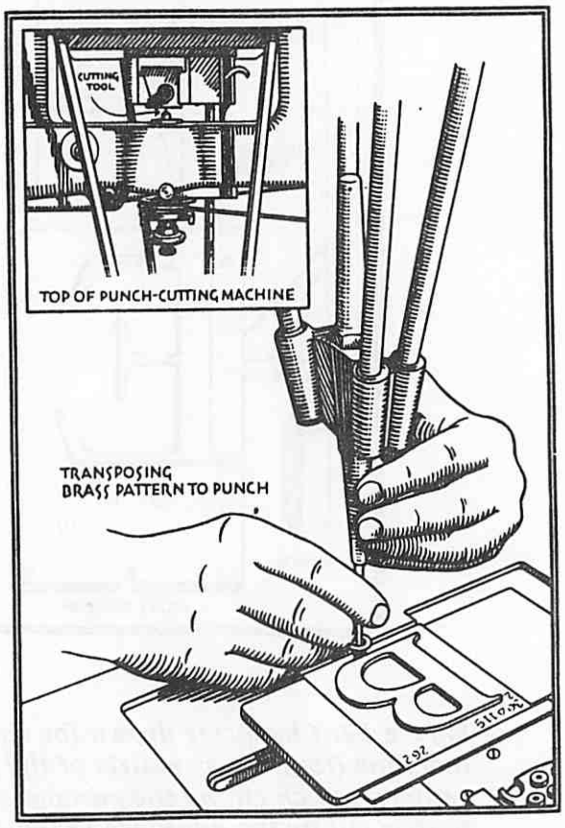

CUTTING TOOL
TOP OF PUNCH-CUTTING MACHINE
TRANSPOSING BRASS PATTERN TO PUNCH

Figure 13. Tracing the brass matrix to make a punch.

After the drawings were finished, the design was cut on the pantograph. The first stage in the cutting of matrices or punches consisted of tracing the letter drawing upon a plate; then, depending on whether a punch or a matrix was to be the final result, either the background or the character was cut away. In the second stage of the process, the resulting pattern was used as a guide for a cutting tool on the machine which either etched the character shape into a matrix, or carved the shape on the end of a bar of steel to make a punch (Figure 13).

Type designers were not very happy with the new machinery and the resulting restrictions imposed on new type designs and were almost unanimous in criticizing the new means of producing fonts [WARD35, VANK57, ZAPF65]. They were also unhappy that the collaboration between designer and punchcutter had now been reduced to a manufacturing process where a designer was simply the starting point of an assembly line consisting of an anonymous, insensitive drawing office. It was at this point that the designer lost control over the design since he usually did not collaborate with the drawing office which produced the oversized versions of the type that were eventually cut on the machine. He also had no control over design changes when the typeface was rendered in many different sizes. Eric Gill wrote in his *Essay on Typography* [GILL36], 'It is difficult enough for the designer to draw a letter ten or twenty times as large as the actual type will be and at the same time in right proportion; it requires very great experience and understanding. It is quite impossible for a set of more or less tame employees...to know what a letter enlarged a hundred times will look like when reduced to the size of the intended type.' In his book *On Designing and Devising Typefaces*, van Krimpen complained, 'The punch cutting machine, having fallen into the hands of engineers who know nothing about letter design, is neither being applied nor used in the best possible way. It certainly seems to be discouraging to try and teach these engineers to at least understand something about letter design and then to use their machine in a better way. I have tried and am trying still but, having a few more things to do I can not devote too much time to it; and so I doubt I will have any success.' William Morris stated it more succinctly, "Letters should be designed by an artist, not an engineer."

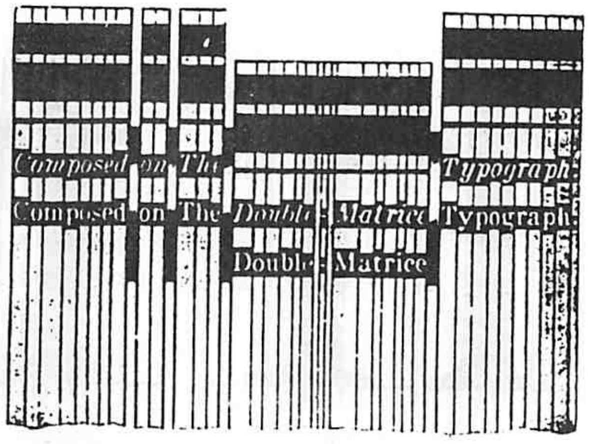
In a process contrary to that employed by most designers working with these machines, William Dwiggins retained control over his designs throughout the design process [DWIG40]. He started a design by drawing a couple of control characters, which he gave to the design office to make into punches and matrices which could then be cast as type. After determining that the sample design was worth developing, Dwiggins drew the remaining characters, sometimes modifying weights and serifs. His final drawings were all done freehand but at a much smaller size than that required by the pantograph procedure. Dwiggins relied on the drawing office to provide the precise drawings needed for use with the machine.

Typecomposition by machine

In 1884, Ottmar Mergenthaler invented the Linotype machine which cast an entire line of type in a single operation. Each character in the line was selected by a keystroke on a keyboard. Each keystroke released a brass matrix which was then lined up next to the matrices for the other letters in the line. Special wedges were inserted by the machine between words. After the entire line of type was assembled, the wedges were adjusted so that the line was justified. The line was then cast in molten lead. Rows of these lines were stacked to make a page of type. Any corrections to a line required that the entire line be reset and recast.

To facilitate the use of different fonts within a line, each matrix contained the pattern for two characters. These letters had to match in width. The most common pairing used was an italic type with a roman type. Traditionally, italic types were narrower than their companion roman type, but since the pairing of two styles on one matrix required that the two designs match in width, italic types that were wider than the older versions were designed for the new machinery. The resulting italic fonts did not space as well in a line (Figure 14).

In 1887, Tolbert Lanston patented the Monotype machine which cast single pieces of type to form a line of text. The text was first typed on a machine consisting of a keyboard and a paper punch. Character, spacing, and positioning information


Composed on The Double-Matrice Typograph
Composed on The Double-Matrice Typograph

THE LINOTYPE composes, justifies,
THE LINOTYPE composes, justifies,

Figure 14. Alignment of roman and italic type on the same Linotype matrix (above), type set on a Linotype (below). Note that the italic is very widely spaced.

was generated on a perforated paper ribbon. This ribbon was then used to drive the typecaster by means of forced air flowing through the perforations on the tape. The typecaster contained a square plate on which was etched a 16 x 16 grid. Each square of the grid contained the matrix for one character (Figure 15). A particular pattern of holes in the paper ribbon corresponded to a character in the grid. When the typecaster detected a particular pattern on the ribbon, the corresponding character was positioned on the typecaster so that a piece of type containing that character would be cast. Special perforations indicated the amount of space to be positioned between words to justify the text. (For a more detailed description of the operation of these machines, see LEGR16, SEYB84b.)

The Monotype machines used typefaces that were designed according to a unit system; each character's width was a multiple of a fixed unit size. Unit sizes were specified in units per em. Common widths were 18-to-the-em and later 36- and 54-to-the-em (Figure 16). The narrow characters were 6 units wide,

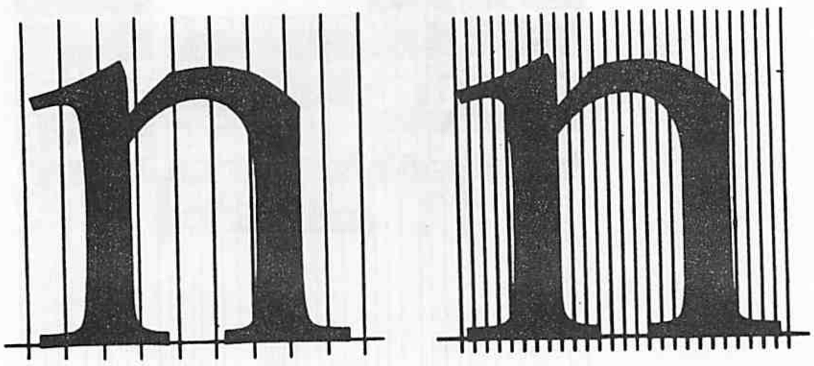
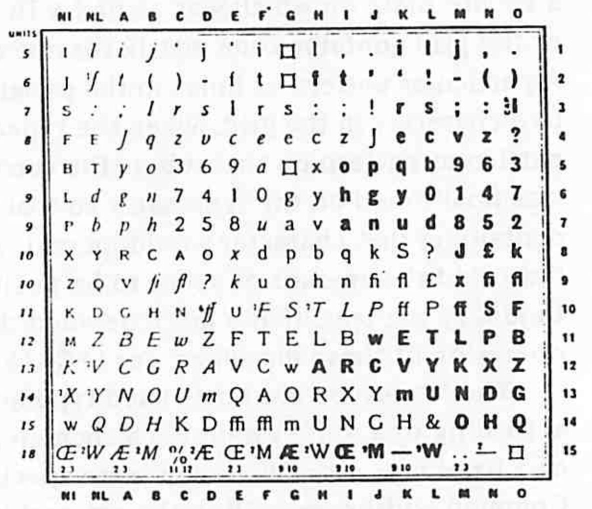


Figure 15. Letters drawn to an 18-unit grid (left) and 54-unit grid (right).

while the widest letters were 18 units wide. The Monotype system required that a font be divided into several different width categories each containing a multiple of sixteen characters to correspond to the sixteen rows in the font matrice. Each of the characters in a row had to have the same width, although one width could be used in more than one row. Commonly there was one row of the narrowest width containing characters such as i, l, 1, and punctuation, and one row of the widest width containing ligatures such as ﬁ and ﬂ, with the intermediate rows containing characters fitted to a unit size somewhere in between (Figure 17).



UNITS	NI	NL	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
5	ı	*l*	*i*	*j*	'	j	'	i	□	l	,	.	**i**	**l**	**j**	**,**	**.**	1
6	J	*¹/*	*t*	*(*	*)*	-	f	t	□	**f**	**t**	**'**	**'**	**!**	**-**	**(**	**)**	2
7	S	*:*	*;*	*l*	*r*	*s*	l	r	s	:	;	!	**r**	**s**	**;**	**:**	**:l**	3
8	F	E	*J*	*q*	*z*	*v*	*c*	*e*	e	c	z	J	**e**	**c**	**v**	**z**	**?**	4
9	B	T	*y*	*o*	3	6	9	*a*	□	x	**o**	**p**	**q**	**b**	**9**	**6**	**3**	5
9	L	*d*	*g*	*n*	7	4	1	0	g	y	**h**	**g**	**y**	**0**	**1**	**4**	**7**	6
9	P	*b*	*p*	*h*	2	5	8	*u*	a	v	**a**	**n**	**u**	**d**	**8**	**5**	**2**	7
10	X	Y	R	C	A	O	*x*	d	p	b	q	k	S	?	**J**	**£**	**k**	8
10	Z	Q	V	*ﬁ*	*ﬂ*	*?*	*k*	u	o	h	n	ﬁ	ﬂ	£	**x**	**ﬁ**	**ﬂ**	9
11	K	D	G	H	N	*¹ﬀ*	U	*¹F*	*S*	*¹T*	*L*	*P*	ﬀ	P	**ﬀ**	**S**	**F**	10
12	M	*Z*	*B*	*E*	*w*	Z	F	T	E	L	B	**w**	**E**	**T**	**L**	**P**	**B**	11
13	*K*	*¹V*	*C*	*G*	*R*	*A*	V	C	w	**A**	**R**	**C**	**V**	**Y**	**K**	**X**	**Z**	12
14	*¹X*	*¹Y*	*N*	*O*	*U*	*m*	Q	A	O	R	X	Y	**m**	**U**	**N**	**D**	**G**	13
15	w	*Q*	*D*	*H*	K	D	ﬃ	ﬄ	m	U	N	G	H	&	**O**	**H**	**Q**	14
18	*Œ*	*²W*	*Æ*	*¹M*	%	*²Æ*	Œ	¹M	**Æ**	²W	**Œ**	**¹M**	—	**¹W**	..	²—	□	15
	2 3		2 3		11 12		2 3		9 10		9 10		9 10		2 3			
	NI	NL	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	

Figure 16. Monotype matrix layout. All characters in a horizontal row are the same width.

Within a few years, both Monotype and Linotype were well established and, as a result, the early twentieth century saw the emergence of typeface alterations on a grand scale. Many popular typefaces were redesigned so that their dimensions met the restrictions imposed by the new technology. New typefaces were designed that could be used with either system or both. In designing fonts for the Monotype matrix, type designers had to know where the characters were to be placed on the matrix before they began the job of designing a typeface. Because of the restrictions of the matrix, many lowercase characters were widened while others were narrowed, some to an extreme degree. Although the Linotype machine did not require that fonts be sized according to a unit system, many of the fonts designed for use with the machine corresponded to a unit system so that they could also be used with Monotype machines.

Some typefaces were designed to be used with several different machines. One such type was Sabon, designed by Jan Tschichold in 1960, which was designed to be used not only by linecasting (Linotype) and single-type (Monotype) machines, but also for hand-composition in foundry types [DREY68]. One design was to be used in all methods. To further restrict the problem, the type was to be modeled on a sixteenth century typeface but 'modified to suit contemporary taste and needs.' Because of restrictions imposed by the new technology, the roman and italic designs had to match in width (they were both positioned on the same matrix for the Linotype machine), and the design had to be adapted to the smooth machine-made paper and the electric-powered printing mechanism of the present day. The working drawings were made for use with a modern pantographical punchcutting machine in a size twenty times as large as the first trial fonts.

With the development of high-speed printing presses used for the production of newspapers, the technology changed again. Newspaper type had to be legible in small sizes, sturdy enough to stand up to the high pressure of the electrified presses, and thick enough to print legibly on low-quality newsprint. With these considerations in mind, typefaces were designed that had thicker stems and heavier serifs, had a larger x-height for legibility, and were narrower so that more letters could be fit into one column of text, thus increasing the amount of information that could be packed into an article. The Times of London, in the early 1930s, commissioned Stanley Morrison to design a typeface specifically to meet the criteria imposed by the presses they were using at the time. His resulting design, Times New Roman, remains one of the most used typefaces today, for both newspaper printing and other typesetting such as books and advertisements. The Times later changed their press machinery and with it the typeface used to print the paper.

Many newspapers continue to use typefaces designed for earlier technologies despite the fact that their equipment and paper stock have changed a great deal since the typefaces were designed. As a result, these newspapers are not as legible as they could be. A mitigating factor may be that there simply are no other typefaces that stand up to the printing equipment as well the ones designed for it many years ago. Because of the wide variety of printing machines used today, no one typeface would be suitable for use in a majority of printing houses. Perhaps it is for this reason that no one at this time has commissioned a study similar to the one Morrison undertook to produce a typeface suited to the newspaper technology used today. (For a detailed study of newspaper typefaces, see GÜRT85.)

Phototypesetting

The application of photography to typesetting was originally proposed by William Friese-Greene in 1898 in a patent entitled 'Means for Composing Characters by producing Photographic Negatives therefrom' [MORI59]. Friese-Greene, however, decided to devote his time to 'Kinematography' or motion-pictures and thus abandoned his efforts at phototypesetting. The next patent was taken out by Arthur Dutton in England in 1919, followed by one in 1923 by Albert Bawtree. Because the integration of photography with printing was neither expedient nor inexpensive at this time, neither of these devices was commercially successful.

The next generation of phototypesetting machines was produced in the late 1940s and first exhibited in 1950. These machines, considered to be 'first-generation' phototypesetters since they were the first machines to be commercially developed, generated type images on photographic film or paper but otherwise did not differ in their mechanics from metal casting machines and thus were still constrained by the same restrictions that bound their earlier counterparts. The pot of molten metal was replaced by a strip of photosensitive paper and the matrice was replaced by a film negative. The mechanism for setting type and for designing and using it remained the same. It was at this time that the term *hot-type* began to be used to distinguish the older typesetting machines which still relied on heated metal to cast type from the new machines whose typesetting process was completely photographic (and thus *cold-type*).

Because the technology remained pretty much the same, many types were designed for use with both hot-type machines and phototypesetters. Some of the original phototypesetters could provide enlargements or reductions of fonts from an original master through the use of magnifying lenses and would also produce italic, back-slanted, shaded or condensed characters through photographic distortion. The two major hot-type machines, the Linotype and the Monotype, were both adapted to phototype becoming, respectively, the Fotomatic and the Monophoto. Each of these typesetters was constrained by the same conditions imposed on its predecessor: the Fotomatic used a matrice containing two equal width characters on it; and the Monophoto used a matrice containing characters that fit within a unit-spacing system. Both machines were capable of magnifying or shrinking the size of the characters that were typeset from one original pattern. The Monophoto machine could set type from 6 to 24 point and the Fotomatic could set type in certain sizes from 6 to 36 point.

With the advent of phototypesetting, type was no longer cut in a myriad of sizes by a skilled punchcutter or punchcutting machine; instead it was drawn in about four sizes and photographically scaled to the needed size. In fonts that were magnified or reduced, the density of the resulting image was sometimes altered, thus resulting in a character whose components were too light or dark, irrespective of the design of the original (in effect, the characters were under- or over-exposed). To compensate for this, the typesetters could adjust the amount of light by increasing or decreasing the number of times they flashed the light beam when setting these characters. This method of sizing characters lead to a degradation in the quality of typesetting made from phototypeset material especially in the sizes that required the most enlargement or reduction from the original. Other changes in phototypesetting fonts were the result of the photographic technology which burned away the edges of the type, thus producting letterforms that were thinner than their earlier counterparts. To compensate for this, the weights of phototypes had to be made heavier than letterpress types to achieve a comparable image quality [SEYB84b].

Second-generation phototypesetting machines stored a selection of *masters* on a medium such as a revolving disk (the Photon), a grid (the Linofilm), or film strips. The characters were still designed according to a unit-system, but in many cases, the units were smaller. This reduction in unit-size provided for a wider range of widths in the characters. The requirements of type designed for these machines varied considerably. A few of the machines required the use of *unit-count* or *unit-cut* fonts. The font widths were wired into the logic of the typesetter. The width of a particular character was the same for all designs of a specific size so that an 'a' was always four units wide, regardless of the typeface. This required that all fonts to be used on the machine be designed to conform to the widths stored in the machine.

Third-generation phototypesetters are characterized by machines that do not use photographic masters, but instead reproduce the characters electronically by scanning them across the face of a cathode ray tube (CRT). The early models of CRT typesetters relied on photographic masters of characters stored on grids, which were then scanned and painted on the face of the CRT screen (Figure 18). Later models incorporated the use of stored digital data that was converted on the fly into a character on the CRT screen. The machines utilizing digitized data were capable of generating cleaner images since the results did not depend on a technique that scanned a photographic master—a process that was *noisy* since it tended to add nicks and warts to the output image when any irregularities were encountered along the edges of the original image.

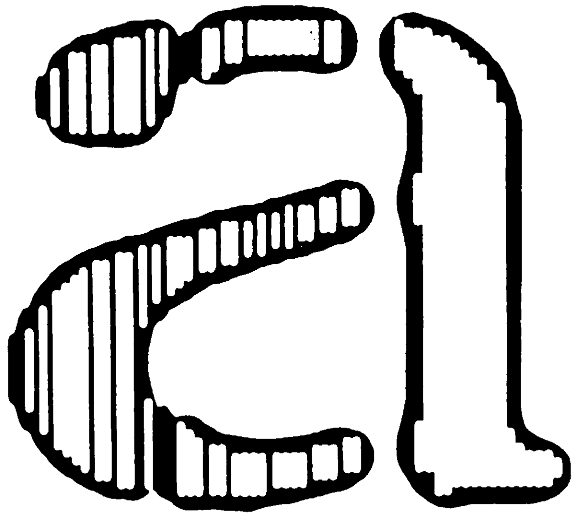


Figure 17. Character scanned on the face of a CRT.

For the first time, character images were reduced to a number of small elements which were then put together to produce characters [HOLL67]. This resulted in designers using a new design technique in which these small elements were manipulated to form type designs. This also led to the development of computer systems for automating the process.

The first typefaces designed specifically for a CRT typesetting machine were produced between 1976–78. These were the designs Marconi and Edison, by Hermann Zapf, Demos and Praxis, by Gerard Unger, and Video by Matthew Carter (Figure 19). Zapf and Unger designed their types for a typesetter manufactured by Hell-Digiset, which originally had a resolution of 600 lines per inch, although later designs were able to produce a much higher resolution. In order to maintain a high quality in the type design, each of the letters was designed by hand on a raster grid. One design was made for each letter; in order to produce different sizes of the design, this design was either reduced or enlarged by changing the size of the grid-square that was drawn. If the design was enlarged, the resolution was reduced, since the machine was drawing larger grid-squares; correspondingly, if the design was reduced, the resolution increased, because each of the grid-squares got smaller. If the original em-square was 100 x 100, (at 600 lines per inch, that is equivalent to a 12 point font), a 6 point character produced from the grid would actually be printed at a resolution of 1200 lines per inch resulting in less *jaggies* or jagged edges, and a 24 point font would be printed at a resolution of 300 lines per inch resulting in more jaggies, since each of the characters had exactly the same number of pixels in it. In the design of fonts for the current technology, this characteristic is reversed: the pixel shapes are more jagged at lower sizes, and less jagged at larger sizes.

ABCDEFGHIJKLMNOPQRSTUV
WXYZ 1234567890
abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUV
WXYZ 1234567890
abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUV
WXYZ 1234567890
abcdefghijklmnopqrstuvwxyz

Figure 18. Typefaces designed to be used with a CRT typesetter: Video by Matthew Carter (top), Edison by Herman Zapf (middle), and Demos by Gerard Unger (bottom).

To design the Demos typeface, Unger studied the characteristics of the CRT machine carefully so that he could take advantage of the shapes that it could render more successfully [UNGE79]. The design of the serifs, the angle of the joins, and the thickness of the strokes were all based on careful study of the curves that the machine produced. Unger chose those that most closely reproduced the curves that he had drawn in the original design. Because the CRT beam tended to round the edges of the serifs and fill in the junctions of the stems, he adapted the design so that it had rounded serifs and joins that were not very deep (Figure 20). He also studied the gradation of the edges of the characters to make sure that the curves generated by the CRT matched the curves drawn in his original design. He was not trying to match the curves exactly, but he wanted to make sure that the digital curve produced a pleasing character shape (Figure 21).

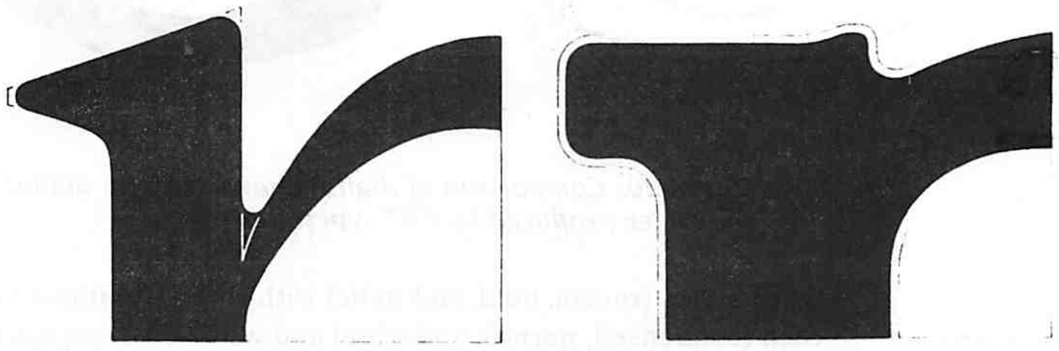


Figure 19. Etching away of edges and filling of notches as a result of the phototypesetting process (left); design of Demos' corners and notches are rounded to prevent photographic distortion (right).

Other designs made for phototypesetting machines took into account other factors of the reproduction process [SCHU70, GÜRT77a, MORI32]. Because type designs were now photographically enlarged or reduced to the desired size, the subtle adjustments that were made to a design so that all sizes appeared to be the same were no longer part of the type production process. To compensate in part for this omission, Bram deDoes designed his typeface Trinité in

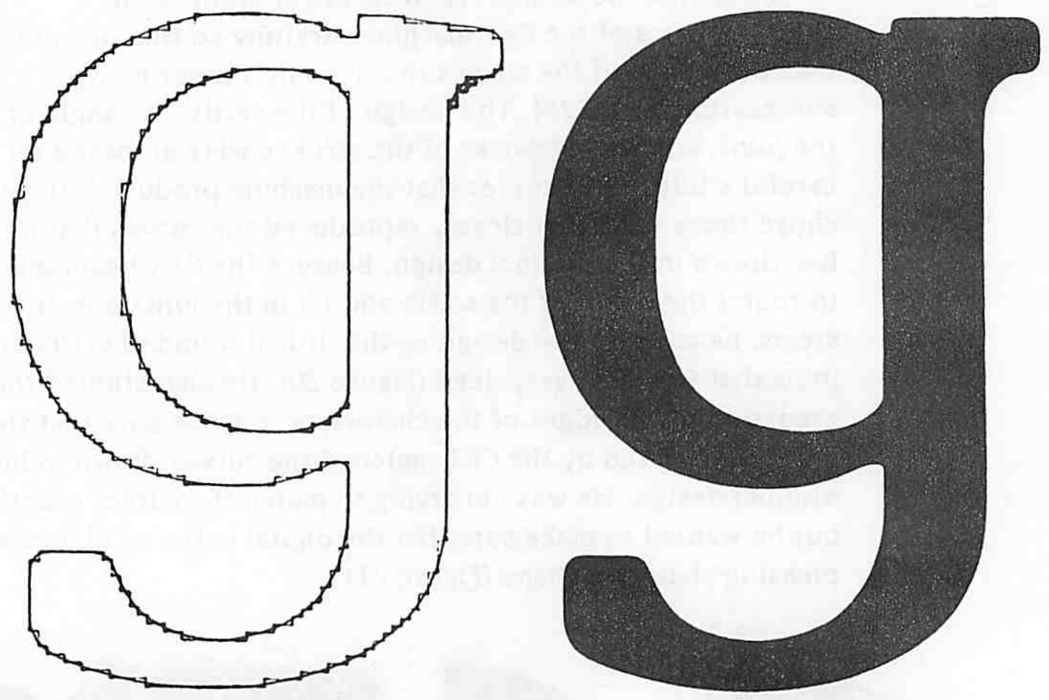


Figure 20. Comparison of digital character with actual character produced by CRT typesetter.

three styles (roman, bold, and italic) with three variations of each (condensed, normal, and wide) and with three lengths of ascenders and descenders (referred to as 1, 2, and 3, for the short, medium, and long extruders, respectively). In chosing a type size and style for the smaller sizes, a typesetter could select a wider, shorter, and bolder version which corresponded more closely to the appropriate design for the type at that size [DEDO].

While each of these technological developments has had an effect on the type design process, none of them has resulted in the development of letterforms that are radically different from those used with an earlier process. With each new printing device, though, the letterforms have changed to fit the available technology. Nowhere is this difference more apparent than in the changes that letterforms are now undergoing in their adaptation to a digital medium, first introduced in the early CRT typesetters.

Chapter III

Computer-Aided Type Design Systems

Available programs for interactive graphical input have neither surpassed the power of early systems, nor provided the subtle and diverse control of pencil and paper. The explicit action-response pattern of the computer dialogue is sufficiently cumbersome and distracting that the designer usually sketches his design away from the machine, using pencil and paper as his medium of externalization. The designer will bring his sketch to the computer for evaluation or transformation only when the design reaches a stage that warrants the overhead of digitizing it.

C. Herot, *Sketch Recognition for Computer Aided Design*

With the development of high-speed, computer controlled processors and high resolution graphics devices such as display screens and laser printers, new methods of encoding information have been formulated to take advantage of the new technology. These technological advancements have contributed to the increased use of digital information in place of traditionally analog forms, an example of which is the use of digital typefaces for the display of textual information.

Typefaces for digital output devices are created by converting a black and white image into a matrix of black and white *pixels* (from *picture elements*). The matrix is called a *raster* or *bitmap* and can be stored as binary information. A typeface or letterform that has been stored as digital information is referred to as digital type and can be used by phototypesetters, CRT displays, ink-jet printers, laser typesetters, and other output devices capable of utilizing digitized images. The resolution of the output device varies considerably and is determined by the number of pixels per inch that the device is capable of rendering. Available resolutions range from about 60 to 2400 dots per inch. There is a direct relationship between the resolution of the output device and the legibility of output produced by the device, with the legibility factor increasing as the resolution increases.

Converting letters to digital shapes

Conversion of a letterform design into digital information is usually done in one of two ways. Neither method is completely satisfactory, and both require that the design of the typeface be completed before the characters are converted into digital images.

Scan conversion

The first conversion technique is that of scanning a character with a high resolution optical scanning device. Characters can be scanned as either outlines or as solid shapes. The process consists of several steps. First, the scanner passes a beam of light repeatedly over the character image, with each pass covering only a small band of the image. The dark parts of the image are converted into dark pixels, the light parts into white ones. The scanning width of the beam can be very small, sometimes as fine a few microns in width, thus producing a very high resolution raster image. The process is limited, though, by the resolution of the scanning beam and the precision with which the characters have been drawn. Often, small imperfections in the shape result from scratches, dirt, dust and blurred edges on the character drawing, all of which contribute noise to the scanner. These imperfections result in dropouts and pickups that are small indentations or bumps on the interior and exterior edges of the digital character and must be removed before the characters can be used as digital data (Figure 21). An additional problem is a staircase effect along the edge of a letterform, a phenomenon also known as the *jaggies*. This problem occurs on any edge that is not aligned exactly vertically or horizontally, such as curves or diagonals, and also affects lines that should be parallel or perpendicular to the baseline but are not lined up exactly with the direction of the scanner beam.

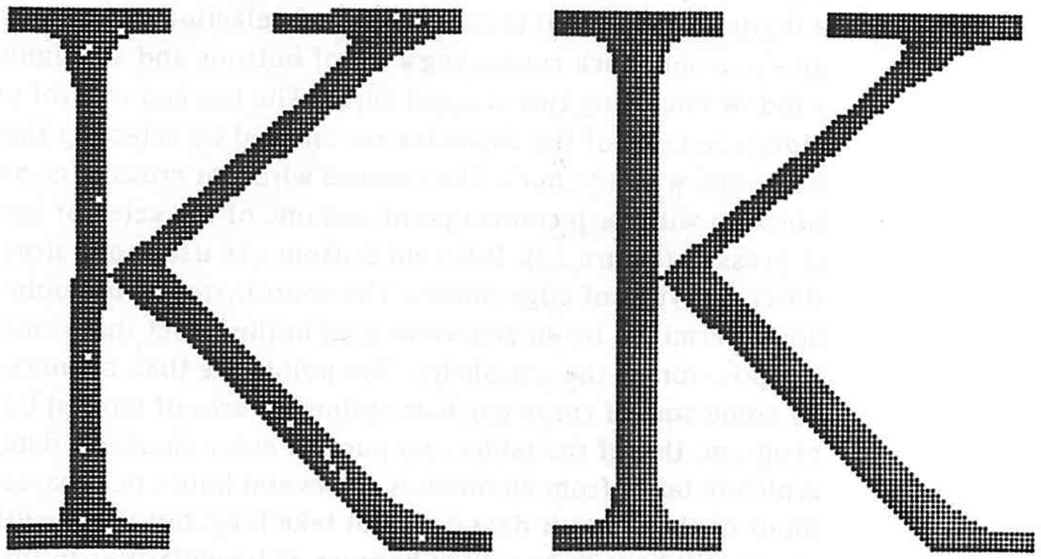


Figure 21. Scanned character (left) and fixed up character (right).

In the next step of the scanning process, the matrix image of the character must be hand edited to remove any digitization errors. To facilitate this process, the image is printed in both a large and small size. The large size is used to determine which pixels need to be corrected (either turned on or off) and the small image is used to see what the character will look like at its true size. This step is crucial and is done by trained letter-drawing specialists on either a graphics display screen or on a printed copy of the raster image. Any corrections to the digital form are then entered into the system.

Outline contours

The second means of digitizing typefaces for computer output relies on adaptations of techniques formerly used in making punches or matrices with the pantograph machine. A character whose edges have been drawn with a very fine line in a large size is first marked with control points along its interior and exterior edges. These control points usually indicate the boundaries of straight edges on the character, curve transition points, inflection points, and corners. The marked character is lāid on a digitizer tablet that is connected to a selection device such as an electronic puck containing a set of buttons and an alignment window enclosing two crossed wires. The marked control points along the edge of the character are entered by selecting their positions with the puck: the crossed wires, or crosshairs, are lined up with each control point and one of the selector buttons is pressed (Figure 22). Different buttons are used to indicate different types of edge points. The coordinates of the point are determined by an electronic grid in the tablet that senses the position of the crosshairs. The points are then connected by some sort of curve (such as splines or arcs of circles) by the program. Use of the tablet and puck to enter character data typically takes from 20 minutes to several hours per character. Input of the contour data does not take long, but the resulting forms still have to be edited because of irregularities in the bitmaps when the contour data is converted to digital rasters at different sizes and resolutions.

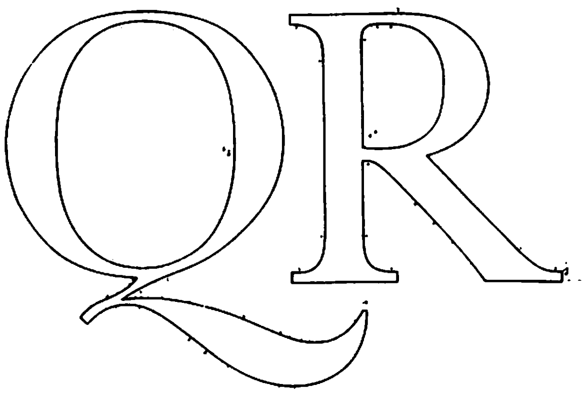


Figure 22. Outline characters with control points marked. (Artwork by Kris Holmes).

An alternative method of entering points uses either a program or a fixed set of data to describe or list the coordinates of the control points. Additional instructions are given specifying how points are to be connected through the use of straight lines and curves.

Once an outline has been entered, it is used to produce bitmaps of the characters, possibly in several different masters or sizes (Figure 23). Scaling the outline, though, may result in distortions to characteristics of the form, such as the serifs (the finishing strokes on a letter), the hairlines (the thin parts of the strokes), and the stem widths (the main vertical strokes in the letter). Some features on the resulting masters may be too light in the smaller sizes and too heavy in the larger ones. If the distortions are very bad, each of the new images must be hand-edited.

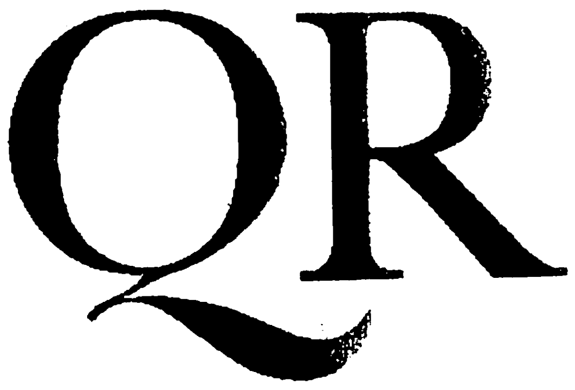


Figure 23. Bitmaps produced by the Ikarus system from the outlines in Figure 22.

Type designs must now be created to correspond not only to the specific resolutions of particular printers; they must also be tuned to the printing technology and its characteristics, whether it be liquid ink used with ink-jet printing or dry toner used with electrostatic devices. Each resolution and each different technology requires that typefaces be designed to take advantage of the particular characteristics of the machine the typeface will be displayed on. For particularly low resolution devices, the type designs may need to be hand tuned using a bit-editor program, one that allows the designer to turn on or off individual pixels in the raster.

Outline designs provide the most flexibility if the letterforms are to be altered in various ways. Designs can be stretched, rotated, slanted, or emboldened by manipulating the control points. Interpolation can be done between two different designs, resulting in a range of designs that are part way between the two originals; for example, a medium-bold face can be created by interpolation between a normal weight text face and a bold face.

Bitmap editors

Some digital designs have been created by a designer who first draws the characters on a piece of graph paper and then creates digital fonts with the designs by using a *bit-editor* program. Typically, such a program provides a display containing a digital rendering of a character and allows the designer to turn individual pixels on and off by selecting them with a mouse or other pointing device. These bitmap editors are popular programs on various computer systems, but they do not provide a satisfactory solution to the problem of creating digital typefaces; they neither provide the fine tuning that is needed in font design, nor the means of efficiently producing large numbers of designs since all characters created in this manner must be created bit-by-tedious-bit.

Early design systems

Early efforts at computerized type design systems were not particularly successful in producing quality output. The systems had primitive forms of input and crude graphics. There was little attempt to involve a type designer in the creation of the systems, so they are characterized by the fact that they were, for the most part, written by and for computer programmers, not type designers. The input method was usually slow and tedious, and allowed little variation in the letters that were produced.

In 1967, two different typeface conversion systems were created [MATH67, HERS67]. Both systems used card data to input data containing coordinates for points along the edges of the characters. These points were then connected by straight line segments. The fuzziness of the display medium and the low resolution of the output devices available at that time tended to round out the straight lines making up the letter to give a smooth edge (Figure 24). Although the input method was slow, these systems did produce higher quality output than was available on other systems current at the time and they also provided an easy means of creating new shapes that could be saved for later use.

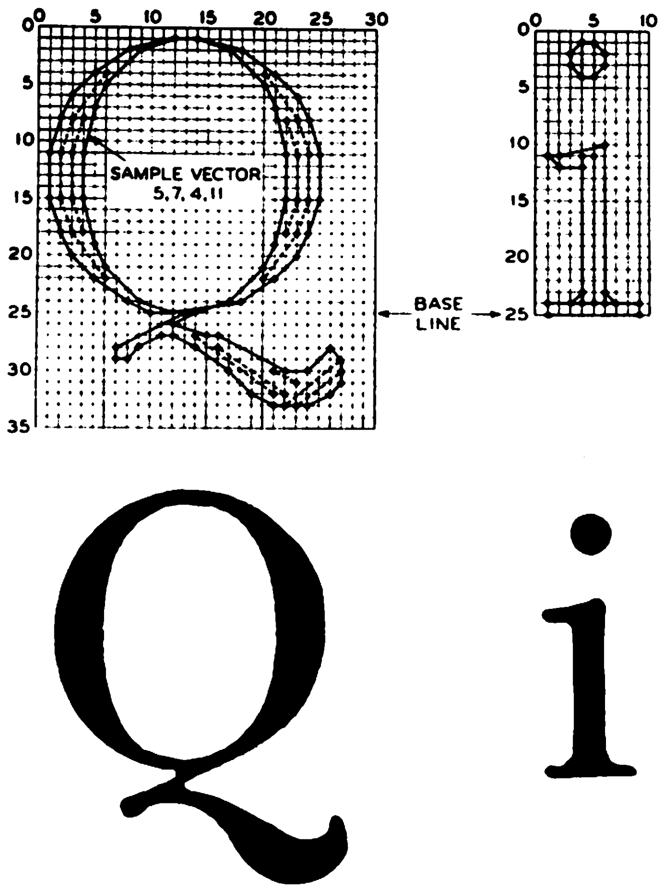


Figure 24. Letters drawn with vectors on a CRT screen from [MATH67].

Another system created about the same time used a different approach. ITSLF, or InTeractive Synthesizer of LetterForms, was designed by H. W. Mergler and P. M. Vargo [MERG68]. The authors isolated certain characteristics in a particular typeface such as the height, width, stroke weights, and serif shapes and thought that they could use these characteristics to produce different typeface designs. The characteristics could be generalized to a set of parameters so that modification of the parameters was the only step necessary to produce different designs. The authors felt that if they limited the designer's role to that of only specifying certain parameters instead of the entire design of each letter, this would lead to consistency in the overall design of the characters. Although it sounded like a good idea, it really didn't produce very attractive letters (Figure 25).

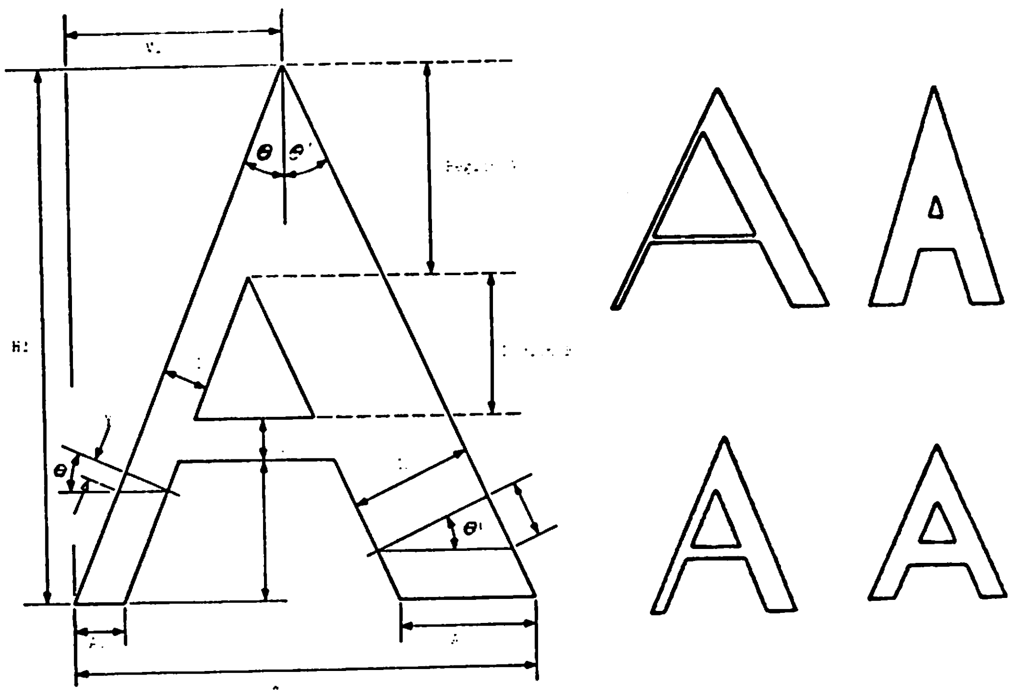


Figure 25. ITSLF character with its parameters and some of the variations produced by varying the parameters.

Work done in 1975 by Phillippe Coueignoux for his PhD dissertation also used the idea of characterizing an alphabet by generating descriptions of the characters and then modifying those descriptions by the use of parameters [COUE75]. He developed a system called Character Simulated Design (CSD), that was a program for the generation of high quality digital characters. He studied the structure of the characters and tried to quantify or extract the consistency in type fonts both between one letter in different fonts and between all letters in one font. He derived primitives such as stems, arms, and noses and defined the spatial relationships between them (Figure 26). From careful evaluation of these primitives, he generated a grammar to describe the implicit structure of the characters in a font. This grammar included rules for specifying the relationship between the parameters, and rules for describing how the primitives were combined within a font. To output a letter, a routine containing a description of the letter was given parameter values for the primitives in the letter.

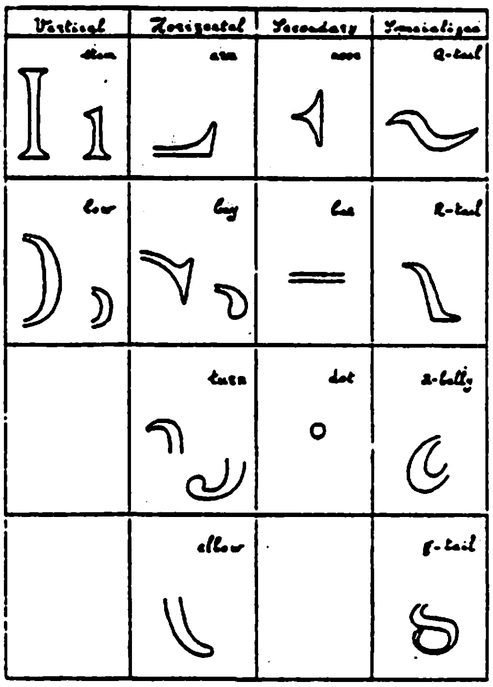


Figure 26. Character parts used by Coueignoux.

In 1978, David Kindersley and Neil Wiseman at the University of Cambridge developed a letter design system named ELF [KIND79, WISE78b]. It consisted of an interactive display with a keyboard and lightpen and was used for the creation, manipulation, measurement, or copying of images. The system allowed a user to sketch a design on the display surface using the lightpen, modify the image, and then convert the finished drawings into raster fonts. Output from the system was on a laser display plotter.

Another system that appeared in 1978 was the PM Digital Spiral developed by Peter Purdy and Ronald McIntosh [PURD78]. Their approach appears to be a modernization of a process proposed by Legros and Grant to draw smooth outlines for a pantograph machine [LEGR16]. The PM system read in scanned character patterns and displayed them on a high-resolution screen. A designer traced around the digitized letters by manipulating a computer-stored logarithmic spiral that was displayed on the screen. The spiral was used as a guide that was fitted to the contours of the letter. When the spiral was positioned so that part of its edge matched the desired contour of the character, the designer specified the start and end points of the match. These coordinates were then saved. Irregularities such as dropouts and pickups inherent in the digitizing process could thereby be removed. The letters were thus converted into a list of curve sections, or links, corresponding to positions on the spiral (Figure 27). After the edges of the letter had been smoothed, the coordinates of the outline were stored and later used to recreate the image. The combination of the spiral with the high resolution screen gave a very sharp edge.

Contemporary design systems

One of the most sophisticated design systems created specifically for letterforms is the Ikarus System, developed by Peter Karow at the company Rubow Weber in 1973–74 [ELSN80, ELSN81] and updated extensively since then. Ikarus is one of the few typeface conversion systems to be marketed as a commercial product. It provides a method of converting data with smooth contours, such as letterforms, into digital data

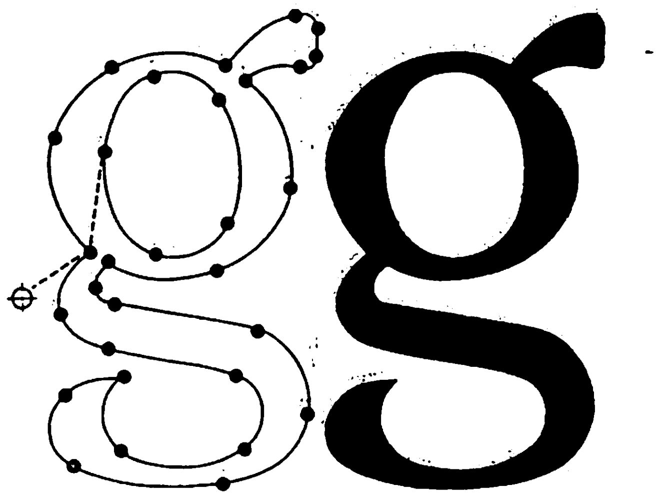


Figure 27. Character produced with the PM Digital Spiral.

through the use of a digitizing tablet and an associated selection device. Characters are entered into the system using the outline contour method described earlier.

Ikarus has quite sophisticated routines for adjustment of character shapes. It allows groups of characters to be controlled by universal parameters such as height, width, horizontal and vertical alignment, or slant. Characters can be stretched or shrunk in either a vertical or horizontal direction. Editing of characters can be done either to the character displayed on a video screen via cursor or keyboard entries, or to a listing of the data representing a character. This listing is in a format that is readable by a designer. Special functions provide interpolation between weights and rounded or shaded versions of a design (Figure 28).

Metafont is another sophisticated design system that originally appeared in 1979 [KNUT79]. It was subsequently extensively updated by its author, Donald Knuth, which resulted in an entirely new version that appeared in 1985 [KNUT86]. Metafont is a programming language in which character shapes are specified through x and y coordinates of control points and information pertaining to how the points are related and

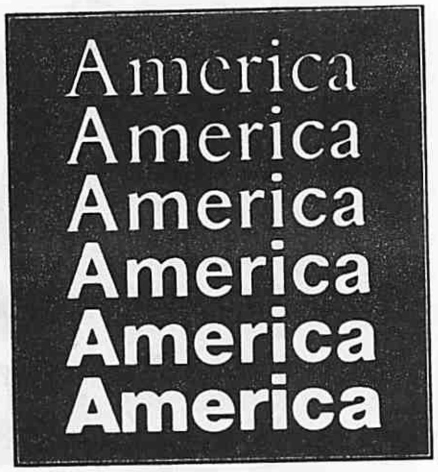


Figure 28. Interpolation between light serif face and bold sans serif face done by the Ikarus system.

connected. The letterforms are described using a mathematical notation by specifying either a pen nib and the path of the nib, or points along the edge of the character outline that can be filled in. Special curve algorithms were developed to draw 'pleasing shapes' using Bezier curves whose control points lie along the edges of the character. Additional routines provide a filling algorithm that has been specially tuned for the design of typefaces. The designer can incorporate parameters within the description to provide global control over the character shapes. Different settings of the parameters can produce variations of the characters for different resolutions, weights, slant, proportions, or orientation (Figure 29). Along with the ability to specify parameters, the designer can also specify subroutines for character parts and then use these parts within any character.

Metafont is a very powerful tool but it does have some drawbacks. To obtain a high quality reproduction of an existing face, a Metafont program must be written for each character in the face, a process that can presently take several hours or days per character. This conversion does not include the specification of parameters that would allow the design to be changed to fit different sizes or resolutions. Parameter specification is not built into the system; it is different for each typeface and thus

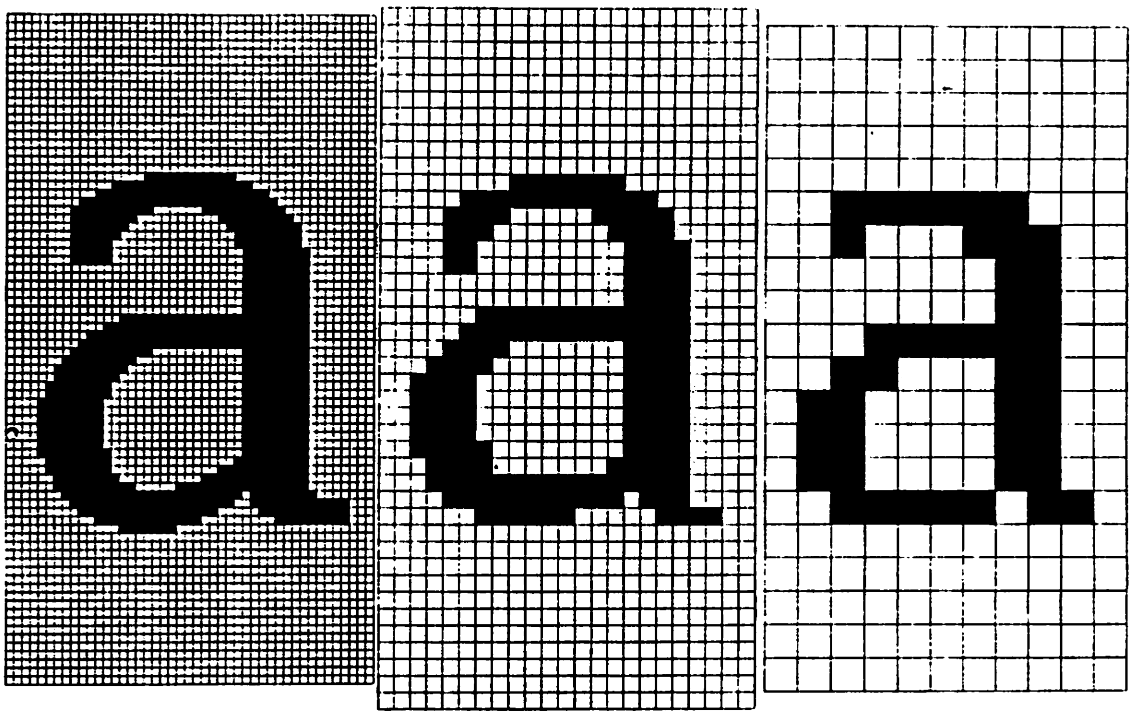


Figure 29. Letter drawn using Metafont and adjusted through the use of parameters for different point sizes: cap height 5mm (left), cap height 2.5mm (middle), cap height 1.25mm (right) (design and programming by Richard Southall).

must be specially designed with the typeface. The concept of parameterization is not a familiar one to designers and failure to understand how to use it may hinder their full use of the system. Another drawback is that interaction with the system can be slow because the system is declarative rather than interactive, requiring the designer to first create the Metafont program containing a mathematical specification of the letterforms, and then to run the Metafont system with the letterform program. Modifications to a character are made to the program, and then the system is rerun.

The Letter Input Processor, or LetterIP, was developed by the Camex Corporation [RICH82, FLOW84]. A modification of a system developed for the composition of display ads for newspapers and magazines, LetterIP is used to convert outline drawings of typefaces into a digital format for use by phototypesetters. It consists of three components: a character digitizer, an interactive character editing system, and a batch-processed data base for storage of the letterform information. It has a high resolution vector display connected to a digitizer tablet with a menu and a puck. The process of entering and editing a character is done at the display terminal by a trained designer. A puck is used to enter control points along the edge of the character or to select one of the editing commands listed on a large menu to the left of the digitizer tablet. As each point is entered, it is displayed simultaneously on the screen. When all the points have been entered, an outline of the character appears on the screen.

Once the letter has been entered, it can be edited interactively. Several features are provided with LetterIP to aid in the editing process. A grid can be displayed along with each character to aid in the adjustment of the heights or widths of the images. A library is provided for the storage of character parts that can then be called up and used either as is, or the parts can be rotated, scaled or reflected before use. Functions to zoom and pan an image from half to double size are also available. Additionally, a gauge or ruler is provided for the measurement of stem widths, distance between points, or the angle between the ruler and the vertical axis, thus ensuring the accurate measurement of proportions of different characters. LetterIP can also display up to three characters on a screen for the designer to check on the appearance and spacing of the letterforms.

Experimental work on type design systems includes work done in Switzerland by Eliyezer Kohen on a Lilith machine [KOHE85], and work by Kathy Carter on Imp, a system for computer-aided type design [CART85a]. Both systems were written by a member of the research group that helped develop the workstation on which the font system was implemented. This was a contributing factor in the design of the font systems since they were each optimized to take advantage of the hardware and software facilities on a particular workstation.

Imp was designed to run on the Rainbow workstation developed at the University of Cambridge [WILK84]. It is a system that allows a designer to sketch using a mouse or a stylus and a graphics tablet. The system tries to organize the complexity of an interactive system by providing a hierarchy of windows corresponding to the organization of information by a designer. This implementation was specifically designed to work with the Rainbow Workstation, a color raster display employing special hardware to support fast window manipulations. With Imp, a designer is able to sketch shapes on the monitor screen by drawing with the stylus on the tablet, then to interactively specify an outline shape by positioning and connecting points along the edges of the character sketch. Font designs are stored as outlines or rasters and can be edited in either form. Although the graphics of the system are very impressive, the system is somewhat limited in its handling of character data: no curve drawing routines are provided and all curved edges are rendered with straight line segments; there is no way to align an outline image with a raster to insure that the character parts digitize evenly. An advantage that it has over other type design systems is that it uses the spacing algorithm developed by Kindersley [KIND76] so that spacing of characters can be done automatically without any specification of sidebearings by the designer.

The Lilith font editor is geared for generation of fonts in the middle resolution range, or between 100 and 500 dots per inch. The system works in four stages: first, the outline of a character is input through the use of contour data that is then digitized; next, the curves of the characters are contour edited, after which rasters can be generated in several sizes; and finally the bitmaps can be fine tuned (Figure 30). Guidelines that are used to determine the character pattern include ascender and descender lines, the middle line (the x-height), and the right and left line (sidebearings). Implementation of the font editor was done in Modula-2 and took advantage of the Lilith window software and an extensive library of subroutines for the display and manipulation of data. The system has been optimized to provide a a rapid redisplay of the screen image when objects

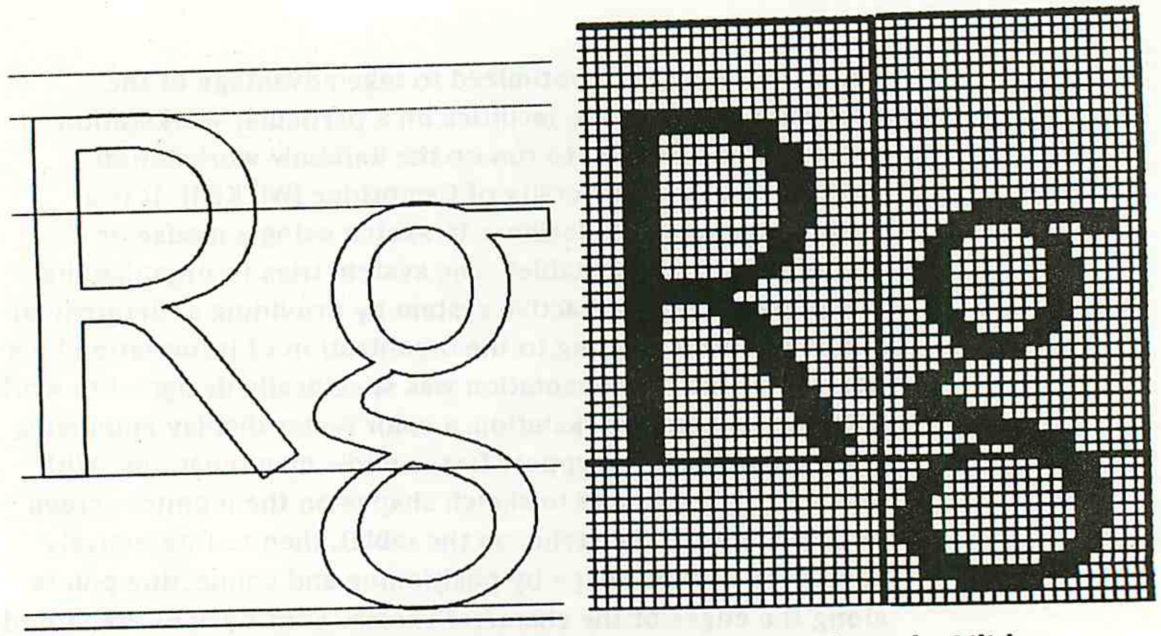


Figure 30. Contour and raster character from the Lilith font editor. (Illustrations courtesy of Rachel Hewson).

are dragged across the screen and to fill objects with closed contours.

Evaluation of current techniques

Each of the systems surveyed can be classified according to several criteria. Here we review these categories and summarize their assets and their drawbacks:

- *Input of data* – All these systems generally fell into one of two categories with respect to how the letterform images were entered into the system. The first group used an encoding of a specific pattern of pixels that were then manipulated in one of two ways: either the digital image was edited using features of the system, or it was used to derive a descriptive image after which the digital image was discarded and the description was saved. The second category of systems started with a description of a character shape from which the raster pattern or bitmap was be derived. This description was entered either through descriptive commands to the system or through specification of coordinates of points along the edges of characters. The description could be modified to produce changes in the letterforms.

• *Emulation of pen and paper* – A design system may attempt to emulate the actions of a designer by providing tools or commands that are similar to the use of a pencil and paper. The motion of a pen on paper was a common metaphor of these systems, although the type of pen varied. Metafont, for example, used this approach by providing an analogy to pen-drawn letters. Movement of the pen was controlled by specifying the points through which the center of the pen passes, thus producing ductal forms that look like pen strokes. The pen angle and the shape of the nib could be varied to produce variations in the characters. Imp allowed a designer to sketch with a flat-nib pen at a user-specified width and angle, but final character shapes had to be specified through outline data. Other systems provided an outline model, where points were specified along the edges of a character and then connected to create outlines of the characters that were later filled in. Use of this approach allowed a designer to draw letters that could not be duplicated with only single strokes of a pen. Metafont allowed both the pen metaphor and the outline metaphor to be used.

• *Visual feedback* – Only a few of the early systems provided visual feedback, but all of the later ones did. Some of the systems, such as LetterIP, Imp, and the Lilith editor provided interactive manipulation of character features through use of a high-resolution display screen and a pointing device. Other systems required that manipulations to the data be done through editing of the data points.

• *Target users* – The users to which these systems were addressed and the environment in which they were used affected both the quality of the letterforms generated and the success of the system in an environment outside the original development testbed. Some of the systems, such as ITSLF and CSD, were only experimental and thus were never used by anyone other than the authors, or at any site other than the one where they were developed. Other systems, such as Ikarus, were marketed and licensed to companies that used them for the production of digital designs. All the systems, however, provided only input and editing capabilities; the systems were optimized for the copying of existing designs and required that new typefaces be drawn with pencil and paper. Input of data into these systems was not usually done by the designer but typically was done by persons trained to use the system and who might have only a rudimentary knowledge of letterforms. This lack of expertise may have lead to degradation of the design when it was digitized, as the subtleties of a design can be lost if they are not perceived or understood by the system operator.

• *System designers* – Systems for the design of letterforms fall into two categories: either they are experimental systems that have been designed by persons whose background and experience has been largely with computers, not with type design, or they are production systems, designed for the large scale production of typefaces. In either case, little has been done to create a system for the designer. With the first type of system, the author made little attempt to reconcile the designer's model with the world of computers and in the second type of system, the emphasis was on fast, efficient production of digital letters from pre-existing character shapes with no attempt to facilitate creation of new designs.

• *Curve handling* – The conversion of a curved analog image into a digital one was not an easy process. The designer, when drawing a letter shape, does not think about whether the curves are cubic splines, arcs of circles, or parts of a predefined curve such as the PM Spiral. Translation of a design into a description involved conversion of curves in the image into a representation of those curves so that they could be easily manipulated as the design was modified. The type of curve chosen had to satisfactorily reproduce the curves needed by the designer, but was not allowed to control the design process. Satisfactory results were obtained with each of these three types of curves, but no one of them has emerged as being obviously better than the others.

• *Design by parts* – Throughout the history of type design, there has been speculation (usually not by type designers) that typefaces are simply a collection of parts that have been arranged to produce a uniform set of letters. If a set of parts could be designed, and a set of rules developed to combine them, the process of type design could be simplified. A designer needed to design the various parts and the combinations would be taken care of automatically [GOSH83]. This idea is too simplistic, however, as there are subtle changes to the various pieces when they are combined to form character shapes. ITSLF and CSD developed within in a computer environment, attempted to employ this innovative approach to letterform design. Although they both provided an unusual consideration of the problem, they were not very successful in creating beautiful designs. There may be a viable solution to this problem, but these systems did not achieve it.

• *Use of parameters* – Many of the systems had the essential design of a typeface built into them and then allowed manipulation of the design through the control of parameters that controlled various features of the letter such as their size, weight, slant, or the use of serifs. This use of parameters within a design made it easy to create uniform shapes when typefaces were created since the designer was allowed to control the dimensions of the typeface independent of the letter designs themselves. These dimensions might include such measurements as the x-height or height of the lower case letters, the cap-height or height of the upper case letters, the height of the ascenders and depth of the descenders, the stem width, and the slant. If parameters were used to specify certain characteristics such as height or slant, then a change in any of the parameters was applied to all of the characters that were controlled by that characteristic.

Defining the parameters that controlled a design was a difficult task. A type designer does not think about parameters when creating a new typeface, so it is difficult to quantify or measure the exact relationship between one letter and another. ITSLF allowed the designer to specify the values of certain parameters for the letter E and then used those values to modify knowledge already stored in the system to create the remaining letters of the alphabet in a similar style. Metafont provided the most flexibility of any of the systems, allowing the designer to specify both the parts and the letters and the relationships between them. Changing one of the relationships (or parameters) had the effect of changing one of the characteristics of the typeface.

• *Copying of typefaces* – Most of the systems discussed so far can be classified as 'copying' systems since only two of them allow a designer to actually use the system for designing original typefaces (Metafont and Imp). These copying systems allow the input and modification of character shapes through use of an input device, but they do not provide an interface that allows a designer to duplicate the design process interactively. The provision of an interactive design environment seems to be a crucial step in making the transition from the older technology of the pantograph to the new technology of a computer workstation with a high resolution bitmap screen. The systems that do provide sufficient means for designing new typefaces have other drawbacks that have already been discussed.

We have reviewed here some of the past and current systems used for the generation of digital typeforms. It is clear that none of them fully satisfies the need for a design system that can be used by a type designer who is familiar with traditional design techniques, but who may not be familiar with new computer hardware. We describe in the next chapter what is needed in a system that takes into account the type designer's needs and the means he uses to create typefaces that are not only aesthetically pleasing but are also designed to be used with the new generation of digital printing systems.

Chapter IV

Adapting the Past to the Future

In reorganizing or readapting an art or a skill to new conditions and new ways of thought the most obvious source of inspiration is in the work of the masters of the past.
Nicolete Gray, *Lettering as Drawing*

Having set the stage by describing the traditional methods of type design and the changing technology associated with this field, we are ready to describe the development of a system for the creation of digital typefaces that attempts to bridge the gap between the two. The system we designed, which we have named Paragon, will allow a designer to develop new techniques for manipulation of typefaces within a computer environment. Within the system, we provide:

- a highly interactive, easy to use *type design system* modeled on the designer's traditional tools

- a *graphics interface*, allowing graphical communication and graphical manipulation of objects

- a variety of *basic tools* that have been tuned to the type designer's specifications or needs, not a general design system

- *enhancements to the basic tools* where we can take advantage of the computational and display capabilities of a computer workstation

- a means of creating *extensions to the basic tools* to accommodate new techniques, yet remaining within the paradigm presented

- a *simple world view* consistent with a designer's perceptions with hiding of features a designer should not be concerned with such as storage of library functions or typeface data

Research Contributions

The development of this system has provided contributions in three areas related to the integration of the font design process with research in the field of computer science. Our first contribution is an understanding of the visual process of designing a typeface and a translation of that understanding into a computer metaphor that mimics the process. Current systems for the generation of type designs are geared toward achieving a final product; they do not concern themselves with the beginning of the process, only the end. We chose to study the initial design stages and to model a system that allows a designer to create digital designs directly. In addition, we provide raster versions of the designs as the material with which a designer works from the beginning of the design process rather than restricting their use to the final stages of design, as they are in commercial software. We are concerned here with the design process rather than the design results. The metaphor that we have developed could, conceivably, be applied to other types of design problems (such as, perhaps, graphic design, mechanical design or architectural drawing) that are concerned with the translation of a visual idea into a written one.

To achieve this goal, we first had to decide how to automate a task that is very idiosyncratic, depends primarily on visual feedback and is not easily explained to someone not familiar with the problems associated with the creation of very small but very precisely drawn objects. We needed to analyze and define this process so that we could create a metaphor for a set of design tools that could then be used as the basis for a computer design system. This process of definition and analysis is generally known as *knowledge engineering*, our efforts were focused on standardizing the tools and actions of many designers into a single consistent model. Our analysis was predicated by the statement made by Herot over 10 years ago when describing future design trends, 'It is not sufficient that the system be merely *possible* to use, it must be sufficiently comfortable and easy to operate that the designer can integrate it into his design strategy. This new class of user will rightfully demand more amenities of the system, amenities that it must provide if it is to be an effective design tool.' [HERO76]. By working and talking with type designers, we developed a consistent and workable model that was used in the design and implementation of Paragon.

Our second contribution is a model of a design system that can be enhanced by non-technical users. Rather than designing and implementing a *static* design system whose functions are completely specified before the user begins working with it, our system is *dynamic* in that it is built upon primitive actions so that new functions can be defined interactively by the people who are using the system. This approach allows the user to develop tools that are more closely suited to those he is familiar with and thus allows each individual user to customize the system to suit his own standards. What makes this system different from existing extensible systems is that the customization process is entirely graphical; that is, all extensions are made through an interactive graphics interface.

To create this model, we had to determine how to provide the means by which the functions within the system could be augmented by the users of the system. Around the same time as Herot's work, Negroponte suggested that 'if a system is to be person oriented, that person should at least design it and should be able to change it at a moment's notice' [NEGR76]. Licklider concurred when he wrote, 'It will be necessary for the designer to view the prospective user as a design extender and the delivered system not only as an application package but as a kit of design-extension and incremental-development tools.' [LICK76]. Although we provided a single model for the design process, we also wanted to provide a means of customizing the system to cover individual models within the design metaphor. This provision allowed each designer to develop a working model that fit with his design paradigm and thus allowed him to create a system that was closer to his true working environment. The customization process is implemented via a Threaded Interpreted Language (TIL) built into the system and invoked entirely through graphical means.

This model differs from earlier systems such as Smalltalk [GOLD83], Emacs [STAL81], and Rehersal [GOUL84] in that it allows a novice user to extend the repertoire of the system without any knowledge of programming or programming language constructs. Because designers have an almost native distrust of computers and other mechanical devices, one of our objectives in designing this system was to provide the means for the system to grow without requiring the user to have any knowledge of programming metaphors. The customization process is a simple macro definition facility but we have implemented it in such a way that a designer sees it as an integrated part of the design system and can access and use it without any prior programming or computer experience.

The third contribution of our research is the development of new tools for designers that enhance and expedite the digital design process. These tools are adapted to existing computer technology and allow a designer to take advantage of the sophistication and fast processing available on the current generation of computer workstations to speed up the design process.

To solve this last problem, we needed to specify functions that would span the gap between the techniques used by designers which are associated with an older technology and ones more suited to the current technological developments. Rather than automating a process that was closely associated with a previous generation of printing machines, we wanted to provide a means for the transition between two technological generations, thus bringing type designers solidly into the twentieth century. The process of automating an existing (although not well understood) process that was already outdated was not seen to be either a step forward or a viable topic of research. We wanted to avoid the possibilility that we might find ourselves in the same situation as the White Queen and Alice, that is, running just as hard as we could to stay in the same place. By including functions that allow a designer to view and proof a design that is rendered in the same format as it will be printed on the current generation of laser printers, we would provide the means for designers to develop new techniques for the design of digital typefaces for raster output devices.

Current Printing Technology

There are a variety of output devices in use today for the display of digital letterforms including ink-jet printers, laser printers, phototypesetters, and dot-matrix impact printers. Each type of output device differs in how ink is actually deposited on the surface of the paper, or, in the case of phototypesetters, in how marks are made on film.

In the past, type designers had to take into account the printing technology and the paper quality when they designed a typeface. These considerations have not changed, but the technology has, so there are a new set of problems to be dealt with in creating and evaluating a typeface design. We no longer have to consider *wear* on the surface of the type, a factor that in the past led to the use of heavier serifs and sturdier hairlines that stood up to the high-pressure, high-volume presses used, for example, in the production of newspapers; instead we have to consider the type of internal marking engine used with today's printers.

A *laser printer* contains a laser beam that sweeps horizontally across a metal drum as the drum rotates. An electric charge is deposited by the beam on the surface of the drum so that when the drum rotates past a toner cartridge containing fine black carbon particles charged to the opposite polarity, the particles cling to the drum in the places where the two charges attract. A piece of paper is then passed between the drum and a heated cylinder that fuses the carbon particles onto the paper [AYAL84].

There are two basic types of laser printers available today. In a *writes-white* device, the laser beam de-charges the drum in the areas of the page that are to remain white. In a *writes-black* printer, the beam charges the areas of the page that are to become dark. Because the charge on the drum spreads a little due to the characteristics of charged particles, the writes-white device de-charges an area slightly larger than the actual white area on the page, thus reducing the dark areas a small amount. This produces lighter pages than a writes-black device, on which the charges in the dark areas spread out. The result is that character bitmaps printed on a writes-white device tend to be lighter than the same bitmaps printed on a device that writes black. Both types of devices leave a slightly fuzzy edge on the resulting letterforms.

An *ink-jet* printer shoots a continuous stream of small electrically charged drops of ink through a very fine nozzle at a piece of paper [KUHN79]. Because of their electrical charge, these drops can be deflected to land in the proper position on the page, or to drop into a tray positioned to collect surplus ink. Because a drop of ink, no matter how small, tends to spread slightly when it hits the paper, and because the positioning of each drop is not precise, letterforms produced on these devices tend to be slightly darker and blobbier than letters produced on a laser printer. This has some advantages, notably that the spreading of the ink tends to reduce the appearance of the jaggies on diagonal edges, but it also gives a less crisp rendering of the letterforms.

Phototypesetters of today are usually Cathode Ray Tube (CRT) devices; that is, they have a beam of light that writes not on a screen, but on a piece of film. The beam *etches* the characters onto the film in the places where the beam hits the film. Because the beam is round, it tends to round out the edges and corners of the letterforms. Depending on the type of film, the device can produce either a white on black image (a negative image usually on transparent film) or a dark on white image (a positive image usually on photopaper). These typesetters are capable of very high resolution output, extending from around 700 dpi up to 5000 dpi.

The design of typefaces for display on CRT screens (such as terminals found in a computer environment) requires consideration of issues not addressed here [HOLL67]. These typefaces are usually quite low resolution (less than 200 dpi) and their rendering is sometimes dependent on the speed at which a beam can be turned on or off as it is deflected across a CRT screen. Since this is really a different medium altogether, we have chosen to disregard them in this system.

It should be noted that the production of fuzzy, squishy, blobby, or other imperfect renderings of the letters does not mean that there is something wrong with a printer; it is simply a characteristic that must be taken into consideration when designing letter shapes to be used with that device. One may, of course, choose to buy a printer for which these distortions have been minimized but in none of the printers are they nonexistent. There have been no machines in the history of printing that render an exact letterform image as a result of the normal techniques of printing. Letterforms printed on early presses had ink-squash around their edges where the liquid ink squished out from under the edges of the metal type when the type was pressed onto a piece of coarse paper. When the paper surface became smoother, letters could be rendered with more precision, but they were still not exact replicas of the typeface shapes. Because we are now using new technology, we have a new set of problems; it is these distortions that we must compensate for when we draw a new type design. Designs of the past solved problems that existed with old technology; if we hope to produce high quality typographic output using the technology of today we need new designs that can overcome today's problems.

Because of the differences in the output produced by different types of printing devices, it is imperative to *proof* a new typeface on the output device on which it will be used. This process consists of printing the characters at their intended size and in various combinations so that both the interaction of the character shapes and their spacing relative to each other can be judged. Evaluating output from one type of printer in order to produce the same results on equipment employing a different marking engine will not be successful. It is important also to evaluate the typeface when it is printed via a true *marking* device, or one that actually puts marks on paper, as contrasted with a *display* device on which the designs are displayed as video light patterns. Designs destined for paper output, when displayed on a terminal screen, are often distorted due to inconsistencies in the *aspect ratio* of the screen, or the ratio of the horizontal size of the screen pixels to the vertical size of the pixels. In addition, a typeface must be evaluated at its true size to determine whether the design is successful at that size. The bitmap pattern of the character can be displayed on a screen, but there are currently no screens with a resolution as fine as that of current marking engines on which to display a design at its true size.

Task Analysis

We have seen that changes in the design process evolve from earlier techniques to meet the needs of the current technology. It appears that this transition phase is a useful way to reconcile the means of dealing with a new form of technology with the forms and functions associated with an earlier one. What we would like to provide, therefore, is a system that can mimic the current means of rendering a design, yet also allow these methods to metamorphose into those more suited to the creation of digital designs. We need to supply the means for a designer to work with a familiar environment, but one that can also grow to meet unforeseen needs or demands brought on by changes in technology that dictate changes in the design process. Because this process is quite complex, the task of determining just what we needed within the system was not an easy one. Although each designer has his own methods of rendering a design, there are many similarities in how different designers work. Out of their myriad skills and methods, we had to build a system that would allow each of them to develop a design metaphor that was familiar and easy to use. If the principles of the traditional design stages can be maintained, the resulting system is more likely to be used by a designer familiar with the old methods. If the designer is also provided with tools that are specific to the computer environment and she is allowed to incorporate these tools into her design technique, then the potential exists for her to develop a new means of designing that is more tuned to the new medium in which she works. One aim, therefore, of our research is to provide a means for effecting the transition from the traditional methods of creating typefaces to new techniques specific to the raster revolution.

Interactive Design

An interactive typeface design environment utilizes developments from many overlapping areas: interactive user interfaces, adaptive or extensible systems, graphics systems allowing manipulation of objects, man-machine interactions, human visual perception, and ergonomic design. We want to provide *synergy*, or a combination of these areas within a creative environment to form a new approach to the design process.

Since we are creating an interactive design system for use by persons whose profession is the production of graphic objects, it would be incongruous to provide a system that does not take advantage of the current research in user interface design that emphasizes visual interaction and manipulation of objects. The use of icons and an interactive graphics display screen with a mouse or other pointing device to provide selection of objects on the screen has now become commonplace. One of the first systems to use this approach was Pygmalion, a system created by David Canfield Smith for his PhD thesis [SMIT75] which was implemented in Smalltalk, and was an attempt 'to provide an artistic resource which computer scientists can use to create.' Smith had a broad view of a computer scientist, applying the term to anyone who knew how to do something and wanted to use the computer to do it. With Paragon, we would like to provide a computer resource that graphic artists can use to create.

Paragon falls into the category described by Sheil [SHEI83] as *viewer-based tools*, or those in which information needs to be retrieved quickly, displayed effectively, and modified easily. The users have a limited knowledge of the inner workings of the system with which they deal.* We cannot, however, rely only on the use of visual feedback and graphics to achieve an acceptable result. As stated by Foley and Wallace [FOLE74], 'The clarity and vividness of computer graphic communication is not an

* He contrasts these with *knowledge-based tools*, or those that must know a significant amount about the content of a user's program and the context in which it operates, such as in an expert system.

automatic consequence of the mere use of drawings. Conscious design effort must be applied by an application programmer to provide clarity and vividness in the user's communication with his machine. In computer graphics, the effort must be applied to designing the action sequences by which the user communicates his desires to the machine, and the pictures by which the machine communicates responses. The objective is to make both paths of communication natural to the user to increase his productivity in technical or artistic tasks. This design effort is sometimes described as ***human factors design*** or ***ergonomic design.***'

This idea is expanded by Bennett [BENN76] when he suggests that the quality of graphic interaction language may be examined in terms of what the user *sees* at each interaction point, what he ***has to know*** in order to interpret what he sees and what actions he can take at the interaction point. He states that 'the most fundamental part of user orientation is the one involved in determining the basic constraints that will define and structure the repertoire of graphic forms and graphics operations, that will limit it to a learnable and usable size.' He felt that a system should be designed so that a user could avoid boredom, panic, frustration, unexpected long delays, confusion, and overload of information. The response of the system to user interactions should be immediate and visual.

We have already discussed the way a designer works and ways in which to recreate a designer's environment. This process of evaluating the actions performed by the users of the system, was labeled ***task analysis*** by the Star development group [SMIT82]. Before we began the work of designing the internal workings of Paragon, we evaluated the tasks that a designer performs in a working environment so that a set of smaller and more syntactically simple actions could be extracted to form a base repertoire of operations upon which more complex actions could be structured.

By providing visual counterparts to every aspect of the system including the commands and the display of information, the implementors of the Star system tried to relieve the user's short-term memory from having to 'remember' too many details. They decided that visual communication was more efficient for the display of information so they chose to make all objects and actions in the system visible. Everything to be dealt with and all commands and effects had a visible representation on the display screen or on the keyboard. Commands were generalized to provide similar actions no matter what objects they were invoked on. For example, the *move* command moved documents to the printer queue, mail to a mailbox, characters within a document, or icons on the screen. We have chosen to model our system along these same lines.

The user interface can be compared to the dialogue between the system and the designer. A friendly system will converse in a language that a designer can easily learn and understand, whereas a system that has not been written with the interface as a major concern may be complex, frustrating, and not easily learned or used. We must provide both a graphics interface and an integration between the use of the system and the extension of the system.

Design Environment

Borrowing from Barstow's definition of an interactive programming environment [BARS84], we can state four requirements that we would like to satisfy in our design environment: first, we want to provide a large set of tools, most of which are specific to the purpose for which the system is written; second, we want to use an underlying structure as an organizational tool; third, we want to allow incremental program development both in the design and the maintenance of the system; and last, we want to create a highly interactive environment and provide a fairly rapid response rate between the user and the environment. We also want to tailor the appearance and the interactions of the users with the system so that it is easy to learn and easy to use. There should be provisions for incremental learning, so that as a user's sophistication with the system increases, his use of the system can also increase in complexity.

As the system is running, it must provide immediate visual feedback for every selection on the screen; the user must not be left to guess what the results of the action were. At the very least this can take the form of messages displayed that describe the state of the system at that particular point. These messages can provide tutorial information, help information, error messages, or prompts.

Jacks [JACK70] suggests that a graphics system should provide facilities for providing a group of descriptive geometry operations that may be applied to the design on the screen and that are analogous to a set of subroutines provided in a program library. In addition the system should provide *sequence recording* where the system has the capability to remember a sequence of operators and the facility for the sequence to result in a new operator. The resulting operator should appear no different to users than the primitive operators in the system. There must be a facility for remembering and naming operations, for defining the input and output to and from an operation.

In defining the graphics operations that we have chosen, we have attempted to follow the guidelines set out by Scott in her book *Introduction to Interactive Computer Graphics* [SCOT82]. There, she describes the components of an interactive graphics system: first, it contains a command language that is a set of rules by which the user and the computer carry on their conversation such as command selection: activating an available graphic operation; positioning, or supplying information about location and size; and pointing, or identifying existing objects; second, the system must be able to handle lots of data for graphic objects; third, it must respond instantly by visually displaying the result of each user; fourth, it must provide an input technique that is simple and direct; and last, it establishes an 'intimate' association between the graphic and non-graphic data. In addition, a graphics system should provide facilities for geometric transforms such as translation, or the uniform motion of an object along a straight line, scaling that increases or decreases the dimensions of an object, and rotation where each point on the object moves in a circular path around the center of rotation.

Traditional Design Practices

A type designer's traditional tools include paper, pencils, and erasers, and sometimes a pen or brush. The first stage of letter design is sketching of letterforms, serifs, or related shapes. These sketches might be done with a flat or fine nib pen, but if the character shapes begin by being recognizably calligraphic or characteristic of the drawing implement, they usually do not end up so; the path of the pen provides merely a skeleton that is then fleshed out by continuous smoothing and refinement.

Once an idea has been developed, the characters are drawn with finer lines and smoothed edges. At this point, they are probably drawn in a size larger than that of the final design size, although it is not uncommon for a designer's first sketches to be done in very small sizes. The second stage drawings are usually outlines, typically from 3 to 10 inches tall, since it is easier to modify outlines than to change filled shapes. The outlines are drawn on very smooth transparent or translucent paper and the edges are refined by the designer: a shape is drawn on one side of the paper and modified by erasure and redrawing until a satisfactory drawing is obtained; the paper is then turned over and, using the shape seen through the paper as a guide, the edges are refined. By using both sides of the paper, the designer can modify the previous version without having to erase the edge being changed, and he can avoid drawing on top of what might turn out to be a better edge. The designer alternates drawing on the front and back until either the shape is finished or the paper begins to wear out. In the latter case, the shape is traced onto a new sheet of paper and the process begun again.

Each character is drawn in this way, but since one of the characteristics of a typeface is its *salient features*, or those features that distinguish it from other typefaces, parts of characters that have been designed can be traced onto other characters, thus facilitating the production of the final design. This is particularly true of serifs, stems, bowls and arches (Figure 31). Some of these parts are merely shifted to accommodate different characters; others must be rotated or reflected to achieve the desired result.

Usually before any character is drawn in its final shape, the dimensions of the typeface have been determined so that the size of the characters and the features within them will match. These metrics are all offset from the *baseline*, or the line upon which all the characters rest. The other measurements include: the *x-height*, or the height of the lowercase x and thus the height

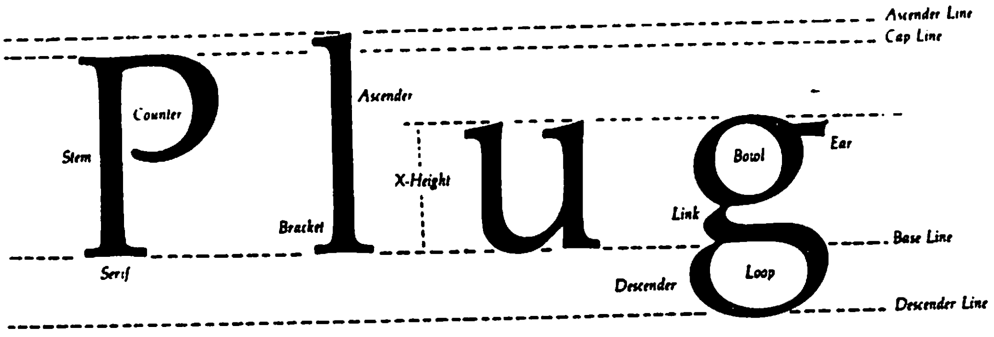


Figure 31. Salient features of a typeface.

of the central part of the lower case letters; the *ascender* height and *descender* depth; the *cap-height*, or the height of the capital letters which is not necessarily the same as the ascender height; and the *o-height*, or the amount a curved or angular edge will overhang the nominal height. These edges are drawn taller (or deeper) than their squared counterparts to compensate for the visual perception of curved or angular edges as being shorter [LEGR16, GILL80]. Other horizontal guidelines may also be used but need not be.

Within the current systems for specifying letterforms, the practice of fitting each letter within a box and then lining up the boxes to form words has carried over from the days of metal type. Each letter must be placed within its box so that when it is aligned with other characters, it appears to be centered between them. To facilitate this adjustment, vertical guidelines called *sidebearings* are used. These indicate the left and right edge of the bounding box for the character. Two characters are spaced by aligning the right sidebearing of the letter on the left with the left sidebearing of the letter on the right (Figure 32). Each letter has only one set of sidebearings, so the boundaries chosen must permit the letter to space well with all the other letters. Difficult-to-space letter combinations may be *kerned* if the output system permits it, in which case the spacing for a particular letter pair is individually adjusted. Other letter combinations may be designed as *ligatures*, or letter pairs (or

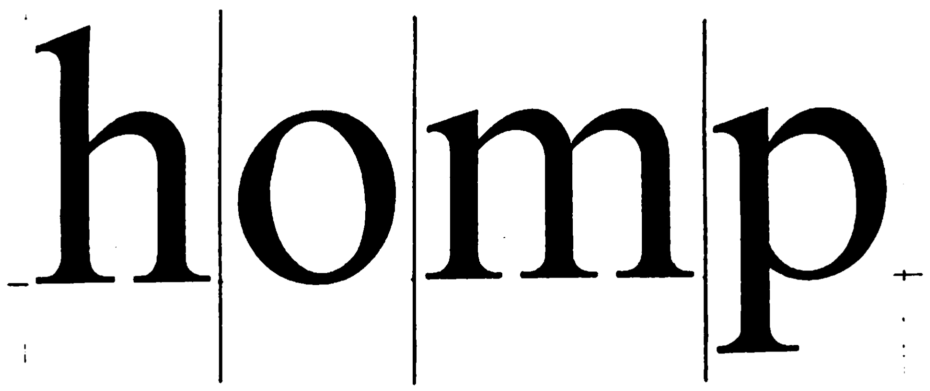


Figure 32. Characters with sidebearings lined up.

triplets) drawn as one glyph; for example, the letter 'f' followed by the letter 'i' is usually drawn as a ligature.

When several key letters have been designed, a test word is constructed. Key letters usually include one from each shape category—an ascender, a descender, a letter that fits within the x-height, a letter with curved edges, a letter with straight edges, and a letter with diagonals. Each character is traced from its proof letter (the perfected drawing) onto a clean sheet of paper and the outline filled in. The characters are then aligned and the drawings taped together so that the spacing between them is even. Then the entire construct is taped to a blank wall and the designer stands away from the letters as far as possible and evaluates them. The objective in moving far away is to see the letters as small as possible, with an optimum viewing size being that for which they are designed (text faces are usually designed for an optimum size of 10 or 12 points). Since it is difficult to find a room long enough to perform this process reliably, the designer often studies the shapes by looking at them through a reducing lens, which is, as may be supposed, the opposite of a magnifying lens—it makes the observed objects smaller. The goal of this exercise is to see what the letters will look like at their intended size so that deficiencies in their shapes can be determined before a significant amount of work has been expended.

These test letters are then refined and modified. When they are acceptable, work continues on the rest of the alphabet. Evaluation of the drawings is done repeatedly throughout the design process. Uneveness in the letterforms, both within a character and between characters, is removed so that the typeface as a whole retains continuity between its features and its overall presentation. A well-designed letter is of little use without a well-designed set of counterparts to accompany it.

Once the typeface design is finished it must be turned into a font that is a representation of the design at a particular size and resolution. Although there are several processes extant for doing this, the one we are concerned with converts the letterforms into *bitmaps*, or digital renderings of the design, via an input and conversion system. Each bitmap is a pattern composed of small, discrete elements, or pixels. The bitmap is created by converting an outline representation of a character into a raster pattern similar to that produced by drawing the character on a piece of graph paper and filling in the squares that fall inside the character outline. These bitmaps can be printed on digital output devices such as laser printers or CRT phototypesetters.

The digitization process averages about 30 minutes per character. The edges of each letter shape are first marked with *control points* and then, one at a time, the drawings are placed on a bitpad and the control points are selected by lining up the crosshairs on a puck with each point and pressing an indicator button. The location of the control points will vary depending on the input system used, so that a drawing marked for one system will need to be marked differently for use with another system. Typically, control points indicate the beginning and ending of straight line segments, the beginning and ending of curves, points along a curve, and corners. Once the points are entered into the system, they are scan-converted into outline renderings of the letter that are then digitized or converted into a bitmap representation of the letter shape. This digital shape can then be printed on a suitable output device and evaluated by the designer. The typeface features need to be individually tuned for each different marking engine for which the typeface is intended to be used.

The entire process of design, refinement, digitization, evaluation, and design modification usually takes a minimum of two years from the start of the initial drawings to a finished typeface. The sophistication of computer input systems has not simplified the designer's task; it has merely made it faster to achieve the final digital result. This, however, has been the case throughout the history of type design: the designer's tasks have changed over time, but because his task is defined by the means of rendering the design into a finished product, he remains subservient to its production, whether it be done by a punchcutter, a punchcutting machine, or a workstation and a bitpad.

Modeling the Design Process

Having enumerated the tools and tasks of a typeface designer, we can now specify what must be provided by a computer design system. First, we must have some capability of sketching or drawing. To provide this, we must have access to an input device that allows freehand movement of a cursor on a screen. Ideally, this device should resemble a pen, which leads one to speculate that a stylus or light pen would be the closest solution available with today's technology. Barring access to one of these, either a mouse and tablet or puck and bitpad would also provide free access to cursor movement. It is not clear how well a designer can adapt to using a device that is positioned in the palm of the hand and moved in a manner similar to that of rubbing your palm on the surface of a table rather than a more upright device that more closely resembles a drawing implement. It is possible that although it may be initially rather clumsy to use, a designer can get used to this different sort of movement and it is probably also the case that much depends on the individual designer. A secondary question is whether such a device can be positioned accurately, but here, only actual use of the device can give a satisfactory answer.

If we can draw with the device, we can also erase. This leaves only the surface on which to draw lacking from our model. A high resolution bitmap screen can provide the surface, and a method of overlaying bitmaps or planes can supply both the transparency for fine tuning a design (allowing a designer to draw on one sheet, yet see another sheet underneath) and the different sheets necessary for drawing individual characters. Lacking that, simply erasing and redrawing shapes on the screen, although slower, would be sufficient.

Curved Edges

The rendering of curved shapes within a letterform requires a very precise means of specifying the edges of a character shape: the final outline drawings when done by hand may be done to a precision of a hundredth of an inch. Slight variations from this pattern could unacceptably change the characteristics of the design. We would like to use a standard algorithm for generating, duplicating, or displaying curves but will need very high precision or very specialized control along the curved edges. Current typeface conversion systems use a variety of curve-generation techniques with some success, so it must be possible find algorithms that will closely approximate the designer's wishes.

Curve algorithms in current use with font-generating software include conics, which are used by the LetterIP system in use at Bitstream, by Ikarus output routines, by a contour fitting font editor developed by Pratt [PRAT85], and by an in-housefont development system at Imagen [BRAD87]; parametric cubics that are used by Metafont [HOBB85a] and storage of curve information within Ikarus; Bézier splines used by Fontographer [FONT87], Illustrator [ADOB87] and the in-house font design system at Adobe; and Hermite cubics used in work done at Xerox Palo Alto Research Center (PARC) on the automatic fitting of curves to rasters [PLAS83].

In selecting a representation for curved edges in our model, we must consider two properties: the order of continuity of the joins between edges and the location of control points that define the edges. In modeling a complex shape we will use several line segments joined at their endpoints. This congruence of two endpoints is often referred to as a *knot*. The line segments may be curved or straight and the joins between them may be smooth or angled. If two line segments are joined at an angle such that the slopes of the two segments at the knot are not equal, then the join has zero-order continuity (Figure 33a). If the line segments are tangent at the point of intersection they have first-order continuity (Figure 33b); and if the curvature of the two line segments is the same at the knot, they have second-order continuity (Figure 33c). We would like to employ a model that allows up to second-order continuity.

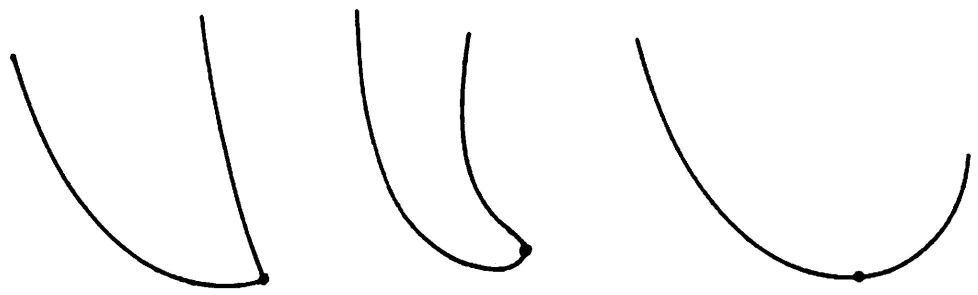


Figure 33. Zero-order continuity (left), first-order continuity (center), second-order continuity (right).

The location of control points to define the path of the curve is our second consideration. We will examine several types of curve modeling to determine their suitability for our purposes. Drawbacks to each of the models will be examined as we describe each of the types of curves.

A *Bézier curve* is defined by at least four control points. The first and the last points lie on the endpoints of the curve; the second and third points control the shape of the curve but lie off of it (Figure 34). To change the shape of a Bézier curve, one must adjust the control points that lie near the piece of the curve to be changed. Only the piece of curve that is under direct control of these points (that segment of curve that is between two knots) will be affected; changes will not affect other pieces of the curve. Control within the piece, however, is not *localized*; moving any point will change the shape of the entire piece. This has distinct disadvantages for a designer desiring to change only a small section of an outline; changes to one control point may affect the entire outline with unpredictable results.

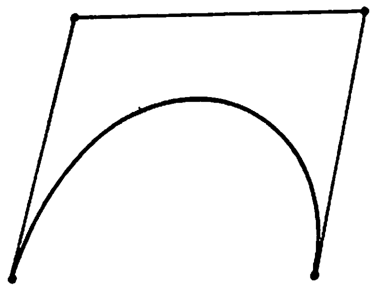


Figure 34. A Bézier curve and its four control points.

To produce a complex curve, several Bézier curves can be pieced together. Order of continuity can be achieved through the positioning of control points that coincide or lie along a straight line: zero-order continuity is achieved simply by having two endpoints coincide; first-order continuity is achieved by having the first edge of the first curve be collinear with the last edge of the second curve; and second-order (and higher) continuity can be achieved through further geometric constraints upon the control points. Since we do not plan to incorporate constraint solving in Paragon, maintenance of continuity would have to be done manually by the designer.

The use of ***spline functions*** allows a higher level of continuity to be achieved automatically. These functions also provide the use of localized control: changing the position of a control point changes only the portions of the curve that are located near the point; it does not affect the curve as a whole. The b-spline algorithm only approximates matching the endpoints rather than matching them exactly in its attempt to fit a curve to a set of control points and does not necessarily pass through any of the points (Figure 35). It does, however, guarantee that the curve that is generated has both first- and second-order continuity. The use of b-splines reduces the need to piece curve segments together; control points can be added without affecting the *degree* of the curve, which is a measure of how difficult it is to control. Multiple control points that coincide can be specified to achieve regions of high curvature. A disadvantage to this model is also apparent: a designer needs to control exactly which points a curve will pass through; an algorithm that only approximates the control points is not close enough when one is making a precise drawing of a character shape.

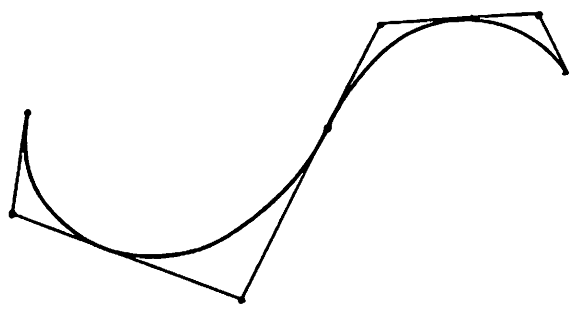


Figure 35. A b-spline curve and its control points.

Conic curves and the means of using them in the specification of outlines for letter shapes have been discussed by Pratt [PRAT86] and Pavlidis[PAVL83]. These curves can be computed quickly, and are fairly easy to specify but require a little more understanding by the designer as to just how to control the shape of the curve than spline curves. The user must position three points for each curve segment: two endpoints and one point that controls the shape of the curve, often called the tangent point (so called because a line drawn from that point to each of the endpoints is tangent to the conic curve at each point). In addition the sharpness of the curve can be specified, a factor that controls how close the curve passes to the tangent point. Continuity of the curve at a point that anchors to two different segments is not guaranteed but can easily be achieved by aligning the two tangent points and the anchor point.

Another form of curve modeling was developed by John Hobby as part of his thesis [HOBB85a]. This model uses parametric cubics with a means of computing the curves through a system of linear equations. First-order and approximate second-order continuity can be achieved. Control points are specified that lie on the curve with additional freedom provided by the ability to specify tangent directions at each control point. This model also provides for approximate locality so that changes to one part of the curve will not adversely affect other parts of the curve (this is approximate in that it provides for an exponential decline in influence relative to the distance from the knot rather than limiting the decline to within a fixed number of knots). In addition, curves generated with this algorithm are invariant to changes of scale, degree of rotation, or angle of reflection. Because this curve model was specifically designed to be used with the font-generating capabilities of Metafont, it has been tuned to correspond to a designer's needs and thus most closely models the environment that we want to achieve with Paragon.

A further advantage of his curves are that they pass through all of the specified control points. This is very useful for a designer who wants to be able to accurately position points that will lie on the contour of a letter. It is much easier to position and move points when the user knows that there is a direct relationship between the points that he specifies and the curve that will be drawn. With conventional b-spline, Bézier splines or conics, the user must specify or move additional control points that do not lie on the curve that is drawn.

The specification of Metafont curves differs in a subtle way from other curves in that with splines or conics, the shape of a line passing through a particular point is determined by the type of point, i.e. whether is is a straight or a curve point. With Metafont, the designer specifies the shape of the path between two points. Although this difference does not affect the final shape of the outline, it does affect how the user specifies what the outline should look like.

Because of the ease with which curves can be specified with the Metafont algorithms and their closeness to the design paradigm that we wanted to achieve, we chose to use Hobby's splines within our system.

Guidelines

When designing a typeface, a designer may want to display guidelines for the baseline, the x-height, the ascender height, the descender depth and the left and right sidebearings. Since these are not absolutely required by the designer (depending on how the designer works or what sort of characters are being designed), they are not displayed automatically, but are added by the designer as they are needed. In addition, other guidelines, either vertical or horizontal, may be drawn. Angled guidelines are useful when the designer is working on any shape that contains angled edges, or when working on a typeface that is not oriented in a strictly horizontal alignment such as an *italic* or a *slanted roman* face.

Rulers

We would like to provide a means of measuring the components of a typeface using a variety of different metrics. The measurement of font sizes and features is usually done in points (approximately 1/72") or picas (12 points = 1 pica). Additional measurement units that we might want to provide include millimeters, inches, *Didot points* (and the related *cicero* used in many European countries to measure font and type sizes and equivalent in use to our point and pica though not equivalent in size), dots per inch, *dpi*, for a particular point size and resolution and *dpm*, or dots per *em*, a space equal to the square of the body of the type used. To provide the means for measuring different features, the ruling device should be movable by the designer to any orientation; that is, it should not be restricted to just horizontal or vertical measurements. It may also be helpful to provide a protractor for measurement of angles and a means of comparing two or more measurements.

When an outline typeface design is converted into a bitmap representation, or digital typefont, the guidelines (especially x-height and the sidebearings) must align with the pixel boundaries. This will usually result in an adjustment in the positions of the guidelines with respect to their original positions. This adjustment should be automatic so that the designer does not have to worry about aligning each character with the different guidelines, but the adjustment will need to be monitored by the designer to insure that the resulting characters are not grossly distorted. The coordinates of the new alignments must be stored with other data specific to the font.

Scan Conversion

Because scan conversion of the outline shape is such a crucial step in the process of creating digital characters, it will be worthwhile to take a closer look at it. There are many obvious problems in this process, and some not so obvious. Because we are dealing with particularly small objects, and ones that will not withstand much distortion, we must be especially careful in providing the means for careful tuning of the final character shapes.

The process of converting outline characters into digital bitmaps is not simply a process of filling a contour as is done for outline shapes in many graphics systems. Letterforms have certain properties that dictate that techniques that work sufficiently well on large, relatively imprecise objects do not work well on small, finely designed letterforms. The property that most affects this situation is the small size at which these objects are generated. When the final size of the character must fit in a box that is less than twenty pixels square, the difference of a few pixels here and there can make quite a difference in the final result.

When a typeface is designed, it is usually designed as a *family* of related styles; a roman, a bold, and an italic are the most common faces, although some designs include variations such as condensed, slanted, expanded, or ultra-bold. The relationship between these styles must be such as to allow the difference between them to be readily distinguished, since they will be used in conjunction with each other to provide contrast in the textual appearance of a document. An obvious example is the use of roman text for the body of a document, with emphasis on specific words indicated by the use of either a **bold** or an *italic* font for those words.

When creating a typeface design, the designer usually has taken into account characteristics such as the stem width and the width of each of the characters when distinguishing one style from another. Bolder characters are often wider overall than their roman counterparts; the stem widths of the characters will be wider, and other features may be exaggerated. This care in the design, however, can be thwarted by the peculiarities of matching a design to a particular grid size. If, for example, a roman style letter has been designed with a stem width that ideally would fit into 2.5 pixels, but after conversion rounds to 3 pixels, and the matching bold design ideally fits into 4.3 pixels but instead gets rounded to 4, the resulting font may not have enough contrast between the two styles to make them readily distinguishable on a page of text.

In fonts generated for medium resolution laser printers (240–400 dpi), this situation is not uncommon. For a 10–12 point font, the stem width of the lower case letters is usually only 3–4 pixels wide. If one is trying to generate both a roman and a bold face at this size, it would be helpful to know in advance what stem width will be used at a particular size for a particular resolution so that critical dimensions can be determined early in the design process.

The situation is actually even more complicated, since a designer wants to generate not only different styles, but several different sizes of a font. A one pixel wide stem is generally too narrow to be used for any letter size due to the vagaries of printers in printing such fine lines, so a minimum stem width usually starts at 2 pixels. If we wish to generate fonts in sizes ranging from 5–6 points up to 14 points, there are not enough increments in whole pixel sizes to give us a different stem width for each size, given that we want to create a classic *text face*, or one that is intended to be read at small sizes and in quantity such as those used in the printing of books, or column copy in newspapers and magazines.*

If we were to increment the stem width by one pixel for each point increment, the stems of our largest characters would be as wide as the total width of the smallest characters. (Note, we are not suggesting an increase in the total width of the character by one pixel with each increment; we are suggesting an increase in

* A *display face*, by contrast, is designed in larger sizes and usually attempts to attract attention; it may, therefore, have gross character distortions, tight inter-character spacing, or ornamentation built into the design; although a great deal of care and precision goes into the design of a display face, it is not necessarily designed to be easily readable.

the stem width of the characters. In the case of the lower case 'm', this could mean an increase of 3 pixels in the total width of the character—one for each stem.) In this case, a more important measurement than dpi is that of dots per em, or *dpm*. If we know how wide the body size is in pixels, we are better able to estimate the widths of various character parts. To accommodate all the various font sizes while maintaining the design, we may need to create several different sizes with the same stem width, and use other distinguishing characteristics to indicate the differences between them (changing the vertical dimensions of the characters, for instance, or the distance between the stems).

The objective in creating digital letterforms is to fill an idealized outline, one that has in itself no width. If we were to determine an outline of a character and draw that outline with a one pixel wide pen, then fill the outline, the resulting bitmap would be one pixel too wide between each pair of edges in the character (Figure 36). This is also known as the *off-by-one* phenomenon. This difference is unacceptable when dealing with objects whose total width may only be a few pixels wide to begin with. For this reason, we must provide a conversion algorithm that fills to the edges of a given outline but not beyond it.

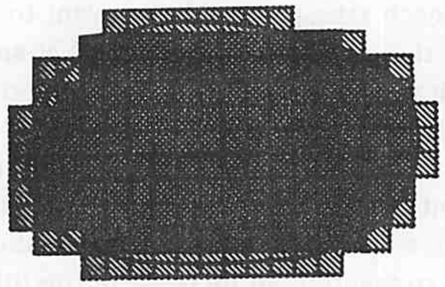


Figure 36. A bitmap that includes the outline when it is filled is one-half pixel too wide on each side resulting in the off-by-one phenomenon (light hashing); bitmap that correctly fills the outline (dark hashing).

Another consideration when generating bitmap characters from outlines is the position of the outline in relation to the underlying grid. Slight changes in the alignment of a character upon the grid can have significant effects in the final rendering of the bitmap. The dimensions of parts of the characters can vary depending on their placement on the grid (Figure 37). In particular, one feature that is affected by the grid position is the amount of overhang of the curved and angular shapes—if we are digitizing on a very coarse grid one pixel can be quite large relative to the character height (Figure 38). We may not want to overhang the edges of the character at both the top and the bottom of the x-height, but instead just overhang one or the other (and at the same time maintaining consistency between the letters within the typeface). We must, therefore, provide a means of either shifting the outline or shifting the grid to better align the edges of the characters with the edges of the grid.

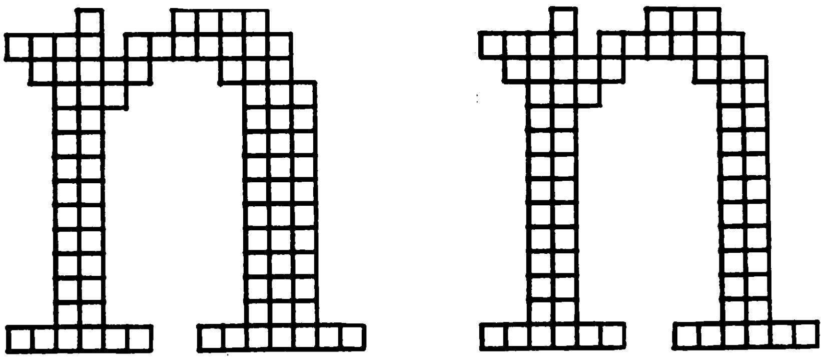


Figure 37. Uneven stem weight results from mis-alignmentof outline character upon grid.

Displaying Bitmaps

Displaying a bitmap that has been derived from an outline character introduces several problems that need to be resolved. The first obstacle is that the coordinates that define the outline may not map evenly into coordinates that define the grid. This has been mentioned before in the example of a stem width that

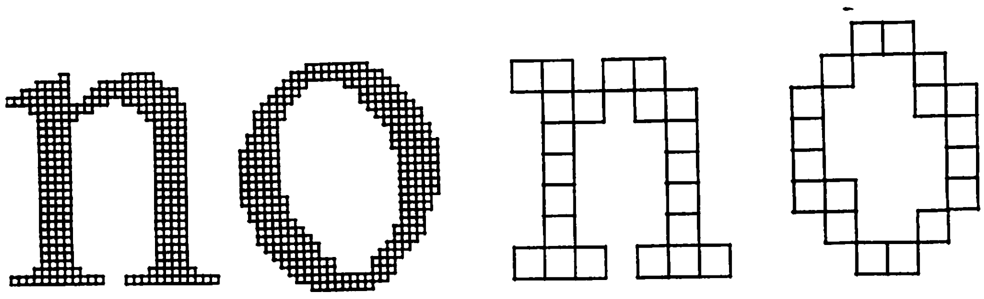


Figure 38. Overhang of curved edges produces acceptable results at high resolutions (left) but results in letters that are too large at low resolution (right).

should be 2.5 pixels wide; we must decide whether to round down to two pixels or up to three. A further problem develops when we try to display the outline representation of a character along with a digital representation of the character at the same size as the outline on a bit-mapped screen. To help elucidate the problems that this process presents, we will describe a detailed example. All computations are approximate (rounded to either one decimal place or an integer value).

In the following discussion, a *raster* character is an outline character that has been scan-converted into a bitmap. The *size* of all characters within a font is the total vertical dimension of the font, or the measurement from the depth guideline to the height guideline. We also need to distinguish between the resolution of the raster character, or the resolution of the projected output device (here referred to as *real* pixels), and the screen resolution, or the resolution of the display that we are looking at (here referred to as *screen* pixels). In the current implementation, the screen resolution is 100 dpi, with each paper window encompassing a 600 x 768 area on the screen (three-quartersof the available screen dimensions). Thus the maximum screen size of any character is 600 screen pixels.

Given an outline representation of a character, with coordinate knots marked, we want to convert the glyph into a 12 point raster character for a 300 dpi printing device. The height of a 12 point character at a resolution of 300 dpi is 50 real pixels (at 300 dpi, 72.27 points per inch, and a 12 point character, size=300/72.27 x 12 which is 49.8 pixels. Because we cannot print partial pixels, this is rounded to 50). Now suppose we want to superpose the raster image over the outline image. To do this, we can magnify the pixels in the raster so the raster image is the same size as the outline. If the size of the character is 400 screen pixels, we can display each raster pixel as an 8 x 8 square on the raster screen (400 screen pixels/50 real pixels = 8 screen pixels per real pixel). This example works out very well. Suppose, however, that we want to generate and display a 10 point raster from the same outline. The size of the raster in this case is 42 pixels (300/72.27 x 10=41.5 which rounds to 42). When we try to display this, each real pixel is represented by 9.5 screen pixels (400 screen pixels/42 real pixels=9.5). If we round this either up or down, the raster image will be 5% larger or smaller respectively than the outline we started with (Figure 39). It is here that we have to make some decisions about what we are going to display.

If we decide to scale the outline to fit the raster, the outline of the 12 point character will no longer match the outline for the 10 point character. If, however, we do not scale the outline of the 10 point character, the raster will not appear to fit within the outline.

An alternate representation would be to present both the 12 and the 10 point rasters with the same size pixels (so that instead of displaying the 12 point character with 8 x 8 pixels and the 10 point character with 10 x 10 pixels, we would display them both with pixels the size of the larger character, thus 8 x 8 pixels). In this case, the absolute size would not match, but the relative sizes would be drawn to the same scale.

Ideally, we want to provide a selection for the various representations that the designer might like to see. These choices include:

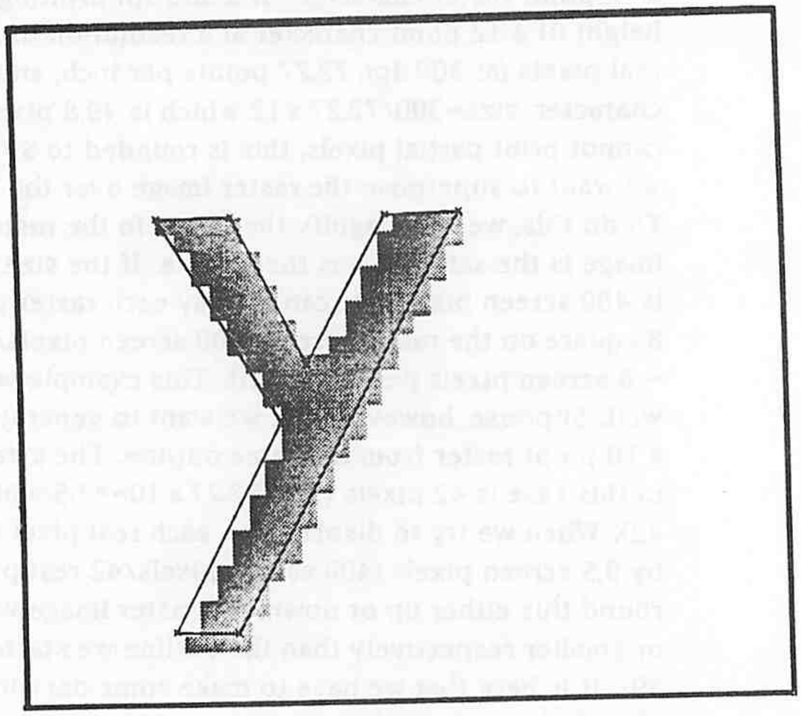


Figure 39. Raster image is larger than the outline image due to rounding of pixel size.

- scaling the outline image to the raster if there is a difference of more than a certain percent; that is, we will adjust the coordinates of the outline image (and the guidelines) so that they match the raster grid. This means that if the raster is 5% larger than the outline, we will also enlarge the outline by 5%. Because the difference is dependent on the chosen resolution and point size, the mapping only applies to the screen display and not to the coordinates of the knots that are stored in the database. Those knots will remain independent of the raster conversion process. This representation will most likely be used by a designer to compare outlines to bitmaps.

- selecting a *master* size and displaying all rasters at that size; that is, all rasters will be displayed with the same size screen pixel that is calculated for the master size. This representation is most useful for comparing bitmaps to bitmaps.

Another problem occurs if we try to scan-convert a character and the resulting raster character is of a higher resolution than the screen resolution. For example, if we try to generate a 72 point character at 720 dpi the raster size of the character is 717 real pixels. If our screen is only 600 screen pixels tall, we are unable to display the entire raster character at one time on the screen. The system, at this time, is limited to those sizes that can be displayed in their entirety on the screen.

Spacing

Once several characters have been designed, we must be able to evaluate them when they are displayed in a line of text with other characters. This requires that we provide some means of specifying and possibly displaying the spacing of the characters. There is one traditional way to space digital characters and two less common ones, each contingent on the mechanism that will be used to typeset the characters. The formatters TEX, Scribe, Troff and most other computer typesetters maintain a model of letters as being enclosed in little boxes and consequently contain routines to line up the boxes along a common baseline. The more complex systems allow the boxes to be overlapped along their edges, but to use this technique with one of the current typesetting systems, the amount of overlap would have to be specified for each pair of characters. This is not a practice that we want to encourage as it is simply too time consuming and too complex to specify the spacing for every pair of characters.

An alternative spacing mechanism has been developed by David Kindersley and Neil Wiseman [KIND69, KIND76] for their Logos system. The spacing between characters is calculated automatically and requires no specification of sidebearings by the letterform designer. Logos spacing uses mathematical calculations for determining the optical center of each character and a standard width is then used between the centers of adjoining characters (Figure 40). The precise formulae for determining the character centers is a trade secret, so only typesetters who have licensed the calculations can actually set characters using this algorithm.

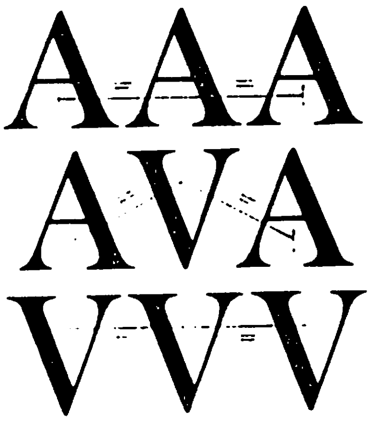


Figure 40. Spacing of characters as determined by the Logos system.

A third form of spacing allows characters to have uneven boundaries, with the boundary of one character butting up against the boundary of the next (Figure 41). This approach has been called *sector-kerning* [NAIM85]. The advantage of this method is that the boundary can vary along the height of the character so that a character such as 'V' with a wide upper boundary has wide spacing at its top, but narrow spacing at its bottom. If it is aligned with another character whose height is the same as the V, the upper boundary will determine the space between the two, but if it is aligned with a smaller character (one that does not go above the x-height), the small letter will butt up against the narrow part of the V and will thus space closer to it, which is how it should be. As with the Logos system, however, there are currently few typesetting systems that allow this sort of spacing to be used.

No matter which spacing algorithm we choose to employ, the grid size is still a limiting factor for any spacing that is specified for a character; that is, the boundary specified by the spacing mechanism must conform to the resolution of the final character grid. If we generate a character at 300 dpi, we must

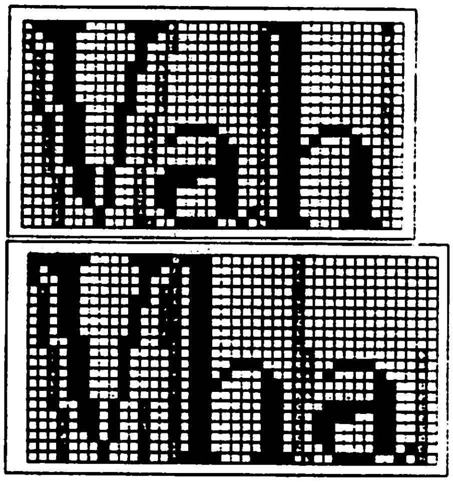


Figure 41. Uneven character sidebearings allow for adjustable spacing.

also space the character at 300 dpi. This indicates that final spacing determination and evaluation must be done at the font resolution, not at the outline or ideal resolution of the character drawings.

Interpolation
One feature of existing input systems that has been found to be quite useful is the ability to interpolate between designs to create a typeface lying between two extremes, such as the creation of a bold face, given a roman and an ultra-bold. This technique usually requires that control points on the two extremes be matched and of an equal number so that the system knows which points to interpolate between. Work has been done to develop a more sophisticated algorithm for use by the Ikarus system which allows interpolation to be done between characters with an unequal number of control points.

Library Functions

In addition to providing storage for data, we want to save any additional functions that a designer has defined. This *customization* should allow the designer to create new commands, add them to the system, then have them saved automatically when he finishes with a design session. The functions should become part of the system repertoire so that they are included in the actions window each time the user runs the system. One user's functions, however, may not be suitable for another user of the system. We want each user to be able to create a personal version of the system without interfering with another user's version.

Database Facilities

Once the designer has created some typefaces or typefonts, they need to be saved in an orderly fashion without requiring that the designer know too much about the organization of this data. The internal organization of the data must be easily accessed and updated, and must put no constraints upon the designer such as requiring that letters be designed or retrieved in alphabetical order. We must also provide storage and access for glyphs that are neither letters nor printable characters, but may be merely pieces of characters that must, in some way, be identified in a consistent and coherent manner.

To provide a sufficiently rich and varied view of the data that needs to be saved, we need to identify the different sorts of information that a designer might want to store, and provide a means of identifying it without forcing the designer to be too cryptic. We do not, for example, want a designer to have to refer to 'serif type X986.3'. We would, however, like to provide different categories of data so that we can incorporate some sort of organization on large amounts of information, but we must be sure that the categories are of sufficient depth and breadth to cover any organizational possibilities that a designer might want. An obvious hierarchy is to provide a means of saving individual typefaces, within which are individual fonts (characterized by a specific size and resolution); each face or font may have individual characters (alphabetic, numeric, or pi) associated with it (each of which is part of the repertoire of recognized printing characters), in addition to which there may also be glyphs, or character parts such as serifs, stems, bowls or arches.

The use of component parts as salient features within a typeface could be supplied by graphics software that contains graphic manipulation routines for reflection, rotation or copying of selected areas on the display screen. Other transformations that would be useful are the ability to enlarge parts of the drawing (to 'zoom' in on it), and to move or delete parts.

Summary

We have evaluated the task that we are planning to emulate and have proposed a variety of tools that we will use in the implementation of a digital type design system. These tools span the gap between the familiar and the innovative in providing a variety of functions to a designer using the system. We must now devise a mapping between these tools and our workstation environment. In the next chapter, we describe the system that we have designed to meet our objectives.

Chapter V

Architecture of the System

We should recognize that some applications are best thought of as design problems, rather than implementation projects. These problems require programming systems that allow the design to emerge from experimentation with the program, so that design and program develop together.

B.A. Sheil, *Power Tools for Programmers*

The organization of Paragon's design environment is based on an underlying model that roughly corresponds to the functionality of the system. Currently, it consists of a *basic design system*, a *database* of glyph information, a *library* of actions, and a *customization toolkit*. The basic system contains *primitive* operations directly implemented in C code. Some of these operations are specific to type design while others which are more general include graphics commands used to manipulate data and operations to retrieve and save information in a database. Design-specific operations include provisions for specifying guidelines (such as baseline, x-height, height and depth), selection of different point sizes or resolutions, and selection of specific glyphs from the database. The database allows a designer to save individual character data as well as glyph, font, or typeface information (a glyph is a character part such as a serif or stem, or a combination of other glyphs). The library contains user-defined commands that are built up through use of the customization toolkit. The toolkit package contains functions that allow a designer to create a personalized type design system by combining primitive operations into new functions and then adding these functions to the design environment.

Implementation

Paragon has been implemented on a Sun workstation with a high-resolution display monitor (100 dpi, a low resolution for output devices, but a high resolution for screen display), 2 megabytes of memory, a keyboard, and a mouse with three buttons. The two means of user input are the keyboard and the buttons on the mouse. The mouse provides control of a cursor that is displayed on the screen and thus used to point to objects on the screen which are selected by pressing one of the mouse buttons.

When we began work on this project, a Sun was the only machine available to us that had the hardware and software for interactive graphic applications. Other machines with appropriate computing power that were available were large scale multi-user multi-processing systems but these had no provision for the display of graphics or interactive manipulation of graphic data. The available graphics software on the Sun at that time was not very satisfactory, but since our experience with machines of this sort was not very extensive and the number of machines with the necessary resources was so limited, we had little to compare it to and thus did not realize until late in the implementation just how extensive the problems were. Many of the problems we encountered in the implementation of Paragon were not directly related to the brand of machine but due to the infancy of its software.

The system that we began with was a preliminary version of the software that was eventually released as the Sunwindow package. Use of Sunwindow had a tremendous impact on the final design of our system, an impact that we were unable to anticipate or to alleviate once we were aware of it. One of its features that we had to contend with was the mechanism by which the programs processed input from either the keyboard (key click) or from the mouse when one of its buttons was pressed or toggled. The system could track mouse movement by means of a cursor displayed on the screen, but could not be programmed to respond to mouse movement unless there was an accompanying mouse selection.

The biggest disadvantage in using a Sun was the unstable working environment that we had to deal with over a period of about five years. We began working with one of the first Suns available on the market. The accompanying software included a complete Unix operating system but graphics support was minimal. For the first two years, there were additional releases of the software that made the earlier systems obsolete. Even at the later stages of the development of our system, the Sun graphics software continued to change and eventually reached a state where the next upgrade was incompatible with the earlier version that we were using. Because we could not spend our time keeping up with the various releases, we chose instead to freeze the Sun graphics software that we were using until we finished development on Paragon. Although the Sun software now appears to have reached a stable state, because of our dependence on graphics support software which is no longer available, our final system will not run under the most current version of the software. Paragon was thus obsolete when it was finished.

Another limiting factor in our use of a Sun was the amount of memory available on the machines. We wanted to use some features of the window package that would allow us to do rapid redrawing of the display screen, but due to the limited amount of memory we were using, we were unable to take as much advantage of this feature as we would have liked. At the time we started the project, we thought we had a sufficient memory allocation, but as the system grew, we became more and more limited with what we could do with it.

The final limiting factor in our use of a Sun was the resulting system's dependence on Sun software. Although we originally planned to design the system so that all graphics software was isolated in one module, thus facilitating conversion of the system to run on other devices, we had to abandon this modularity along with some of our early ideas about system design when we decided that it was more important to get any version of the system running on any machine available than it was to maintain future portability.

Another piece of hardware that we considered using with our system was a bitpad with a high resolution input pad and a puck with five buttons. The bitpad would have provided more precise input data than the mouse, but it was not as easy to manipulate since it required that the user watch the puck and not the screen and did not provide continuous cursor movement. Feedback from the puck was provided only at each button click resulting in only one set of coordinates. This did not allow tracking of a cursor on the screen, dragging of objects on the screen, or drawing functions such as those needed to mimic the action of moving a pencil across a sheet of paper.

In retrospect, the Sun had a reasonable working environment. If the choice of machine were to be made today, it would be difficult to choose a system from the many similar systems now available. Each provides some outstanding features yet each also provides drawbacks, ranging from screen size, availability of development packages, accurate display of information, to ease of implementation.

Interface Design Decisions

We wanted to build an interactive, extensible design system for graphic designers who are unfamiliar and, in some cases, uncomfortable with computers. For this reason, we decided that all input to the system would be through movement of the mouse and selection of mouse buttons with the exception of items such as file or glyph names which would be entered from the keyboard. The users of the system are classified as novices, that is, they have little experience with computers and no knowledge of programming.

Keeping these goals in mind, we built and tested several preliminary systems before deciding on form of the final system. While testing some of our design ideas, we discovered that it was difficult to implement our initial ideas for interactive environment with the Sunwindow package. Many of these problems were due to early versions of the package. Two of the problems that we encountered were:

- Popup-menus did not cross window boundaries so if the user positioned the cursor close to the edge of a window, the menu was clipped. We did not want to require the user to always select menus while positioned in the center of the window.

- If a new window was opened, we had to finish all processing in that window and close it in order to return to the previous window. We could not have multiple active windows or switch between overlapping windows.

As the result of several experiments, we decided not to use popup menus, but instead we decided to display all selections as static displays, that is, they would always be on display. Because of the difficulty of getting new windows to work independently, and the confusion that a designer might have in trying to differentiate between windows used for different functions, we decided to have one large window divided into three fixed areas with specific options always displayed in the same area of the screen. The three main areas are paper, actions and messages. There are, in addition, two subwindows for glyph-savingfunctions and toolkit operations. These subwindows are selected, used, and closed before other processing resumes.

We attempted to test different possibilities for an interactive dialogue with the system in our initial design phase, but we soon realized that our rate of progress was too slow to ever get beyond testing interactive ideas to actually developing a working design system. At that point, we decided to select an interface based on our original ideas, develop it and try it out. The details that we were worrying about were not our major research considerations; we were not developing the system to find out whether font designers preferred popup-menus or static ones but to discover whether it was possible to translate the design process as a whole to a computer medium. The motivating factor behind the methods we decided to use was that of minimizing confusion by minimizing the number of decisions a designer had to make that did not have to do with the actual design process.

One of our design decisions was to use only two mouse buttons although the mouse we were using had three buttons. After analyzing the design process, we decided that the user needed to add things to the screen or take them away. If we could have fit a third button into this scheme gracefully, we would have done so, but every time we tried to fit a function in with the third button, we discovered that it was easier and more understandable to fit the function into the two button metaphor. The result of this decision is that we have a few more functions than originally planned, but selection of objects is facilitated since the user only needs to remember that one button adds an item to the screen and the other deletes it.

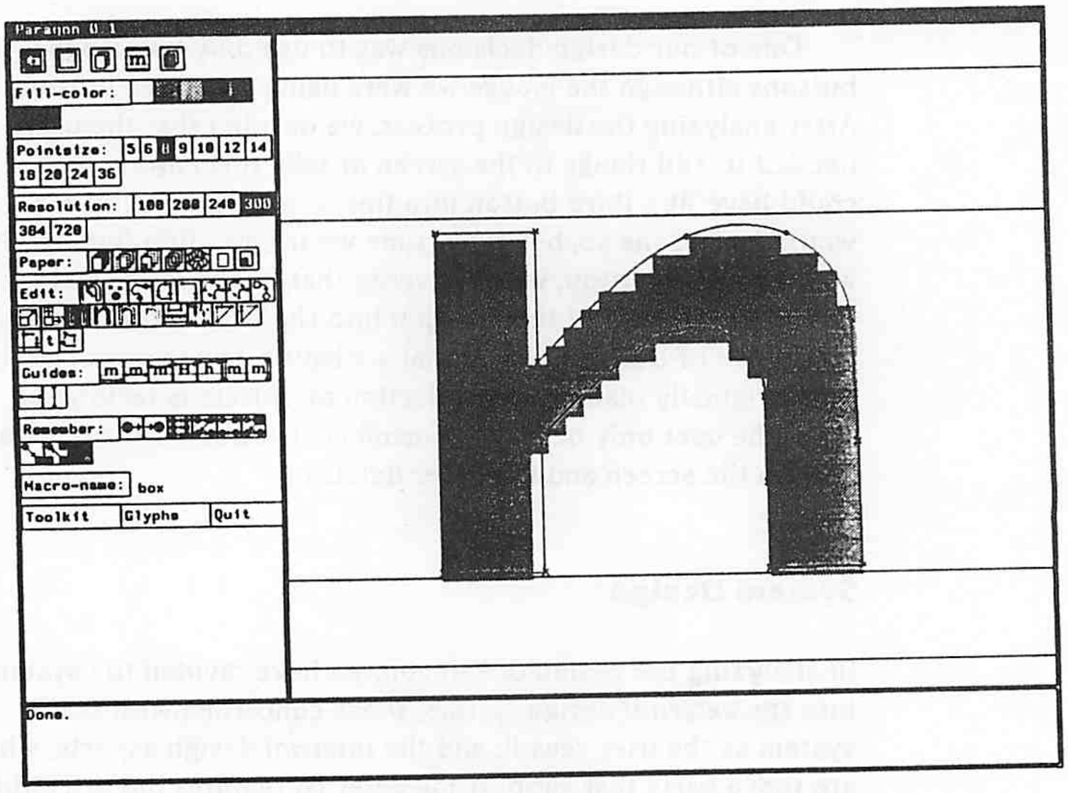
System Design

In analyzing the design of Paragon, we have divided the system into the *external* design factors, those concerned with the system as the user sees it, and the *internal* design aspects, which are those parts that support the external features but are hidden from the user. We will discuss each of these parts separately.

External Organization

The layout of the screen display of Paragon has been divided into three basic windows or panes (Figure 42). Each window has specific information that is displayed in it. In addition to the basic windows, there are two overlay windows used for display of information that is used less often. The three basic windows initially display *actions*, used to select different actions to be performed or objects to operate on; *paper*, on which all drawing and designing is done; and *messages* that are either system messages or help information. The overlay windows are used to display *database* functions such as commands used to select typeface, font, character or glyph information and the *customization toolkit*, used to add, delete, or modify primitive or user-defined actions.

When implementation of Paragon was begun, the original intention was to have all information in the system displayed on the screen at the same time, thus precluding the possibility that a designer would be confused by different overlapping



Done.

Figure 42. Screen layout of Paragon.

windows. Unfortunately, it became apparent that there was too much information to display it all on the screen at the same time. Use of overlays provides a coherent display to the designer in that the layout of the screen and the size of the individual panes remains constant although the contents of the windows change. Because the functions in each display have been chosen to be mutually exclusive, there is no need to switch back and forth between the overlay and the display that it hides. If the overlay is selected for display, any options in the basic window that have been selected are remembered when the overlay is closed and processing is resumed from the original window.

The Option Windows
The actions window, the glyphs window, and the toolkit window all contain icons or labels used to invoke actions or select objects within Paragon. To implement these *option* windows, we had to build our own display mechanism rather than using the

option window package provided within the Sunwindow system. Option windows provide the user with lists of options from which to choose. The Sun package provided routines to display the lists in a fixed format, specify the contents of the lists, and define the function each operation invoked. Unfortunately, all this information had to be specified at compile time which did not allow new functions to be added to the lists interactively. In order to add functions to the system, we needed the ability to add new functions to these lists, or modify the appearance of an existing function (the icon or name representing the function) interactively, thus precluding our use of the Sunwindow option package.

We chose instead to implement our own display and selection mechanism. To provide sufficient flexibility, we read in all the data displayed in our options windows from an external *setup file*. This stricture allows us to control the addition or removal of information from these windows while the system is in operation by changing the data saved in the database related to window display and redisplaying its contents. The versatility that this gives us allows new features or operations to be created and included in the database without reconfiguration or recompilation of the entire system. Operations that can be performed on the data include deletion, addition, or modification. If the content of the database is changed, the screen display will change to reflect these modifications when a refresh of the display is requested. When the system is exited, any changes are saved in a new version of the setup file.

We have divided the setup file into five different categories of *presentation objects*. Each of these objects provides a different, though not exclusive, means of specifying an action. These presentation objects include:

- *switch items* – switches feature on or off

- *function items* – selects one function out of a list of either text objects or icons

- *value items* – selects one value out of a list of either text objects or icons

- *menu items* – selected operation is a function that is executed immediately

- *labels* – accepts keyboard input and is used to identify objects

Within the setup file, each object is identified by a type, a label, and a list of actions, each of which is identified by a name, an operation code (or op-code), and an optional icon bitmap. The icon is used to represent the item in the display window, but if it is not present, the function's name is displayed. Within the screen display, selection of an operation is achieved through clicking of the left mouse button while the cursor is positioned within the boundaries of the icon or label after which the selection is highlighted. Data for switch items includes two bitmaps, one for the *on* state, and one for the *off* state. Selection of a switch icon changes the state of the feature and results in the reverse icon (the one corresponding to the new state) being displayed.

Selection of one of the functions in the actions window determines what action will take place in the paper window. Available functions are either *passive*, that is requiring some input from the designer before the action is performed, or *active*, ones that are performed immediately when the action is selected. Passive functions include ones to place and position points, connect points, draw outlines or filled shapes, and align points; active functions include actions such as erasing a sheet of paper, cycling through a stack of paper, or displaying guidelines.

Any function, value, or menu item can be displayed as either an icon or text. Unfortunately, due to the limited size of Paragon's windows, display of all items using their names takes up too much space, so that all of the information will not fit in the window. As there is no means of enlarging the window, the icon form is currently used for most functions. Labels, however, must be displayed as text names since icons are too obscure for things like file names, and switch items must be icons since both the on and the off state must fit in the same screen space.

Positioning of presentation objects in a window is determined by their position in the setup file, modulo a little bit of space optimization. For example, each list of function items and value items appears on a separate row in the window (or rows if the number of individual items is too long to fit on one line). The first switch item is placed on a separate row from the item that appeared just before it in the setup file. Subsequent switch items, however, are positioned in the same row as the first switch item until the row is filled up. The next switch item is then positioned according to its location in the setup file, with subsequent switch items again positioned in the same row until the row fills up. Because icons are relatively narrow compared to the window width, we could easily run out of room in the window if switch items were each given a separate row in the window. We have tried, therefore, to compact the placement of these objects since we could easily do so.

In the basic system, the actions window contains functions to manipulate sheets of paper, to edit outline sketches, to add guidelines, and to remember a series of commands (Figure 43). It also contains switch items to turn the help function on or off, to display or remove grid or guide lines, and to change the characteristics of the top sheet of paper from transparent to opaque. Value items are provided for selection of a fill color, a point size or a resolution size. (For a detailed list of these functions, see Appendix B.)

To change the display of objects in the *actions window*, we need only change the contents or ordering of the objects in the database. These changes can be done through the use of the *customization toolkit*, invoked through selection of a menu item that results in a customization overlay being displayed in the paper window. This overlay provides access to functions that can be used to modify the behavior or appearance of any of the items in the action window.

The *toolkit* contains operations to add, insert, modify, delete, reorder or select an existing function, in addition to an icon editor. It permits a designer to make changes to an item without affecting the original item, then allows these changes to be saved and installed once they are finished. This installation can later be revoked if the changes are not satisfactory. An existing item can be modified by adding or changing its icon representation,

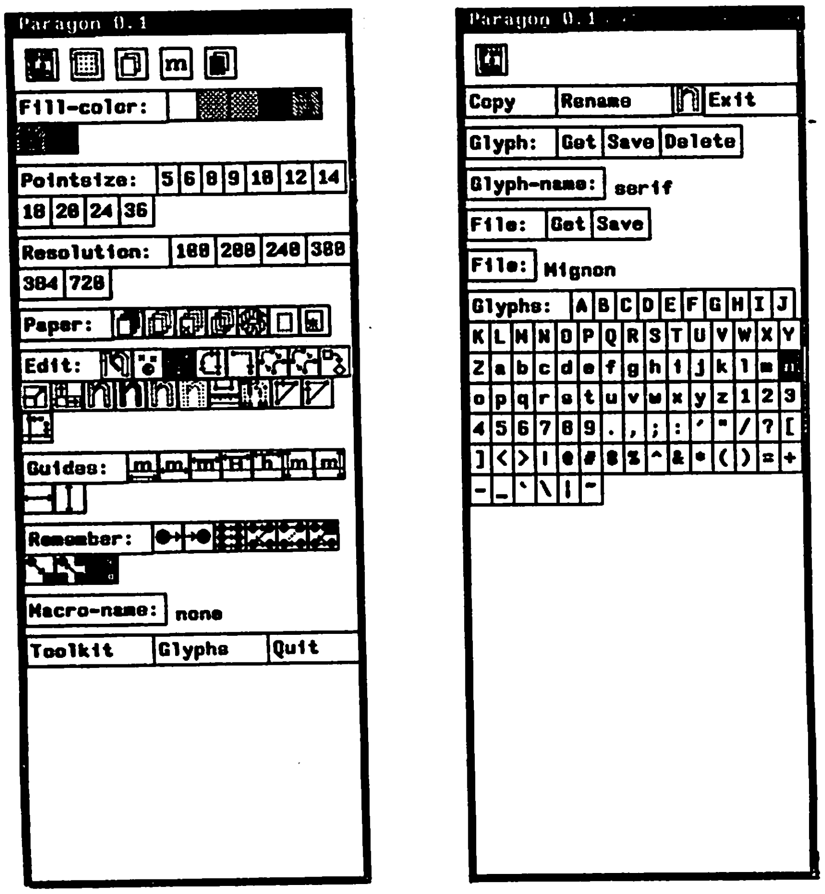

Paragon 0.1
Fill-color:
Pointsize: 5 6 8 9 10 12 14 18 20 24 36
Resolution: 100 200 240 300 384 720
Paper:
Edit:
Guides:
Remember:
Macro-name: none
Toolkit
Glyphs
Quit
Paragon 0.1
Copy
Rename
Exit
Glyph: Get Save Delete
Glyph-name: serif
File: Get Save
File: Mignon
Glyphs: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3 4 5 6 7 8 9 . , ; : ' " / ? [ ] < > ! @ # $ % ^ & * ( ) = + - _ ` \ | ~

Figure 43. Functions provided in the Actions (left) and Glyphs (right) windows.

changing its name, or changing the function that it invokes if it is a user-defined function (a non-primitive).

The contents of the *glyphs window* are controlled by the glyphs that have been saved in the typeface file. The window contains as a default the names of all single characters present on the keyboard, a list that includes letters, digits, and punctuation. When a typeface file is read in, characters that are in the file are highlighted so that the designer knows which of the characters listed are actually present in the file. If the designer saves new glyphs, he can specify their names either by selecting them from the default list, or by typing in a name. If a new name is typed in, the name is then added to the character list. In addition to items used to select glyph names, the window contains functions to save, retrieve or delete glyphs, to save or retrieve files, and to specify file names.

The Paper Window

The paper window is used for creation or modification of character designs. Operations that take place in this window include drawing, editing, copying, rotation, reflection, scaling, filling, smoothing, and specification of guide or grid lines. Selection of these operations is made in the actions window. Each of these operations is discussed in more detail later.

Selection of a function in the actions window may result in a specialized cursor being displayed when the cursor is within the boundaries of the paper window. These special cursors are used as an aid to help the designer remember which function has been selected. Different cursors include ones for drawing (a small pencil), knot placement (a knot cursor), measurement (a ruler), placement of guidelines (crosshairs), and selection of an object (a highlighted knot). If deletion or removal of an object is selected by pressing the middle mouse button, the cursor displayed is the same as the standard one for that operation but it is surrounded by a box drawn with dashed lines.

The Message Window

While a designer is using Paragon, the message window provides informational messages about the state of the system. These include system messages or prompts to the user. In a few cases, such as when an outline is being calculated, the system may not respond to user commands. In these cases, a message describing what is causing the delay will be displayed.

Help information is invoked by selection of a switch icon representing a user manual. When the function is on, the icon displays an open book; when it is off, the book is closed. If the function is on, selection of any other function results in a help message being displayed in the message window. Messages will continue to be displayed until the function is turned off. If a new function is defined by the designer, she may add a help message for the function.

Internal Organization
Paragon consists of eight modules that largely parallel the different windows presented to the user. The paper, actions, messages, glyph, icon and toolkit modules contain subroutines that control responses to mouse selections or movement within the appropriate window.

- The paper module subroutines control the placement, positioning, addition or deletion of objects on sheets of paper. These functions include drawing, selecting sheets of paper, positioning of knots, and placement of guidelines.

- The actions module contains subroutines to set up and display function options in the actions window and sets up the action to be performed when a mouse selection is made in the paper window: it changes the cursor, saves the op-code to be executed when next selection is made in the paper window, and sets up a mechanism to accept the correct number of arguments to the op-code. For example, if the designer selects the function *move-segment*, the cursor is changed to a bulls-eye (indicating that a point is to be selected), the op-code for moving a segment is saved, and the number of parameters is set (in this case, two: a point on the source segment and a destination point.) When a selection is made in the paper window, the operation corresponding to the op-code is invoked, and the appropriate parameters are accepted. In the case of our example, where the first selection is supposed to be an existing point, the system will wait and prompt for an appropriate first selection before accepting the second selection.

- The help module displays system help messages or prompts in the message window. If the user selects the help function, help messages will be displayed with each selection of a function icon until the help function is switched off.

- Subroutines in the glyphs and toolkit modules handle the display of the glyphs overlay and the toolkit overlay, respectively. They also handle mouse selections within the window while the selected overlay is displayed. The glyphs subroutine also contains functions to save glyphs in external files. The toolkit subroutines include functions to add new macros to the system.

- The curves module contains all the curve drawing or scanning routines. These are basically the same routines that are used in the Metafont program although they have undergone minor changes to adapt them to our system.

- The icon module contains the icon editor, selectable from the toolkit window. This was adapted from an early icon editor provided with the Sunwindow package.

Within Paragon, there are three separate databases containing non-overlapping information. The *display* database contains data used to display information in Paragon's option windows. The *glyph* database contains data associated with typeface and glyph information. The *library* contains information about user-defined macros.

Display Database

Information stored in the display database is used to modify or update screen displays. It is initialized from data read in from the setup file during system initialization. There is one set of data saved for each of the windows that uses a setup file to initialize its screen display: the glyphs window, the actions window and the toolkit window. The database consists of a hierarchy of *nodes* and *node-lists*. Each group of functions is a node-list. There are five types of node-lists, each corresponding to one of the five types of setup objects. For each node-list, we save node-header information consisting of: the label of the list (e.g. Paper), the type of list (setup type), the number of nodes in the list, a pointer to the visible node (the one that is highlighted), the display type (whether it is displayed as icon or text), its op-code(this tells us what action to take if the node is selected), and pointers to the next and previous node-lists.

By keeping all the node lists connected through pointers, it is very easy to trace through all the lists and find which node has been selected when there is a mouse selection or to redisplay the window if an update has been made to any of the lists.

Each function displayed on the screen is stored as a node. For each node, we save information about its screen position and display (name or icon and dimensions of each), its visibility (whether it is highlighted), information about what to do if it is selected and data related to its position within the node-list hierarchy.

Glyph Database

Information for glyph data is stored in an internal database which is not accessible by the user. Its contents include:

- Typeface information: name, pointsize, resolution, number of glyphs
- Glyph lists: name, status (saved, deleted, modified), sidebearings, list of knots, parents (glyphs included in this glyph), children (glyphs of which this is a part)
- Knot list (information from external file): position (x,y), type (curve, straight), guideline attachments
- Point list (used to keep track of screen info): position (x,y), pointers to various connecting knots such as next point in point list, next point in segment, next point on this sheet of paper, guideline attachments
- Guidelines: position on screen, display information (whether on or off), list of attached points

As it turned out, keeping track of the information in the database was the largest problem that we faced. Initially, we made some decisions about what information we needed to include in the database. As we used the system, however, we determined that some of the information that we included was not needed, while other information was, so the database grew and shrunk, mostly the former. For example, in saving knot data, we initially chose to allocate a large array of knots, connected by two linked lists, one for used knots, one for available knots. Each knot contained a pointer to the knot before and after it in the list. In addition, if the knot was connected to other knots in the screen display, we needed to keep track of that information, and again in both a forward and backward direction due to the curve drawing algorithm that we chose to use. When we added functions to handle different sheets of paper, we realized that we needed to add information about which knots were on which sheet of paper. If a sheet of paper was deleted, the knots on it had to be deleted as well and returned to the available knot list. In providing the function to attach knots to guidelines, we determined that we needed to keep a list of which knots were connected to which guideline. All of this information was kept in one list with many pointers so that we would not have to update multiple lists if a knot was deleted.

Information that we decided that we did not need was usually so embedded in the database that it was more difficult to remove than to leave in so it was usually left in. For example, after watching designers use the system, we noticed that they never positioned unconnected knots on the paper. Each knot on the display was always connected to at least one other knot. Had we realized this earlier in the design stages, we would have revised the knot-list model so that it was based on segment lists rather than individual knot lists. It is not clear that the database would have been any smaller, but it might have been easier to manage.

Library Information

The library database consists of two types of information. The first is a stack containing macro functions that have been saved and named but not yet added to the system; the second is storage for user-defined functions that have already been added. For each macro, we needed to save information about the functions it is composed of including each function's op-code, number of parameters, next function in the list and previous function in the list. For macros that have been added to the system, we had to save additional node information used for displaying the function in the actions window.

External Databases
There are three external sources of data for the system. These have already been discussed briefly. First there are *glyph files* containing typeface, font and glyph information. These files are read and written by the system at the user's request. *Library files* contain information about user-defined macros. These files provide the system with information on the appropriate actions to take when the function is invoked. The final type of external data is the *setup file* which has already been discussed. It is used to initialize the display of function names and icons in the actions, glyphs and toolkit windows.

System Operation

A description of the architecture of Paragon gives little idea of how a designer actually interacts with the system to create type designs. We will attempt to remedy that omission by providing a detailed example of how a designer uses the system, followed by an evaluation of some of the decisions that were made during the implementation process.

At startup, the screen display presents the user with three windows: the actions window, the paper window, and the message window. The actions window provides functions that are used to draw, edit, and save glyphs drawn in the paper window. The message window displays system messages.

The actions window contains icons for functions used to manipulate objects in the paper window. Functions are implemented in a *prefix* notation; the arguments to the action follow the operator selection. We chose to use a prefix style because it seemed more natural to designers to specify the operation that they wanted to perform, then the objects on which to perform it. This means of specification worked out well within the model of our system as it allowed us to implement a *mode-less* interaction paradigm. When a function was selected, we saved the information we needed to invoke it without actually invoking the function, and thus going into a *mode* where the system was expecting certain input and that had to be exited before regular processing could be resumed.

Functions that take no arguments are called *active functions* since they are active as soon as they are selected. Functions that take arguments are called *passive functions* since they must wait for user input before any operation is performed. If an active function is selected, an action will be executed immediately, in which case the designer sees the result of the action right away. For example, if the designer selects the function to save a glyph, the glyph information is saved immediately. If he selects a new piece of paper, the new sheet of paper is displayed on top of the stack.

Selection of a passive function sets up the operation that will be performed when the user selects the arguments to the function. For example, if the designer selects the action to add a knot, nothing happens until the user clicks the mouse in the paper window, at which time a knot is added to the display. To change the operation that is executed in the paper window, the user simply selects a new function. A selected function does not need to be executed; if the user changes his mind about which function to perform, he simply selects a different function.

The actions window also contains *value* items and *switch* selections. Selection of a value item sets a value that may be used with an action selection. Fill patterns, for example, are represented as a list of value items. Selection of one of the fill patterns establishes the pattern that will be used the next time the fill operation is invoked. Switch selections are used to turn on or off certain display features such as guidelines, gridlines, and the help function.

Within the paper window, the left mouse button is used for selection of an object and the middle button is used for deletion of an object. The right button is not used. Mouse action may require the user to *press* a button in which case it should be held down while further actions are performed, or that she *click* the button, in which case she should press and release it.

Drawing

At system start-up, the drawing function is enabled, a state that is indicated to the designer by a small pencil cursor that appears when the cursor is positioned in the paper window. The designer can sketch shapes by holding the left mouse button down and dragging the cursor. A one pixel wide pen is drawn as long as the button is held down. Lines can be erased by holding down the middle mouse button and dragging the cursor along the line to be erased. The eraser is 8 pixels wide (the width of the cursor) so erasure is not as precise as drawing. This feature is provided since it is difficult to erase a one pixel wide line with a one pixel wide eraser except by multiple passes of the eraser. The characteristic of the eraser being wider than the drawing point, incidentally, mimics the features of a real pencil where the eraser is wider than the graphite point. To erase everything in the paper window, there is a function in the *paper* list called *erase-page.*

Once a shape is drawn, the designer needs to position control points (or knots) along its edges. He may select one of three functions for adding knots: adding individual straight-edge knots, adding individual curve-edge knots, or adding connected (straight-edge) knots. Since all shapes must be drawn with connected lines before they can be digitized, it is easiest to start by selecting and using the *connect-knot* function. At each press of the left mouse button, a knot is added to the current sheet of paper. Each knot is connected to the previous knot. If an existing knot is selected, that knot is added to the connected list and no new knot is created. Each knot can be connected to at most two other knots. If a fully connected knot is selected, that is, one that already has its full compliment of connections, an error message is displayed and no action is taken. To create disconnected lists, re-selection of the connect-knot function will start a new list. Also, selection of the first knot of a connected list will close the polygon, finish off the list, and uncouple the selection process; the next knot selected will not be connected to the previous knot.

The two other add-knot functions will be used less often, but are useful in that they can be used to change an existing knot to that of the selection type. For example, the user might draw several connected knots, then use the curve-knot function to convert some of the knots into curve-edge knots.

The connect-knots function connects all knots by straight lines. Once the user has drawn a series of connected knots, she can select a function to draw a smooth curve through the knots (Figure 44). If there are no curve knots in the list, all the knots will be connected by straight lines, but if there are curve knots, the resulting shape will generally conform to the following conventions:

- all curve knots will have a curve passing through them

- two straight knots will be connected by a straight line

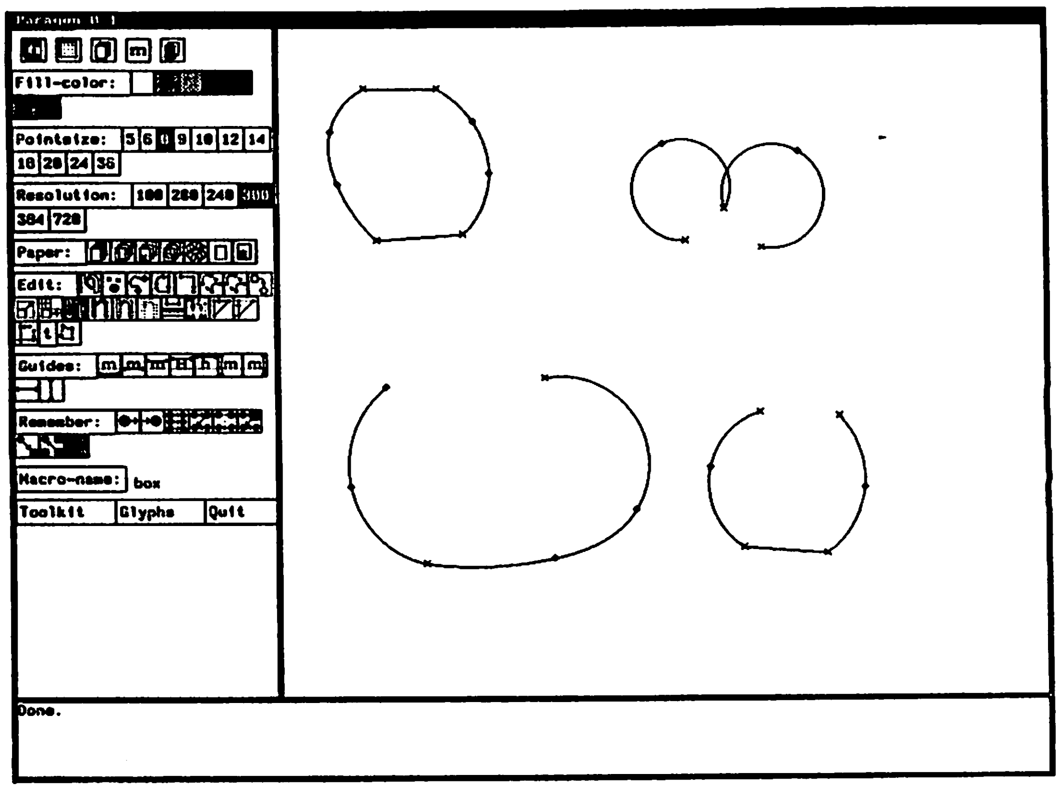
- a straight knot between two curve knots may have zero-continuity (that is, the curve may or may not break at that knot)

- to create a knot that is tangent to a curve (for example, the join of a curve and a straight line with first-order continuity), the point of tangency must be a straight knot (but this does not guarantee first-order continuity).

Because there are many possibilities, experimentation with the different knots to get a feel for how the curve and straight knots interact, will provide a better model for the user than a description of the curve characteristics.

When drawing many shapes, a designer will want to standardize measurements through the use of guidelines. Selection of one of the guideline functions changes the paper cursor to a *guide marker*. Specific guidelines are those that are displayed with a letter in their icon and include depth, baseline, x-height, cap-height and height. They can be added or moved, but can only appear once per window (you cannot have two x-heights, for example). General guidelines (either horizontal or vertical) are also available and are indicated by a guideline icon without a positioning letter. These guidelines can appear more than once.

Display of guidelines is controlled by a switch selection. This switch simply erases or displays the guidelines on the current sheet of paper; it does not erase their values. To erase or change a guideline, the user must explicitly delete it using the middle mouse button when the guideline function is selected.


Fill-color:
Pointsize: 5 6 8 9 10 12 14 18 20 24 36
Resolution: 100 200 240 300 384 720
Paper:
Edit:
Guides:
Remember:
Macro-name: box
Toolkit
Glyphs
Quit

Done.

Figure 44. Outline shapes with curve and straight knots: mixture of straight and curve knots (top left); straight knot between two curve knots with zero-continuity (top right); straight and curve knots showing tangency at the point of intersection (bottom left); straight and curve knots showing no tangency at the point of intersection (bottom right).

Filling Shapes

If the user has drawn a closed polygon, the polygon can be filled using the *fill-segment* function. To fill a closed shape, the user indicates which polygon he would like to fill by selecting one of its knots. It will then be filled with a solid black color pattern. To change the pattern used to fill a polygon, the designer must select one of the available fill patterns before selecting the polygon. When the polygon is filled, the selected fill pattern will be used to fill the character. If two overlapping polygons are filled with different patterns, the patterns will overlap, thus the designer will see three patterns: the pattern for the first object, the pattern for the second object and the intersection of the two patterns where the figures overlap.

The polygon can also be filled with a raster calculated for a specific size and resolution. To do this, the user selects a point size and a resolution and indicates the depth and height of the glyph by positioning the depth and height guidelines. When the fill routine is invoked, the polygon will be filled with a bitmap calculated to fit those specifications (Figure 45). For example, if the user selects a grey fill pattern, a point size of 8 and a resolution of 300 dpi, and positions the appropriate guidelines, Paragon will generate and display a grey-filled raster pattern equivalent to the dot pattern that would be generated if the character were being digitized at that size (details of this operation were given in the previous chapter). The raster fill pattern will only be calculated and displayed if all the needed components (the point size, the resolution, the depth and height guidelines) are selected. Thus to revert to a solid fill, the user should remove or clear one of these values. The point size or resolution can be cleared by clicking the right mouse button in the function label; in the case of the point size, the user would select the label *pointsize* rather than one of the numerical values.

Use of the bitmap-fill routine along with the availability of different fill patterns gives a designer the ability to draw an outline then fill it with different raster patterns calculated for different resolutions or point sizes. The resulting display allows the designer to see where two bitmaps differ and perhaps where the outline shape should be modified to generate better bitmaps. The designer might draw a lower case 'n', specify a resolution of 300 dpi, a point size of 10 points, and a fill pattern of 25% grey, and fill the character. She then changes the point size to 12 points, selects a right-diagonal fill, and selects the fill function again. A different bitmap is displayed on top of the first one, but the differences between the two can be seen where the two fill patterns do or do not overlap (Figure 46).

Once a shape has been drawn, it can be modified through moving of points and recalculation of the curves connecting the points. Functions are provided to move a single point, *move-point*, by dragging the point while the right mouse button is pressed, or moving a connected segment or series of points, *move-segment*, by dragging one of the points in the segment. A

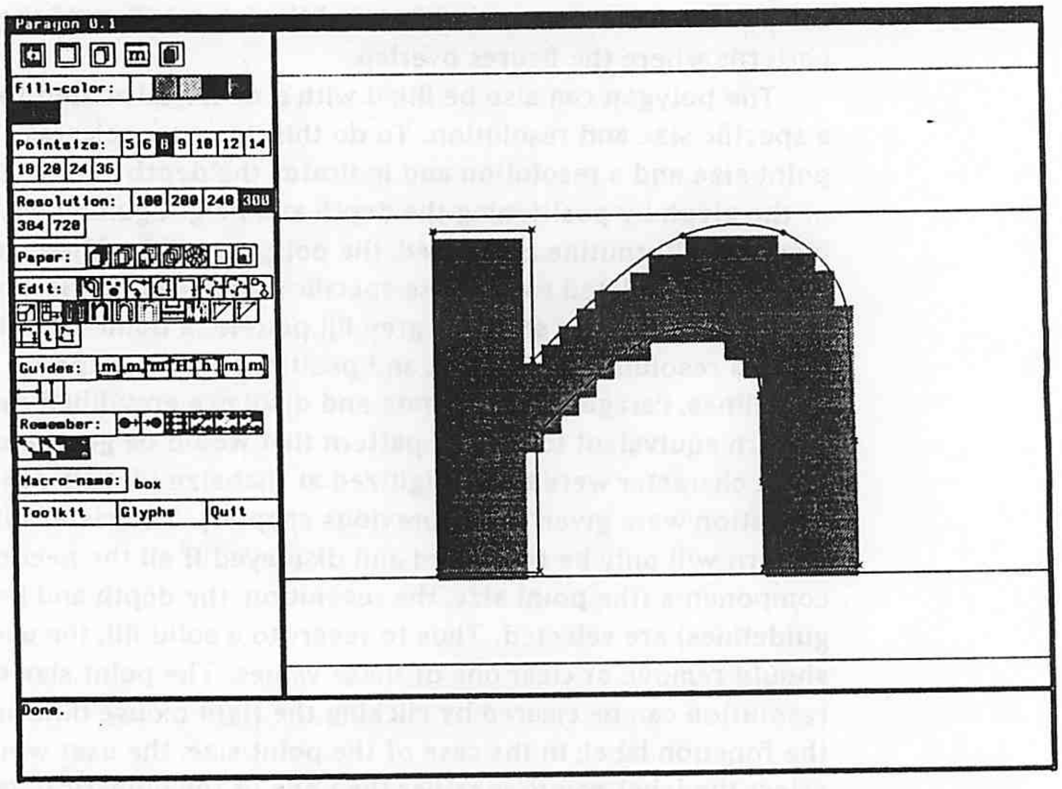


Figure 45. Outline and bitmap calculated for an 8 point, 300 dpi character.

segment can also be copied, an action that is performed in the same way as moving a segment, except that, when copying, the original segment remains in place.

Additional Functions

Two functions have been provided to assist the designer in aligning points either with each other or with a straight edge. The first function, which is used to align two points, has two variations: a *horizontal alignment* and *vertical alignment.* If the designer selects one of these functions, then selects two points on the screen, the first point is aligned with the second point in either the x- or y-coordinate. This alignment is not permanent; if one of the points is subsequently moved, the second point will not move with it.

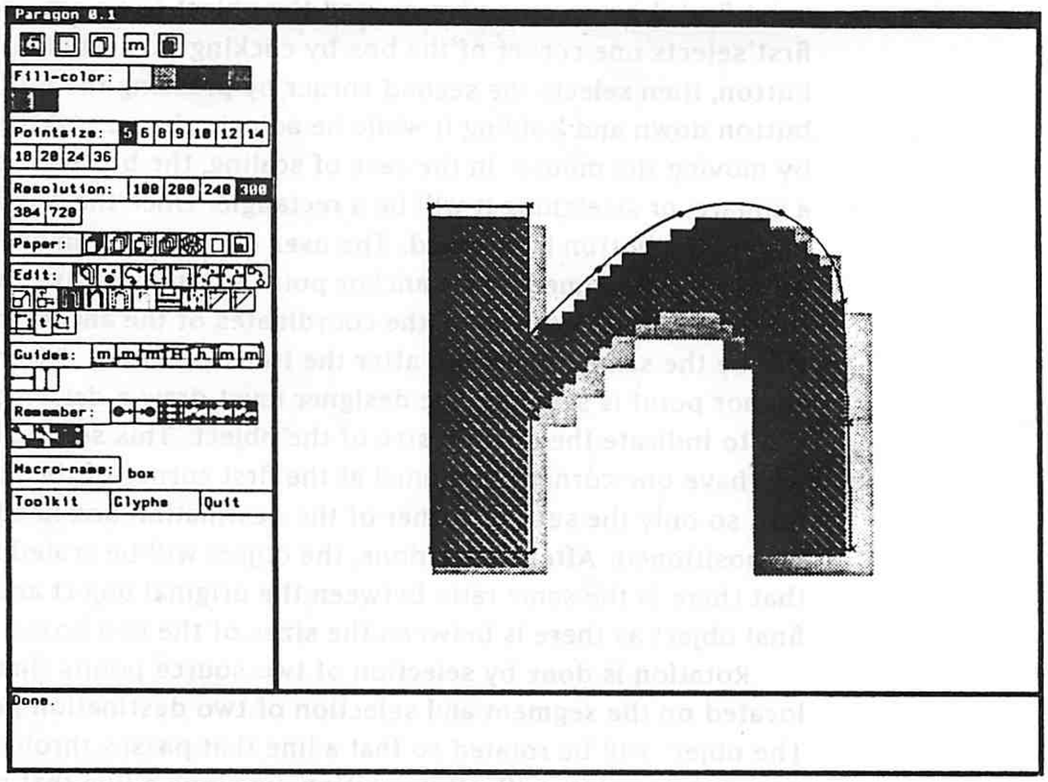


Figure 46. 5 point and 12 point character bitmaps calculated from same outline.

The second alignment function provides the means of *attaching* a point to a guideline after which the point will move whenever the guideline is moved. This function provides a way to specify a set of points that will always remain aligned with each other. To attach the point, the user selects the function; then the point is moved so that it lies on top of the guideline. Each point can be attached to at most one vertical and one horizontal guideline. If the point is positioned on the intersection of two guidelines, it is attached to both of the guidelines.

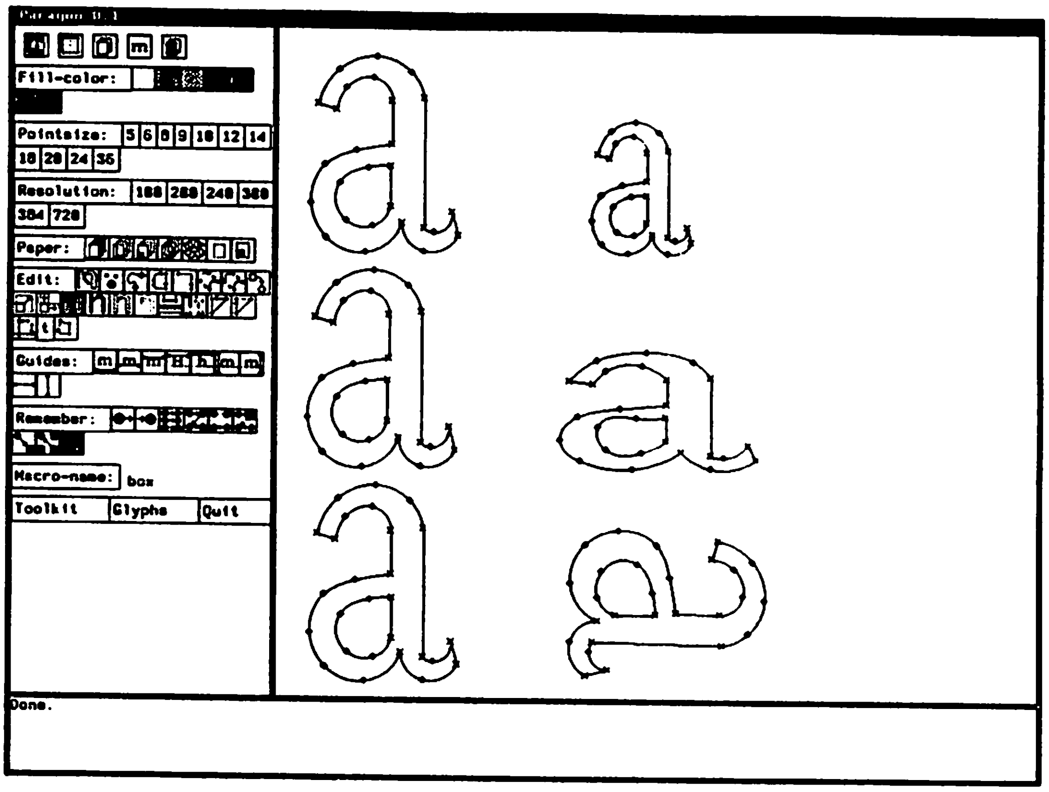
Three transformation functions are provided for segments: they can be *scaled*, in which case the x and y proportions remain the same; they can be *stretched*, where the x and y dimensions may be scaled differently, or they can be *rotated* through the x-y plane (Figure 47). To scale or stretch a segment, the designer must first draw a *source box* around the object to be scaled. He first selects one corner of the box by clicking the left mouse button, then selects the second corner by pressing the left button down and holding it while he adjusts the size of the box by moving the mouse. In the case of scaling, the box will remain a square; or stretching it will be a rectangle. Once the box is drawn, the button is released. The user then selects an anchor point on the segment. This anchor point will remain fixed when the object is scaled; that is, the coordinates of the anchor point will be the same before and after the transformation. After the anchor point is selected, the designer must draw a *destination box* to indicate the desired size of the object. This second box will have one corner positioned at the first corner of the source box, so only the second corner of the destination box needs to be positioned. After this is done, the object will be scaled so that there is the same ratio between the original object and the final object as there is between the sizes of the two boxes.

Rotation is done by selection of two source points that are located on the segment and selection of two destination points. The object will be rotated so that a line that passes through the two source points will, after rotation, lie along a line that passes through the destination points. The first source point and the first destination point will be matched up which may result in the object also being moved if the two points do not have the same coordinates.

Saving a Glyph

Once a character has been drawn and edited, it can be saved. Since there may be more lines or segments on the screen than the designer may want to save, the desired segments must first be *highlighted*. This function changes the smooth line drawn by the outline routine into a dashed line so the user can see which segments have been selected. Once the desired segments have been highlighted, the user selects the *glyph* menu option which pops up the glyph window on top of the actions window.

Within the glyph window there are functions to name, select, save, or delete glyphs, and to name and save files. The window also contains the highlight function so if the user changes her mind or neglects to specify the desired segments before opening


Fill-color:
Pointsize: 5 6 8 9 10 12 14 18 20 24 36
Resolution: 100 200 240 300 384 720
Paper:
Edit:
Guides:
Remember:
Macro-name: box
Toolkit
Glyphs
Quit

Done.

Figure 47. Scaled (top), stretched (middle) and rotated segments (bottom).

the overlay, she need not switch back to the action window to do so.

To save a glyph, the user must first either select a glyph name from the available list, or select *glyph-name* and type in a name (with no embedded blanks) and terminate with a *carriage-return*. Once a name has been selected, the user chooses *save-glyph* and the glyph is saved in memory. If she is adding a new glyph, the glyph name is added to the glyph list in the window. If the user also wants to save the information in a file, she should select *file-name* and type in a file name, (again with no embedded blanks and terminated with a carriage-return), then select *save-file.* Any glyphs that have been saved will be written out to the file. It should be noted that saving of a glyph or a file will overwrite any previous file or glyph with the same name. If a file already exists and the designer wants to add a glyph to it, she should first read the file into memory with *read-file*, save the glyph, then save the file.

Toolkit
To add a new function or item to one of the lists, the designer first defines the operation using the facilities provided for remembering a sequence of commands. After selecting the *start-remembering* function, the designer selects and executes the series of actions that he wants to define as a macro. When he has finished, he selects *stop-remembering*. If the *repeat-memory* function is selected, the remembered commands will be repeated. If user input (such as knot placement or selection) is required when the macro is repeated, prompts will be issued in the message window. To save this macro, the user must name it using the *macro-label* function, and save it using the *save-command* function. If a macro is not saved, it is overwritten by the next macro that is defined and thus disappears.

Saved macros are kept in a list. Selection or definition of a macro places it on top of the list. Selection of the repeat command will repeat the macro that is at the top of the list. To move a saved command to the top of the list, the user must enter its name in the macro-label field (terminated by a carriage return), then select *retrieve-function*. Once the function is at the top of the list, it can be executed by selecting the repeat-memorycommand.

For example, if the designer defines a macro and then selects repeat-command,Paragon will execute the macro that was just defined. The designer then names the macro *draw-box* and saves it after which he defines a second macro, names it *draw-circle* and saves it. Selection of the repeat-command function at this point will result in the execution of the draw-circle macro. To repeat the draw-box macro, the user must enter the macro name in the macro-label field, select retrieve-command, then select repeat-command.

These macro definitions will last only until Paragon is exited. To make the macros permanent, the user must select the *toolkit* option in the menu so that the toolkit window is displayed. This window contains functions to add, select, delete, or edit macros from the permanent repertoire of the system. To add a new macro, the user selects the *add-function* option. He is asked to select the list he wants to add the macro to and is then prompted for the macro name. He may create an icon for the macro by selection of the *icon-edit* function but if he does not, the macro's name will be used as a label. Once he has specified all this information, he selects *do-it*, the macro is added to the actions window and the screen display is updated so that the icon or label is displayed. At this point, the icon can be selected and the function will be executed in the same way as other functions. When the program is exited, all new macros will be saved in an external file and will reappear whenever the program is restarted up again.

Other functions in the toolkit window allow the designer to change the name or icon for any of the existing functions. He first selects the list containing the object he wants to change, then selects the item to be changed in the list. If he is changing the name, he then selects *label-name* and types in a new name; if he is changing the icon, he selects *icon-edit* and the icon editor will appear with the current icon displayed in it. After changing the icon, the user must save it and exit the editor. He then selects *do-it* and the icon is changed in the actions window.

Design Decisions

In designing Paragon and in choosing which functions to assign to specific devices, many decisions had to be made. In addition to determining which functions we wanted to provide for the user, we had to decide how to implement those functions. Some of the decisions that we made worked well within our design paradigm, others were not so successful. We discuss here some of the decisions, how they were implemented, and how well the implementation worked.

User Input

Available means of input to the system were a keyboard and a mouse. The keyboard was used to enter file names, glyph labels, and macro labels. The mouse was used for selection of items in the actions window and for manipulation of objects in the paper window. It had 3 buttons and provided feedback when buttons were both depressed and released, a feature we were able to take advantage of by programming the system to perform certain functions when a mouse button was pressed and others when it was released.

Mouse Buttons

One decision we had to make was that of allocating mouse buttons to different functions. We wanted to maintain consistency between operations and, at the same time, get the most selectivity out of the buttons. Because we were developing an interactive system for non-computer users, we decided to emphasize consistency over enhanced selectivity, a decision that was to prove only moderately successful. We decided to use the left button to select or position objects and the middle button to delete objects or previous selections. This means of selection worked successfully for delineating individual items but did not give us the means of selecting different objects, such as segments. Our solution was to provide a separate function for manipulating segments which required that the user indicate a knot on the segment with the select button (the left one).

A more successful model might have been to use one button for selection of points, one for selection of segments, and one for selection of regions, with some other means used to delete items. We could have also used a variation of a model that was used with the Xerox Star system [SMIT82] so that multiple button clicks selected larger items; one click would select a knot, two a segment, and three a region, though we would still have needed a means of delineating the region.

Function Selection

In addition to maintaining consistency with each selection, we wanted to maintain simplicity by providing as much functionality out of a selection as possible. Rather than requiring a designer to spend time setting things up, we tried to supply complex operations that used fewer selections. As a result, we implemented actions as higher level functions even though this gave us less flexibility in the final system. To be specific, rather than giving the user a choice between items (points, segments, regions, sheets of paper) with a corresponding selection of actions (move, delete, copy, add), we implemented complete functions (move a point, move a segment, move a region, add a point, etc.). This allowed the designer to make one selection to perform an action. The alternative was for the designer to make two selections, one for the action and one for the item. This alternative might have made the customization process easier, but would have been tedious for the designer when she was working with the system to design characters. We chose to provide higher level functions because we thought a designer would spend more time designing than customizing, and we wanted to facilitate the design process. Again, in retrospect, this was not the best solution in terms of ease of implementation or ease of customization.

Screen Appearance
Another decision we made was to provide only one window and divide it into several different regions rather than popping up different windows that overlapped or providing multiple windows of the same type. The motivation here was to reduce clutter and make it easy for the designer to see what was going on. With only one screen layout, the display remained consistent whenever the system was run. Eventually we discovered that we had too much information and too little space to display it in. When we realized that we needed more screen space, our only solution was to usurp one of the regions temporarily by overlaying its display with new information. When the designer is through with it, the overlay is closed and the region is returned to its previous display. The overlays, however, were self-sufficient in that they did not require input from other windows while they were open.

Another problem we discovered was that we ran out of room in one of the regions (the actions partition), and had to re-evaluate which functions to keep in that area, which to move to another region, and which to eliminate altogether. Even after we created the glyphs overlay to handle database manipulations and thus removed all the glyph operations from the actions window, the actions region was almost full. The size of the window cannot be changed by the designer to allow for more icons.

Part of this lack of space comes from the fact that we added a lot of functions to the system that were not originally envisioned. Although this might have been a result of poor planning, it was instead the result of discovering that there were many useful functions to be provided within a graphics environment that went beyond the designer's original specifications. We discovered, as the system developed, that there were a lot of computer graphics functions that were applicable to this environment. Examples of these added functions are ones to attach knots to guidelines, to align knots, and to rotate and stretch glyphs.

Curve Drawing

Implementation of curve-drawing facilities within Paragon was focused on providing easy-to-manipulate and easy-to-modify shapes. Additional provisions that could be added are the ability to directly control the Bèzier control points by displaying and moving them around on the screen, by allowing specification of tangent directions at any point, or by specifying tension on a particular curve. The ability to perform each of these functions is already built into the curve algorithms that were used, but has not been implemented for the designer's use.

A drawback to the curve-drawing routines that we have chosen to use are that the curves cannot be controlled interactively; all interactions are done on objects made up of straight line segments. To produce a curved edge, the user first draws a polygon made up of knots connected by straight lines, marks the knots that should lie along a curve, and then selects a function that will draw a smooth (and possibly curved) edge through the knots. If no curve knots are marked, the polygon knots will be connected only by straight lines.

There is a tradeoff between having powerful curve-drawing functions and providing immediate feedback to these routines. We chose to support more flexible and powerful curves at the expense of providing immediate feedback. The curve-drawing routines we used were developed for a batch-oriented drawing system, not an interactive one. The advantage to using them is that the curve actually goes through each point the designer selects. Adjustment of the curve can be done by adjusting the points on the curve, then requesting that the curve be redrawn. The use of these routines interactively (with the curve moving with each knot adjustment) was not investigated so it is not known whether it is, in fact, possible to modify the existing algorithms to do sufficiently fast calculations to provide interactive feedback. Curve-drawing in Paragon is not unbearably slow but it is not always easy to predict what a curve will look like. Perhaps with intensive use of the system (and familiarity with the curve drawing functions) a designer will become comfortable with the routines we have provided.

Summary

We have provided functions to allow a designer to draw and edit character shapes, to display these shapes as digital representations, and to add additional commands to the system, all as part of the standard working environment. In the next chapter, we evaluate how well these tools worked and describe modifications to the system that would be necessary to make it into a production tool.

Chapter VI

Performance of the System

The possibilities of investigating new alphabetic forms compatible with the computer printout is exciting. Surely this can be accomplished within the limits of traditional letter proportions and beauty, just as the late fifteenth-century punchcutter broke away from his dependence on pen-drawn manuscript letters. It is for a new generation of type designers to meet the challenges of this ancient craft.

Alexander Lawson, *Printing Types, An Introduction*

The evaluation of Paragon as a type design system focused on three areas, each corresponding to one of the problems that we were attempting to solve. We wanted to know, first, whether a designer, with no prior experience, could use and understand the design paradigm that we developed for Paragon; next, whether we had defined and recreated a designer's traditional tools sufficiently well for the designer to use the system with little training; and, finally, whether we had supplied the designer with a complete set of tools to enable him to work easily. Our second area of evaluation was to see whether the new computer-enhanced tools that we provided enabled a designer to create designs for a digital medium easily and whether these tools were of assistance in the design process. We were also interested in seeing whether the designer's process changed to incorporate these tools so that the design process became more efficient. The third area of evaluation was the toolkit package and whether the designer understood how it worked and used it to create and add new functions to the system.

System Evaluation by Designers

The designers who used Paragon were uniformly enthusiastic about its performance. The features that were most frequently praised were the ability to select and manipulate more than one sheet of paper at a time, the display of raster patterns at different sizes and resolutions, and the customization toolkit both for its provisions for extending the design system and for the potential to use it to create individual instantiations of the system. The functions that allowed a designer to attach points to guidelines and the saving of character parts were also favorably mentioned.

Use of the system by designers was a very informal process. Approximately ten designers used the system to varying extents throughout the design process. Some of them saw demonstrations of the system in its early stages; some only saw it when all development work had been finished. All of the comments received throughout the evaluation process were favorable; most designers had suggestions for improvements or additions and all of them wanted to know when the system would be available for them to use on a permanent basis.

The only difficulty encountered in evaluating the tools we provided was that since designers were not used to having an interactive design system, they were willing to adapt to whatever tools we provided for them. When asked for suggestions as to what could be improved or changed, they were more likely to suggest new functions rather than changes to functions that they had already been given. Suggestions that they did make were incorporated into the system when possible.

Traditional Tools

The metaphor we developed for modeling the traditional tools used by designers seemed to work very well. All of the designers understood how to use the pencil and paper paradigm: how to sketch and erase drawings, select and move control points, add guidelines, or switch to another sheet of paper. They were particularly pleased with the versatility provided by the different sheets of paper and the features that went along with them such as displaying a grid or guidelines on the sheet, or making the sheet opaque or transparent. The ability to maintain different versions of a design or different sketches, thus enabling them to switch from one design to another easily, was very important. Their appreciation of this model was based on its similarity to the traditional environment where they might be working on several different ideas at the same time with all the designs residing on various sheets of paper all piled together on top of their drawing table. As they work on different designs, they shuffle the sheets of paper around, rearranging the stacks. The only part of the model that we did not implement was the ability to maintain several stacks and to switch between stacks, but this idea did not occur to us until after the system was finished.

Although the mouse did not give the designers the control that they were used to with a pencil, use of the mouse for drawing and erasing was not a problem since it turned out that most of the designers abandoned this feature right away. Most designs were created by positioning control points directly and then manipulating the shapes by adjustment of the points. Management of curve shapes was slightly hindered by the fact that the curves could not be changed interactively, but this did not seriously affect the design process. The designer's ability to adapt quickly to the curve drawing technique was facilitated by the algorithm we selected that allowed the designer to position the points that would lie on the curves; as designers became familiar with the curve facilites, they said that they could visualize in their mind the evenutal shape of the curve despite the fact that the points were only connected by straight lines.

There was a little confusion over the use of mouse buttons since some designers were used to using the right and left buttons rather than the left and middle buttons, but this was mostly due to familiarity with other drawing or design systems they had used previously. Some were confused by the fact that the third button did not do anything, despite a message that we displayed in the message window that informed the user of this fact.

New Computer Tools

Designers adapted very quickly to the use of tools that they had never used before, because the tools made sense intuitively. The tools were seen as extensions to existing tools and were treated as such. Use of the function to generate and compare bitmaps for different sizes or resolutions was very popular. It allowed the designers to evaluate a designs for a specific device or font size and was considered to be indispensable for the generation of digital designs. A designer would draw or edit a shape, then compare different point sizes or resolutions to see what the raster looked like as the values changed.

Guidelines were always used although the ability to attach knots to guidelines and move them was not used as much. The usefulness of this function was limited to those points that lay directly on top of one of the lines. Although the designer could add as many lines as necessary, too many lines led to confusion as to which line was being used for which purpose. It would have been helpful to be able to specify a fixed distance away from a line and attach a point to that distance which would have eliminated the need for so many guidelines.

Functions to do graphic manipulations on characters such as scaling, rotation and stretching were not used often but were considered to be invaluable when they were used. The ability to move multiple points while maintaining a consistent relationship between them was considered to be extremely helpful.

Macro Definition Capability

The function of Paragon that stood out the most from other design or drawing systems that had been used by designers prior to their exposure to Paragon was the ability to define new macros and add them to the system interactively. The means of saving a new function was easily understood and used after a preliminary explanation (all the designers who used the system were given an initial demo by the author of the system). It was not used extensively, though, since the operations provided with the initial system were generally sufficient for what the designers wanted to do.

One drawback to the extension package was that it was not general enough—a failing not of the model but of the system designer who did not have enough time to implement all of its capabilities. The toolkit package could easily have been used to add new point sizes or resolutions, or new fill patterns, but since these operations were seen as secondary to the ability to add new executable functions, they were not implemented. One of the designers who used the system wanted to change the resolution values (e.g. 300dpi, 240dpi) to be the names of types of printers (such as Imagen or Laserwriter), so that he could select a printer name or type and have the system generate the correct resolution. Because we did not extend the toolkit to operate on value items, the only way to change the numbers to names was to manually edit the setup file.

Future Enhancements

Extensions or improvements to Paragon fell into three categories. The first type were ones suggested by designers who had used the system, the second type were those that were thought of but not included due to space or time restrictions, and the third type were those that we deliberately chose to exclude since we felt they did not fit within our design model.

Designer Suggestions

In evaluating the actions that we provided with the system, the designers had quite a few suggestions for improvements or additions. Suggestions that were made were:

- Invisible connections: we needed a means of connecting two segments together invisibly so that if one was moved, the other moved with it. This feature would be used in characters that have more than one contour such as the letter 'o' where there is an inside and an outside segment. In retrospect, this feature is so obvious, we were a little embarrassed to admit that we had left it out.

• Moving partial segments: sometimes a designer wanted to move parts of segments (group of points within a segment), but not the entire segment. This can be done within the current system but only by disconnecting the part to be moved, moving it, then reattaching it.

• Font metric information: in addition to providing measurements of objects, it would be nice to store these measurements so that we can maintain a list of design dimensions such as stem widths, x-height, or width of each of the characters in a typeface. These dimensions would change depending on the typeface (for example, a designer might want to store the angle of slant in an italic face, but would not need this value in an upright roman), so the system should allow flexible labeling of various measurements. It should also be possible to convert these values into other units such as inches, ems, millimeters or pixels.

• Match measurement: given an existing measurement, it would be helpful to have the system be able to match it in some consistent way. For example, a designer might want to match absolute measurements (screen coordinates) or pixel values.

• Modifying static fields: currently, static fields such as pointsize, resolution, and fill patterns cannot be modified by the designer. It would be nice (and not much work) to give them the ability to add or delete values from the pointsize, resolution or fill pattern lists.

• Representing glyphs as icons: designers could use the icon editor to create an icon representation of a glyph within the glyph saving window, rather than having to use a name for each glyph. Alternatively, but a bit more work, would be for the system to calculate the bitmap of the character at a very low resolution and size and use the bitmap as an icon in the glyph list.

- Local guidelines: since not all guidelines are used with each character, the ability to specify guidelines local to a particular character or group of characters would lead to less clutter on the screen.

- Diagonal guidelines: the system currently has little direct support for the design of italic or slanted fonts. To facilitate this, we need to provide a function that would allow positioning of diagonal guidelines.

- Measurement of angles: we should provide a protractor for measurement of angles that would likely be used in the design of diagonal letters or italic or slanted fonts. Accompanying this, it would be nice to be able to match an angle to an existing one.

Primitive functions to support the addition of these functions by a designer are not available so they would have to be added by a programmer. The difficulty of adding these functions varies, but all of them could be added without compromising the architecture of the current system.

Production Features

Although the designers liked Paragon, and uniformly expressed interest in having a version available for their own use, it was felt that the system was not quite powerful enough for it to be a production version in its present instantiation. There are many additional functions that would need to be added to provide the ability to create and proof the large number of typefaces produced in a working environment. The features that it is missing are ones that exist in current type production systems and should not be difficult to add to Paragon. The most noted feature that was missing was the ability to create output in a specified format such as raster patterns in a run-length encoded format, or outline data with Bézier control points, but the desired format differed with each designer who suggested the addition. It would not be difficult to add a routine to provide output in any of the suggested formats, but this was felt to be as a site-specific addition since each site was likely to want a different output format.

Additional Suggestions
Other features that were suggested for addition include some that we explicitly chose to exclude from the system for one of two reasons. Some were not part of a designer's current working environment (such as the specification of constraints between character parts), while others had already been implemented and thus known to work successfully (such as interpolation between characters). We chose to concentrate on the provision of tools that were more experimental in nature, keeping in mind that if the system was successful enough to warrant it, we could consider adding these already proven features at a later date. When making a decision about what to include and what to exclude (albeit temporarily), the decision in each case was to implement those functions that were new or had not been tried before. The features that were left out were taken into consideration in the planning of the system so they may be easily added at a later date. Functions that are available on other systems include:

• *Additional graphic manipulations:* these would include zoom operations through the use of *windowing*, either to view parts of a character that do not fit on the screen or to select a small area of the screen which can then be enlarged to fill the screen. This allows a designer to work at a higher resolution and to adjust design characteristics with greater detail.

Provision of this feature would also allow us to work with characters that are larger than the screen dimensions. We would like to display as much of the character as possible but would need to provide the means for adjusting the display so the parts that are not visible could be viewed. We would need to be able to *clip* the raster output to the size of the window and provide windowing or a means of selecting and displaying different parts of the raster character in the window so that even if the designer cannot see the whole picture, she can at least see it in parts.

Windowing is used to look through a window at a larger scene behind the window (Figure 48). The portion of the larger scene that is seen through the window is called the *view*. If the view and the window are the same size, there is a one-to-one mapping of the view to the window. If the view is smaller than the window, it will have to be scaled up to fit the window, and conversely, if it is larger it will have to be scaled down. If we are trying to represent exact character shapes, we want to stick to a one-to-one representation; if we want to do fine-tuning, however, we might want to scale the view up or down to fit the window.

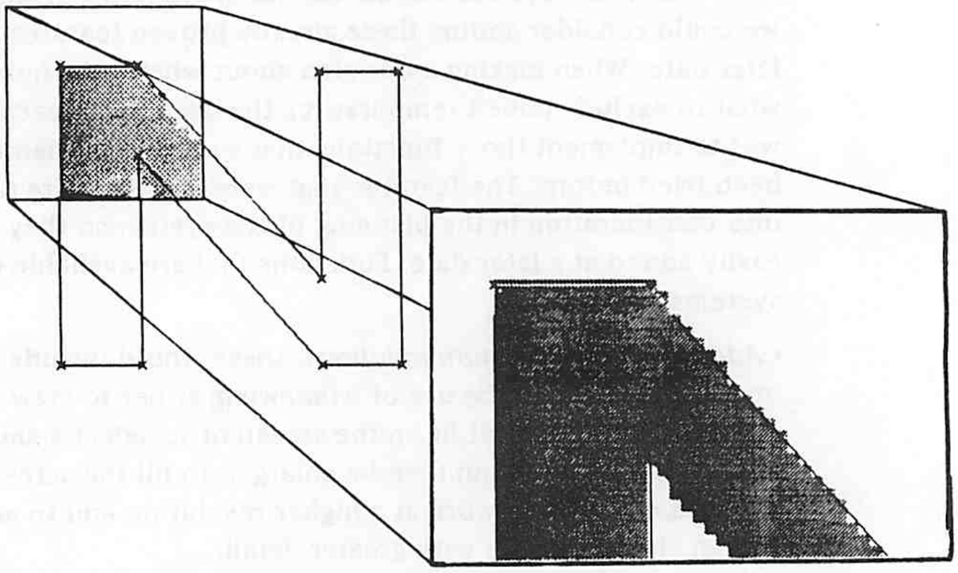


Figure 48. Windowing to see different parts of the character shape when the entire character is larger than the window.

• *Matching of character dimensions:* if one stem on a character is a certain width, it should be possible to specify that some other part of the character (possibly another stem) be matched to the first width. The only way two dimensions can be matched in the current system is to measure one by using the measure function, then to manually adjust the second until its measurement matches that of the first one.

- *Interpolation of character shapes:* given two different designs, interpolation allows a designer to automatically create an intermediate version. Usually, the two designs must have an equal number of control points along their edges, although work has been done on interpolation between shapes with a different number of control points [ELSN81], [KARO87].

- *Output from the system:* use of the system would be enhanced if it were able to produce various forms of output directly, such as GF files for Metafont, font files for Postscript, or SC files for Ikarus.

- *Display of spaced characters:* spacing of characters by lining up their sidebearings is a necessity when designing fonts. Without consideration of the spacing between characters, the font is unusable. If we could provide this on the screen display, it would give us some idea whether the characters work well together and whether their spacing is acceptable, even though it is not an adequate substitute for proofing characters on paper.

- *More sophisticated channeling operations:* we would like to be able to use an arbitrary orientation of straight edges for channeling operations. The system currently allows orientation in a horizontal or vertical direction, but this does not give the designer enough flexibility if he is working on an italic or slanted font. He will most likely want to have guidelines that are not oriented at a 90 degree angle to the baseline.

- *Slanting of characters:* a related operation would be one to automatically slant each character by a certain number of degrees. This would allow creation of slanted versions of roman faces by a simple graphic manipulation that the system could handle without the designer needing to redraw each character. Inclusion of this feature does not guarantee that a face slanted in this manner is an acceptable design, but it would facilitate the creation of such a face.

Although we would like to think that the architecture of Paragon would eventually become stable, addition of these features would still not make the system complete. The more the system is used, the more features it appears to lack. In one sense, this is encouraging, as designers are becoming more creative in thinking up ways that a computer system can enhance their design environment. In another sense, it is discouraging, since it invalidates our premise that we can build a system that is designed with a flexible enough interface to allow the designers to provide the extensions to the system. Many of the extensions that we have listed here would require the expertise of a computer scientist since most of them do not fit within our original design paradigm. Although we could redesign and rebuild the system to include them, it is likely that we will never be able to completely satisfy the design community.

Maintaining a Simple Model

The audience for any implementation of a model environment should be defined before any preparation of the material is undertaken. Whether we are writing a book, delivering a lecture, or designing a computer system, we must understand for whom this presentation is being made and structure our material accordingly. Paragon was designed for novice computer users—persons who have had little or no computer experience and who, for the most part, have had little incentive for doing so. An important goal in Paragon's design was to provide a simple model so the typeface designer will not be overburdened with either tasks or complexity. In view of this goal, several decisions were made as to what features were going to be deliberately excluded from the final system.

We chose to concentrate on crafting an environment for a type designer. We considered other issues, while resisting the compulsion to add every last gizmo and whizbang to the system without careful consideration of its impact on the entire system. We also tried to avoid implementing a simplistic answer to an intractable problem, which in most cases is worse than nothing at all. An example of this is that of meta-design and on a smaller scale, that of design-by-parts, in which individual character parts are designed, and are then used to construct characters using a specification stipulating how the parts are to be used to reconstruct each character. It is not clear whether the specification is created before the character shapes are designed, or whether the specification postdates the designs. In either case, the design of typefaces in such a manner would require an *expert* system with built-in knowledge of how characters are formed.

Although there has been some research in this area [COUE75, HERS67, MERG68, ADAM86, ADAM87], it has not yet come up with any viable solutions. The idea is to design parts of characters from which a complete character set can be built up automatically. A change in one or a few parts can result in a new look for a particular design. An example of this technique is the change of a bracketed serif to a slab serif. This might be useful if someone were generating a font for a printer of low resolution and needed darker characters; a slab serif would give the face a heavier look on the terminals and would thus be more readable if printed on a low-resolution device. The use of character parts, or meta-design is usually supported in theory, but is difficult to quantify or create, especially by a designer.

Experimental work that could be done using a system such as Paragon might provide different curve-drawing routines for a designer so that he could experiment with ones that best suited his needs. Other experimental work could be done with scan-filling algorithms. The one we have chosen to use is a good one and has been specially tuned to a type design environment, but there may be other methods that might suit a designer in other ways. Investigation into methods of automating the process of matching an outline to a particular scan-resolution could also be done.

We tried to create a system that allowed a designer to duplicate his current design environment within an interactive computer environment. We decided not to implement certain features which have been provided in other typeface environments but which are not inherently part of the designer's process. For example, this system makes no provisions for the creation of *meta*-designs, or designs that can be adapted to different point sizes by specification of meta-parameters as is done with the Metafont system [KNUT82]. Although a few designers have attempted to use Metafont as a design system [BILL87], it has not yet been determined that a designer understands how to convert the idea of 'metaness' into an actual design, nor how to go about it through a programming metaphor [SOUT85].

A problem with the meta approach is that the task of creating meta-designs is not well defined. Some meta-designs have been based on a *pen-stroke* metaphor; that is, one in which the designer draws each stroke of a letter with a variable shaped pen nib. Meta-ness is specified by varying the width or shape of the nib of the pen in addition to changing the overall dimensions of each font. The task of specifying meta-constructs to specify the changes that must be made to an outline design is much more difficult and not within the scope of this project. For example, given the stem of an outline character, if a designer wants to make the design bolder, he can simply widen the stem. The task is not quite so easy though, for in order to make the stem wider, he must move one of the edges relative to the other edge. The problems come in deciding which edge to move, in what direction to move it, how far to move it, and, less obviously, how to specify this to a design system. Because we wanted to maintain a model closest to a designer's original means of designing a character, and thus chose to employ an outline method for the specification of character shapes, we made an implicit decision to make no attempt to provide the means of specifying meta-descriptions of characters.

We also decided that Paragon would not provide any means of specifying constraints within or between parts of characters that allow the design to change while controlling which parts of a character change and how they change at different font sizes. This approach has been implemented as a part of the font production process at both Bitstream [FLOW84] and Imagen [BRAD87], both of whom claim some success at using their systems to produce different size fonts from one outline description.

Although both these functions have been shown to be useful in the aforementioned design systems, it is not clear that they are well understood by a designer who has never used them. Specification of meta-ness in the design or constraints to control the design changes at different sizes requires one of two techniques. Either a designer must analyze the relationships between the characters and come up with metrics to describe them and then specify these metrics to the system, or these relationships must be pre-specified by the design system. This is not easy to do nor is it an intuitive process. At the risk of missing some potentially useful features, we decided in the present implementation to include just those features which a designer does understand and use. Inclusion of these more complex features (and possibly others as well) remains a future design goal.

In addition, Paragon is not an input system; we are not providing any functions to support entry of pre-designed patterns. We made a deliberate decision initially that this would be a system for the design of original typefaces, not one for copying either new or old ones. There are several very successful commercial systems available for the conversion of existing designs into digital renderings; it was decided that Paragon would try neither to duplicate these systems, nor to challenge commercial enterprises with what is, essentially, a research investigation.

In this same light, we decided that Paragon would contain no bitmap editing facilities. One of the questions we are trying to answer is whether a designer can design shapes directly that conform to the restrictions imposed by converting them into raster patterns. Since it has already been established by practice that designing letterforms bit by bit is neither a viable aesthetic alternative nor a very productive one, providing this feature was not deemed to be a step in the right direction.

Paragon is also not an expert system. There is no inherent knowledge in the system regarding the aesthetics of typefaces, nor is there any built-in way of manufacturing typefaces given certain parameters or features: the system cannot take a stem and an arch and create the letter 'n'. The design process is a visual one; to create a system which understood the visual characteristics of letterforms and created its own was not our goal. We chose instead to provide the tools and to require the designer to provide the expertise.

Finally, this is not a programming system. There are no variables to be assigned, no arithmetic operations or conditional expressions, no explicit allocation of resources. Although there are internal representations for much of the data, the user has no recourse to their allocation or control. This, again, was a deliberate decision. We are trying to create a system for persons who have neither any knowledge of programming, nor any need to know. We feared that provision of even the most basic programming constructs would have compromised the original design goal: that of providing an interactive, extensible environment for non-computer technicians. We also thought any semblance of programming or the need for programming would possibly drive away those persons the system was intentionally designed to attract.

We have concentrated on the interactive aspects of the system that are design-related such as drawing, editing, saving of information, and storage of new tools. Several areas of investigation have been bypassed not because they were not necessary, or useful, or interesting, but because they were not part of the research issues we chose to investigate. This is not to say that these areas do not have challenging problems that need to be solved, but that the problems they incorporate are not those we are attempting to solve at this time. Topics for future investigation include subjects such as generation of grey-scale fonts, fragmentation of typefaces into parts which can then be used to recreate the whole, a comparison of different curve-drawing algorithms for typeface rendering, and interactive specification of meta-designs.

Chapter VII

Summary and Conclusions

There arrives, now and again, in the history of typography, a time of crisis and following it, the emergence of a new design which takes its place as the watershed between one period and the next.

Stanley Morison, *Towards an Ideal Italic,* The Fleuron

Summary

We have seen that changes in typeface design have come about largely because of changes in the technology used to display the typefaces. With the advent of each new technological change, the design mechanism has adapted to produce characters that are more suited to the current technology.

From Gutenberg's original invention of movable type to the development of font editors on computer workstations, the goal has always been been to adapt the designs to the technology to take advantage of or to minimize problems with each successive invention. Type was first constrained to fit within small boxes so the boxes could be lined up side by side. With improvements to the materials used with printing types, designs became more refined and delicate. Automatic casting machines led to designs that were constrained to fit within fixed-width categories. Photo-typesetters gave rise to typefaces that had to compensate for the effects of the photographic emulsion process. These typefaces also had to be able to stand up to scaling or stretching so that multiple designs could be produced from only a few patterns. Digital typefaces must be created to overcome the effects of conversion of the letter design into a raster pattern, effects which include such problems as uneven stem widths or jaggies along the edges.

We do not know where the future will take us. We can be sure, though, that the proliferation of the written word will be with us for quite some time and that we will need legible, attractive, and recognizable type designs in order to remain a literate and cohesive society. Although a few proposals have been put forth advocating the design of radically new typefaces for the computer age, these design changes are usually as mired in the fads of the present as the old designs are in the history of the past. Our letterforms will remain familiar and easily recognized in the future because our society is too dependent on the use of the written word to be able to consider with any seriousness any drastic changes in such a familiar medium.

Conclusions

We have designed, built and tested a type design system that has been created for the use of designers who are unfamiliar with a computer environment. This was not meant to be a production tool, but an experimental one designed to provide a testing ground for the use of new techniques in the design of digital type faces. Our goals and our achievements are:

- First, we wanted to emulate a designer's environment; we have done this by providing, within a computer model, equivalent tools to those a designer uses and a means of using them similar to that used in a traditional pencil and paper environment.

- Second, we wanted to provide a means of transition for the designer to move into the realm of designing digital faces for digital output devices. To do this, we have provided enhancements to the traditional tools used by the designer, enhancements that facilitate the design process within a computer environment. In addition, these tools assist the designer in the craft of designing digital characters. These tools include guideline adjustment, aligning of knots, and digitization of character shapes.

- Third, we have supplied a means of extending the basic system so that the designer can add tools to the system to create a rich and varied command set customized to his own specifications. The designer can extend the repertoire of available commands by defining macros and adding them to the system interactively. These added commands are permanent additions to the original system functions. The customization is all done graphically; there is no programming required.

Although this system addresses and proposes solutions to many of the problems associated with the design of digital typefaces, there is still much to do in the field. With the development and evolution of raster output devices, be they laser printers, workstation screens, dot-matrix typewriters, or phototypesetters, many problems remain to be solved. Some day we hope to create digital typefaces to equal the beauty of the traditional typefaces produced by the early practioners of this art.

Appendix A

Lexicon of Typographic Terms*

arch - curve shape within a letter connecting two straight stems.

aspect ratio - ratio between the height and width of a pixel. On a CRT screen, this ratio may not be 1:1 in which case characters that are meant to be printed on a device with a 1:1 aspect ratio will be distorted when displayed on the screen.

ascender - that part of a character that extends above the x-height.

ASCII - American Standard Code for Information Interchange. There are 128 codes each assigned to a particular letter, punctuation mark, or control function.

baseline - imaginary line upon which characters are positioned.

Bézier curve - parametric cubic defined by the position of four control points of which only the endpoints lie on the curve.

bit-editor - program that allows tuning of individual pixels in a raster.

* Sources for these definitions include: [AVIS65], [GASK76], [MINT81], [SIMO45]

bitmap – digital representation of an object. A character that has been digitized has been converted into a pattern of small discreet elements or pixels; the pattern is similar to one produced by drawing the character outline on a piece of graph paper and then darkening the squares on the paper that fall within the outline.

bitpad – tablet with a means of sensing the location of a stylus or puck on its surface.

body – the dimensions of the box within which a character is positioned. For any font, the body height is uniform; the body width varies with individual characters. In metal type, the body is slightly larger than the total character dimensions in order to allow for a small edge around the face of the type; in digital type, the characters are often the same height as the body.

bold face – typeface which is heavier and darker than its roman counterpart.

bowl – any curved part of a character surrounding a closed white space.

b-spline – curve defined by 4 or more control points, of which, only the endpoints lie on the curve.

calligraphy – from the Greek *kalligraphia*, beautiful writing.

cap-height – the normal upper extremity of capital letters; in some typefaces this is the same as the ascender height.

casting type – the process of making metal type by pouring hot metal into a mould then removing the type from the mould. Type may be cast in individual letters or as complete lines of type.

channel – a pair of parallel guidelines used to delineate points that should all align along a line that is centered between the guidelines.

cicero – measurement equal to 12 Didot points.

cold type – type that is printed by some means other than letterpress printing such as that produced on phototypesetters or laser printers.

control points – points along the edge of a character outline indicating where the outline changes its continuity or curvature.

counter – the non-printing area within an enclosed or partially enclosed character.

Cathode ray tube (CRT) – electronic beam used to etch images made up of dots or lines onto a display medium such as a terminal screen, photopaper or film.

descender – that part of a character that extends below the baseline.

Didot point – point measurement proposed by Françoise Didot in the late eighteenth century. Used mostly in European countries.

digital – composed of small discrete elements.

display typeface – typeface designed to be displayed at a large size, and used primarily in advertising or display material. Display type is not necessarily designed to be easily readable and may be distorted or ornamented in order to attract maximal attention. (See *text typeface.*)

dpi – dots per inch; used by printing manufacturers in measuring resolution of printing devices such as laser printers or phototypesetters.

dpm – dots per em; used by typeface designers in measuring resolution of typefaces.

ductal – shape that is drawn upon a flat surface with a series of strokes using a writing implement such as a brush or pen.

em – measurement equal to the square of the body type; a 12 point font has an em that is 12 points square.

en – space equal to half an em.

extruder – stroke extending beyond the usual contours for a group of letters.

face – printing surface of a piece of metal type.

family – group of typefaces based on a common design. Commonly there is a roman, a bold, and, more recently, an italic version of a design. Modern type families may contain both serif and sans serif faces.

font – a complete set of characters for a particular typeface rendered at a specific size (and, for digital type designs, at a particular resolution). When metal type was founded (or cast), the pieces of metal that were produced were called the fount.

glyph – character or character part.

glyptal – shape that is carved or etched upon a flat surface (such as the shape on the end of a punch).

grid – raster size for a particular resolution.

hairline – fine line, straight or curved, often used in the upstroke of a letter.

hot-metal type – type produced via means of casting hot metal, including hand cast type and type cast on the Linotype and Monotype machines.

ink-jet – printer employing a fine spray nozzle that shoots drops of liquid ink onto paper.

interpolation – means of producing a typeface design that is part way between two existing faces.

italic typeface – typeface designed to contrast yet harmonize with roman type. It is distinct from roman type that has been slanted as some of the italic character shapes, notably the lower case 'a' and 'g', differ from the shapes used in a roman face.

jaggies – jagged edges resulting from digitization of edges which do not align along a straight edge.

justified – set type which has an even margin along both the right and left edges.

kern – any character whose face extends over the edge of the body so that it overlaps the body of the previous or following character.

laid paper – paper made in a mould containing parallel wires laid side by side, thus producing 'laid lines' on the finished result (in contrast to *wove paper*).

laser printer – printer employing a laser beam in the process of placing marks on a page.

letterpress – printed on a press that uses metal or wood type. An inked roller is used to apply ink to the image on the type. The ink is then transferred to paper by pressing the paper onto the face of the type.

ligature – two or more letters on one body that have been designed as a single character.

lining figures – numerals which align along the baseline.

Linotype – composing machine produced by the Linotype Corporation which cast solid lines of type called slugs.

lower case – the small letters or minuscules.

majuscules – the large letters, also called capitals or uppercase.

master – large pattern from which many sizes of letters are produced usually through the use of a magnifying or reducing lens. Masters may also be used with CRT devices that enlarge or reduce the size of the writing spot to change the size of the resulting character.

matrix – copper mould containing the intaglio, or relief, impression of a punch.

metrics – character dimensions. These are used to provide size information to a text formatting program used to layout a page of text.

minuscules – the small or lower case letters.

Monotype – machine produced by the Monotype Corporation that cast individual pieces of type and was controlled by punched tape.

mouse – pointing and selection device used in conjunction with computer workstations.

o-height – height of the lowercase o.

old style figures – numerals which do not align along the baseline.

pantograph machine – machine which could cut punches or matrices from a large metal pattern.

phototypesetter – typesetter that draws characters on film or photographic paper.

phototypesetting – the process of typesetting characters on film or photographic paper.

pica – measurement of 12 points or approximately 1/6 of an inch.

pi font – punctuation or other characters which are not usually included in a font such as fractions, math symbols, ornaments, or musical signs.

pixel – from *picture element*; small discrete element directly addressable on a raster CRT screen.

point – unit of measurement used to measure font sizes; approximately equal to 1/72 of an inch.

point system – first proposed by Pierre Fournier and later promoted by Françoise Didot, it was an attempt to standardize the myriad of type sizes that were being promulgated with no thought as to their use in combination with other sizes of type or type made by other manufacturers. With standardization of sizes, printing houses could purchase fonts from different typefoundries with the assurance that the fonts were all made to the same specifications.

proof – test version of a character printed on on the medium on which it is intended to be viewed. When a punchcutter cut a new character on a punch, he would hold the face of the punch up to a candle to put soot on it, then would press the punch onto a piece of paper to see what the character shape looked like. These came to be known as *smoke proofs*. Proofs may be printed with a combination of characters so that the designer can check the spacing and evenness of the character shapes with each other.

puck – selection device used in conjunction with a bitpad or digitizing tablet. Contains crosshairs used to select a precise position, and buttons to indicate selection of one of several options.

punch – rod of steel or other hard metal upon which a character shape is carved in relief.

ragged right – text which has been set with an even left margin, but an uneven right margin.

raster – digital representation of an object. (See *bitmap*.)

Roman – the character set used in the printing of texts for Latin-based languages.

roman – upright face containing all the standard Roman upper and lowercase characters, numerals and punctuation.

sans-serif – class of typefaces which is characterized by the absence of serifs.

sector-kerning – spacing system in which characters have uneven sidebearings. The characters will butt up against each other depending on their sidebearing shapes. Characters that have been designed with sector-kerning usually need no special kerning when they are typeset.

self-spacing type – type which is designed upon a unit-grid system so that spaces added to justify a line of text need only be produced in multiples of the unit-size. Popular unit sizes were 18-, 36- and 54-to-the-em.

serif – finishing stroke on a character stem.

set – body width of a character.

sidebearing – the left or right boundary of a character.

slanted typeface – roman typeface which has been slanted, usually to the right.

sort – individual pieces of metal type in a font; when a printer ran out of a particular character, he was 'out of sorts'.

stem – one or more main vertical strokes in a character.

stencil – curved edge used as a guideline for the placement of points which should lie along the curve.

terminal – stroke ending other than a serif.

text typeface – typeface designed to be legible in large quantities at small sizes and used primarily in the printing of large amounts of text such as books, magazines, and newspapers. (See *display typeface.*)

type – piece of metal type.

typeface – unified set of character shapes which may be rendered at a particular size to create a font.

typefounding – the process of cutting punches, stamping and justifying matrices, and casting and finishing type. In early typefoundries, printing with the types may have also been done in the same location.

unit-count or **unit-cut** – letters that are constrained to fit within prespecified unit sizes that were hardwired into the typesetting machine.

uppercase – the capital or majuscule letters. (See *lowercase.*)

workstation – computer terminal with graphics display and fast computing power.

wove paper – paper made on a mould in which the wires are woven together. Wove paper has a finer grain than *laid paper* and thus can reproduce finer details in the printed characters.

writes-black – printer employing a laser beam that leaves a charge on a metal drum in the areas where the drum is supposed to pick up toner particles.

writes-white – printer employing a laser beam that de-charges a metal drum in the areas where the drum is not supposed to pick up toner particles.

x-height – height of the lowercase letters without ascenders and descenders; the height of the lowercase x.

Appendix B

Summary of Commands

Press means to press a button and to hold it down. Click means to press a button and release it.

Switch functions – Selection of one of these functions toggles a switch between two different states.

help – turns on help mode. When the system is in help mode, clicking any icon or label results in a help message related to that function being printed in the message window. To turn off help mode, click the help icon again.

display-grid – turns grid lines on or off. Grid lines will not appear unless a pointsize and a resolution have been selected and a height line and a depth line have been specified (see *guidelines*, below).

display-guidelines – turns guide lines on or off. (see *guidelines*, below).

paper-quality – switches the quality of the top sheet of paper between opaque and transparent. If the top sheet of paper is transparent, any marks on the sheet below it can be seen in the paper window. If the top sheet is opaque, only the marks on the top sheet of paper appear in the window.

paper-switch – switches between two sheets of paper. Use the *new-current* command to mark the sheets you want to work with.

Value functions – Selection of one of these items sets a value that is used with other command selections.

fill-color – selection of one of the fill colors results in that fill color being used when the *fill-curve* command is selected. Fill colors available with the basic system are: white, 25% gray, medium gray, 50% gray, left-diagonal, right-diagonal,and black.

point-size – selects the desired point size of a font. This value is used with the *fill-curve* command when filling a character and is saved with the typeface information. Can be de-selected by clicking the left mouse button while the cursor is positioned within the function label ('Pointsize').

resolution – selects the desired resolution of a font. This value is used with the *fill-curve* command when filling a character and is saved with the typeface information. Can be de-selected by clicking the left mouse button while the cursor is positioned within the function label ('Resolution').

Paper – these are commands to manipulate different sheets of paper.

new-sheet – puts a new sheet of paper on top of the stack. There are currently 8 sheets of paper available.

paper-delete – deletes a sheet of paper from the stack.

paper-current – pops the *current* sheet of paper to the top of the stack.

display-all – displays the contents of all the sheets of paper (effectively making them all transparent).

paper-cycle – cycles through the stack of used sheets of paper. Each selection of this icon results in the next sheet of paper in the stack being displayed.

paper-erase – erases all marks on the surface of the current sheet of paper. This includes all lines and knots.

new-current – makes the sheet of paper that is currently displayed the *current* sheet.

Edit – these are commands to manipulate marks on the current sheet of paper.

draw – provides a pencil-like drawing tool. Holding down the left mouse button while dragging the mouse through the paper window results in a line that follows the mouse trail being drawn in the window. Holding down the middle button provides an eraser.

mark-knot – either adds a new edge knot to the current sheet of paper, or, if the mouse is positioned over an existing knot, changes the knot to an edge knot.

mark-curve – either adds a new curve knot to the current sheet of paper, or, if the mouse is positioned over an existing knot, changes the knot to a curve knot.

connect-points – draws a connected line between knots. Each click of the left mouse button either adds a new connected knot or adds an already positioned knot to a connected line segment. Clicking of the middle mouse button over an existing knot removes the connection between it and other knots. A knot may be connected at most to two other knots.

move-point – moves a knot. If the left mouse button is pressed while the cursor is positioned over a knot, the knot is dragged as the cursor is moved. The position of the cursor when the button is released is the new knot position.

move-segment – moves a connected line segment to a new position. Press the left mouse button while the cursor is positioned over a knot in the segment, then move the cursor while the button is still pressed. The segment attached to the knot under the cursor will be moved to the new cursor position. While moving the knot, only the knot will move; the lines connecting the segment will stretch to the new position.

Once the button is released, the entire segment is moved so that the original relationship between the knots in the segment is preserved.

copy-segment – copys an existing segment. First select a knot in the segment to be copied, then select the destination position of that knot. The entire segment will be copied to the new position.

rotate-segment – rotates an existing segment. Select two knots on the segment (they need not be contiguous), then select two destination positions for the knots. The segment will be rotated so that the first selected knot is positioned on the first destination point, the second knot will be positioned on the line that passes through the first and second destination points.

scale-segment – scales an segment. This function scales a segment equally in the x and y direction. You need to enter four points. First you will draw a square indicating the *source* box. Click the left mouse button to select a point that will be one corner of a square, then select the second corner by pressing the left button and dragging the cursor until all the points you want to scale are within the square. Then release the button. The next point you will be prompted for is the *anchor* point. This is the one point on the segment whose position will not change after the character has been scaled– all other points will be moved relative to this one. Next you will draw the *destination* box, but since the first corner of the source box will also be used as the first corner of the destination box, so the final point to be entered is the second corner of the destination box. The segment will be scaled so that the contents of the first box fit inside the second box.

stretch-segment – stretches an segment. This function scales a segment unequally in the x and y direction. You need to enter three points. First, select a point with the left mouse button that will be one corner of a rectangle, then select the second corner by holding down the left button and dragging the mouse until the rectangle is the desired original size. The first corner will also be used as the first corner of the destination box, so the next point to be entered is the second corner of the destination box. The segment will be scaled so that the contents of the first rectangle fit inside the second rectangle.

connect-curve – draws a curve between knots on a segment. Select one knot on the segment and the curve will automatically be drawn.

fill-curve – fills a closed segment. Select one knot on the segment and the character will be filled with the current fill color. To see the raster for a character at a particular resolution and point size, select the point size, and the resolution, and specify a height and depth line (see Guidelines, below).

highlight – highlights a segment by drawing its connections with a dashed line. Segments must be highlighted before they can be saved as glyphs.

erase-lines – erases all lines on the current sheet of paper but leaves the knots intact.

measure – displays measurements between two knots or guidelines on the screen. Select the first position by clicking the left mouse button, then select the second position by holding the left button down until the cursor is positioned over the desired position. The measurement between the first position and the cursor will be displayed continuously as the button is held down.

align-x – aligns two knots in the x direction. Select the first knot, then select the second knot. The first knot will be aligned with the second one.

align-y – aligns two knots in the y direction. Select the first knot, then select the second knot. The first knot will be aligned with the second one. (Mnemonic: align (this one) with (that one)).

align-guide – aligns a knot with a guideline. If the guideline is adjusted, the knot will move with it. Select the knot, then select the guideline. If the second selection is at the intersection of two guidelines, the knot will be aligned with both guidelines. Each knot can be aligned with at most one horizontal and one vertical guideline.

Guidelines – allows positioning of guidelines in the paper window. Specific guidelines are provided for: depth, baseline, x-height, cap-height, height, left-sidebearing, and right-sidebearing. Only one guideline of each type is allowed. Additional guidelines are provided for arbitrary horizontal and vertical lines with up to 10 of each allowed. It should be noted that specification of guidelines is only a visual guide; no constraints are provided by the system to ensure that a design does not exceed the guidelines specified. A few of the guidelines have specific functions.

depth – lower bound of character. This must be specified for filling of character at a particular resolution and point size.

baseline – line upon which all characters rest. If specified, and a grid is selected, the grid is aligned with this position.

x-height – body size of lower case characters.

cap-height – height of uppercase letters. In some typefaces, this is the same as the *height*, so the cap-height line may not be needed.

height – upper bound of character. This must be specified for filling of character at a particular resolution and point size.

left-sidebearing – left boundary of character box. The character will be aligned along its left sidebearing with the right sidebearing of the character positioned on its left. The character edge need not be (and is usually not) aligned with the sidebearing. If specified, and the grid is switched on, the grid is aligned with this sidebearing.

right-sidebearing – right boundary of character box. The character will be aligned along its right sidebearing with the left sidebearing of the character positioned on its right. The charcter edge need not be (and is usually not) aligned with the sidebearing. In some cases, the right sidebearing may be to the left of the character edge.

horizontal guideline – horizontal line placed at an arbitrary position. There may be more than one horizontal guideline on the screen at one time.

vertical guideline – vertical line placed at an arbitrary position. There may be more than one vertical guideline on the screen at one time.

Remember – provides macro definition facilities allowing a user to customize and save new commands. The last *specified* macro is either the last remembered macro, or the last selected macro, depending on which one was specified last.

start-macro – start remembering commands. All commands up until selection of the *stop-macro* command are remembered in the order in which they are selected.

stop-macro – stop remembering commands. Terminates the list of commands that are remembered by the *start-macro* command.

repeat-macro – repeat remembered commands from last specified macro.

step-macro – repeat remembered commands step by step, pausing between each command. To proceed to the next step, select *step-macro* again.

backup-macro – delete last specified command. If used while remembering commands, the last remembered command is removed from the stack. This command has no effect on macros that have already been saved.

save-macro – save remembered macro with specified name. You must first specify a name in the *macro-label* field.

select-macro – find named macro in memory. You must specify a name in the *macro-label* field. Subsequent selection of the *repeat-macro* command will execute this macro.

cancel-macro – cancel definition or execution of macro. If this is selected while remembering a command, all remembered actions are erased from the stack. If it is selected while executing a macro, all further actions within this macro are cancelled.

macro-label – specify macro name. This is used when saving a macro, *save-macro*, or selecting a macro, *find-macro*. Terminate the macro name with a carriage return.

Toolkit – selection of the toolkit menu option pops up the customization window. Actions in this window allow the user to change action names or icons or add or delete commands in the actions window.

icon-edit – edit icon of currently selected item. A new window will pop up displaying an icon editor with the selected icon displayed in the window. When the icon has been edited, it should be saved and the window closed. (See Edit Icons, below.)

do-it – add new command to actions window. You must first use *add-function* to add the macro to the list in the toolkit window. *Do-it* adds the command to the permanent repertoire of the system and causes it to be saved in a library of user defined actions when the system is exited.

un-do-it – removes new command from actions window. You must first select the list and the command using the *select* function. You can only delete (or *un-do*) commands that have been added by a user; if the selected command is a system-defined action, it cannot be deleted.

save-it – save remembered command for future use. This function saves the command in the system library immediately so that the system does not have to be exited before the command is archived.

exit – exit Toolkit. The paper window will be redisplayed.

actions – These functions are used to add new commands to the actions window.

add-function – add new function to actions window. You will be asked to select which list you want the function added to, then will be asked to type in the function name in the *item-name* slot. The item name and a blank icon box will be displayed at the end of the selected list. You can then edit an icon for the command *icon-edit*, and add it to the actions window *do-it.*

delete-function – delete function from actions window. This command deletes a function that has not yet been added to the actions window. If you have already selected *do-it* for this function, you must use *un-do-it* to remove the function.

list-label – name of selected group of commands. This is the header name for a list of functions. Editing it and saving it by selecting *do-it* will result in a new name being displayed in the actions window.

item-name – name of selected function within the group. This is the name of a particular function. It can be edited and then saved by selection of *do-it* to display a new function name. If the function is displayed with an icon in the actions window, this command will only change the name when the function is displayed in the Toolkit window.

Edit Icons – edit and save new icons for actions window.

Box – draws or erases a box around the outside of the icon. Toggling the item causes the box to be alternately drawn or erased.

Invert – inverts the image in the icon so that white pixels become black and black ones become white. Toggling the item switches between the two images.

Quit – exit the icon editor. You must *save* any changes you have made before you exit. You must select *quit* twice before the icon editor will be closed.

Load – reads in another icon. Select another function in the Toolkit window and then select *load* to load the new icon into the icon editor. You should save the first icon before loading in a new one if you want to save any changes that have been made to the first one.

Save – saves an icon and copies it to the Toolkit window.

Item Name – renames an icon. This currently has no effect.

Fill – fills the background with the selected color. You must select the color before selecting the *fill* command. Available colors are: white, 25% gray, root gray, 50% gray, 75% gray, black.

Replace/Merge – determines whether the *load* and *fill* commands will replace or merge with the pixels that are already displayed in the window.

Proof Background – changes the background color in the proof window in the upper left of the icon editor. Available colors are: white, 25% gray, root gray, 50% gray, 75% gray, black.

Glyphs – selection of the glyph menu item pops up a window containing functions to save or retrieve glyph information that is stored in an external file.

copy – copies one glyph to another. Select *copy* then select the source glyph and the destination glyph.

rename – renames glyph to new name. If both names are already in the list, select *rename* then select the source glyph and then the destination glyph. If either of the names is not in the list, add it to the list (using *glyph-name* and *save-glyph*, then rename the glyph.

highlight – same as highlight command in actions window. It is repeated here so that the user does not need to switch to the actions window to change which segments are highlighted. This command highlights a segment by drawing its connections with a dashed line. Segments must be highlighted before they can be saved as glyphs.

exit – exit glyph naming window. The window will be closed and the actions window redisplayed.

Glyph – save or retrieve glyph information from file.

get-glyph – retrieve selected glyph. Glyph should be selected from the *Glyphs* list.

save-glyph – save information for selected glyph. Glyph should be selected from the *Glyphs* list or entered in the glyph-name field if it does not appear in the glyphs list.

delete-glyph – delete selected glyph. Marks glyph as deleted so that when the file is saved, information for this glyph is not. This can be reset by selecting *save-glyph*.

glyph-name – name selected glyph. Enter glyph name with no embedded blanks and terminated by a carriage return. When *save-glyph* is selected, the name will be added to the glyphs list.

File – save and retrieve file information.

get-file – retrieve named file. Reads in all information from specified external file. File name should be entered in *file-name* field.

save-file – save glyph and font information in named file. Writes out all saved glyphs to external file. Additional information such as position of guidelines, pointsize and resolution are also saved.

file-name – name font file. Enter file name with no embedded blanks and terminated by a carriage return. When *save-file* is selected, the file will be written out. If glyph information has been saved and a file name has been specified, the file is saved when the program is exited if it has not already been saved.

Glyphs – list of glyph names. This list initially contains most commonly used letters, digits and punctuation. New names can be added to the list through the *glyph-name* function.

Bibliography

ADAM86	Adams, Debra Anne, *A Dialogue of Forms: Letters and Digital Font Design*, MS Thesis in Visual Studies, Department of Architecture, Massachusetts Institute of Technology, September 1986.
ADAM87	Adams, Debra A., abcdefg (A Better Constraint Driven Environment for Font Generation), in *Workshop on Font Design Systems*, INRIA/Irisa, May 1987, pp. 69–90.
ADLE73	Adler, Michael H., *The Writing Machine, a History of the Typewriter*, George Allen and Unwin, Ltd., London, 1973.
ADOB87	*Adobe Illustrator User's Manual*, Adobe Systems Incorporated, Palo Alto, California, 1987.
AVIS65	Avis, F.C., *Type Face Terminology*, Glenview Press, London, 1965.
AYAL84	Ayala, Juan E., et al.,Technology Trends in Electrophotography, *IBM Journal of Research and Development*, 28(3):241–251, May 1984.
BARN74	Barnhill, Robert E. and Richard F. Riesenfeld, *Computer Aided Geometric Design*, Academic Press, New York, 1974.
BARS84	Barstow, David R., Howard E. Shrobe, From Interactive to Intelligent Programming Environments, in *Interactive Programming Environments*, McGraw-Hill Book Company, New York, 1984, pp. 558–570.
BAUD76	Baudelaire, P., *The Fred User's Manual*, Internal Report, Xerox Palo Alto Research Center, Palo Alto, California, 1976.

BAUD80 Baudelaire, Patrick, and Maureen Stone, Techniques for Interactive Raster Graphics,*SIGGRAPH '80*, 14(3):314–320, July 1980.

BEAU26 Beaujon, Paul (pseudonym of Beatrice Warde), The 'Garamond' Types, Sixteenth and Seventeenth Century Sources Considered, *The Fleuron*, London, 5:131–179, 1926.

BENN76 Bennett, John L., User-Oriented Graphics Systems for Decision Support in Unstructured Tasks, *User Oriented Design of Interactive Graphics Systems*, ACM/SIGGRAPH Workshop, October 1976, pp. 3–11.

BENN35 Bennett, Paul A., On Recognizing the Type Faces, *The Dolphin*, The Limited Editions Club, New York, 2:11–57, 1935.

BERR83 Berry, W. Turner, and A.F. Johnson, *Catalogue of Specimens of Printing Types by English and Scottish Printers and Founders, 1665-1830*, Garland Publishing, Inc., New York, 1983.

BERS77 Bers, A.A., Implementation and Design for the Printing of Typefaces for an Automatic Phototypesetting System, Academy of Science, USSR, Siberia Division, Computer Center, October 1977.

BIGE78 Bigelow, Charles, *Type Design Principles and Automated Design Aids*, Rhode Island School of Design, Providence, Rhode Island, 1978.

BIGE79 Bigelow, Charles, On Type: Galliard, *Fine Print*, 5(1):27–30, January 1979.

BIGE81 Bigelow, Charles, Technology and the Aesthetics of Type, Maintaining the Tradition in the Age of Electronics, *The Seybold Report*, 10(24):3–16, August 24, 1981.

BIGE82a Bigelow, Charles, The Principles of Digital Type, Quality Type for Low, Medium and High Resolution Printers, Part I, *The Seybold Report*, 11(11):3–23, February 8, 1982.

BIGE82b Bigelow, Charles, The Principles of Digital Type, Quality Type for Low, Medium and High Resolution Printers, Part II, *The Seybold Report*, 11(12):10–19, February 22, 1982.

BIGE83 Bigelow, Charles, and Don Day, Digital Typography, *Scientific American*, August 1983.

BIGE84 Bigelow, Charles, Principles of Font Design for the Personal Workstation, *Byte Magazine*, November 1984.

BILL86 Billawala, Nazneen, *Metamarks, A Series of Studies for a Pandora's Box of Shapes*, Department of Computer Science, Stanford University Technical Report, (to appear).

BLUM35 Blumenthal, Joseph, The Fitting of Type, *The Dolphin*, Limited Editions Club, New York, No. 2, 1935.

BODO18 Bodoni, Giambattista, *Manuale Typografico*, Presso La Vedova, Parma, 1818.

BODO25 Bodoni, Giovanni Battista, *Preface to Manuale Typografico of 1818*, Translated by H.V. Marrot, Elkins Mathews Ltd., London, 1925.

BRAD87 Brady, Fred, A Constraint Based Design System, supplement to *Workshop on Font Design Systems*, INRIA/Irisa, May 1987.

BULL22 Bullen, Henry Lewis, Linn Boyd Benton – The Man and His Work, *The Inland Printer*, October 1922, pp. 60–64.

BULL26 Bullen, Henry Lewis, *Nicolas Jenson, Printer of Venice: his famous type designs and some comment upon the printing types of earlier printers*, John Henry Nash, San Francisco, 1926.

BUNN78 Bunnell, Edward H., *Understanding Digital Type*, National Composition Association, 1978.

BURT59 Burt, Sir Cyril, *Psychological Study of Typography*, The University Press, Cambridge, 1959.

CAME82 Camex Corporation, Letter Input Processor, Brochures from Camex, Boston, Massachusetts, 1982.

CAFL83 Caflisch, Max, Type and Paper: The Interdependency of Paper Surface, Printing Process, and Printing Types, *Fine Print*, 9(4):138–142,158, October 1983.

CART29 Carter, Harry, Observations on Modern Type Design, *Gutenberg-Jahrbuch*, 1929, pp. 297–302.

CART37 Carter, H.G., The optical scale in typefounding, *Typography*, 4:2–6, 1937.

CART54 Carter, H.G., Letter design and typecutting, *Journal of the Royal Society of Arts*, 102:878–895, 1954.

CART69 Carter, H.G., *A view of early typography up to about 1600*, Clarendon Press, Oxford, 1969.

CART84a Carter, K.A.,*Imp - a system for computer-aided typeface design*, Computer Laboratory, University of Cambridge, May 22, 1984.

CART84b Carter, K.A., *A Window Manager for the Rainbow Workstation*, Rainbow Group Note, University of Cambridge, May 23, 1984.

CART85a Carter, Kathleen Anne, *Computer-aided Type Face Design*, PhD Thesis, University of Cambridge, November, 1985.

CART82 Carter, Matthew, *Bell Centennial*, Type and Technology Monograph #1, The Center for Design & Typography, The Cooper Union, New York, 1982.

CART85b Carter, Matthew, Galliard: the types of Robert Granjon, *Visible Language* 19(1):77–97, 1985.

CART87 Carter, Sebastian, *Twentieth Century Type Designers*, Taplinger Publishing Company, New York, 1987.

CASE82 Casey, R.G., et al, Automatic Scaling of Digital Print Fonts, *IBM Journal of Research and Development*, 26(6):657–666, November 1982.

CATI68 Catich, Edward M., *The Origin of the Serif: Brush Writing and Roman Letters*, The Catfish Press, St. Ambroise College, Davenport, Iowa, 1968.

CHAP80 Chappell, Warren, *A Short History of the Printed Word*, Nonpareil Books, Boston, 1980.

CHAT62 Chatterjee, Mohon, *Evolution of Printing Types*, Universal Book Distributors, Calcutta, 1962.

COUE73 Coueignoux, Philippe J.M., *Compression of Type Faces by Contour Coding*, MS Thesis, Massachusetts Institute of Technology, June 1973.

COUE75 Coueignoux, Philippe J.M., *Generation of Roman Printed Fonts*, PhD Thesis, Massachusetts Institute of Technology, June 1975.

COUE81 Coueignoux, Philippe J.M., Character Generation by Computer, *Computer Graphics and Image Processing*, 16:240–269, 1981.

CROW70 Crowel, Wim, Type Design for the Computer Age, *Journal of Typographic Research*, 4(1):51–58, 1970.

CSUR74 Csuri, Charles, Computer Graphics and Art, *Proceedings of the IEEE*, April 1974, pp. 421–433.

DAY68 Day, Kenneth, Types of the Sixties, *The Penrose Annual*, Vol. 61, London, 1968.

DEDO82 deDoes, Bram, *Trinité, 1, 2, 3, AsA types*, Autologic SA, Switzerland, No. 2, June 1982.

DEDO83 deDoes, Bram, Trinité, lecture given at the Fifth ATypI Working Seminar, *The Computer and the Hand in Type Design; the Aesthetics and Technology of Digital Letterforms*, 1983.

DEGA83 Degano, P., and E. Sandewall, ed., *Integrated Interactive Computing Systems*, North-Holland Publishing Company, New York, 1983.

DENM55 Denman, Frank, *The Shaping of our Alphabet; a Study of Changing Type Styles*, Knopf, New York, 1955.

DEVI24 DeVinne, Theodore Low, *Printing in the Nineteenth Century*, Lead Mould Electrotype Foundry, New York, 1924.

DIDO19 Didot, Pierre, *Specimen des Nouveaux Charactères de L'Imprimerie et de la Fonderie*, Perre Didot, L'Ainé et Jules Didot, fils, Paris, 1819.

DOWD61 Dowding, Geoffrey, *An Introduction to the History of Printing Types*, Wace and Company, Ltd., London, 1961.

DOWD66 Dowding, Geoffrey, *Finer points in the spacing and arrangement of type*, Wace and Company, Ltd., Clerkenwell, England, 1966.

DREY68 Dreyfus, John, Sabon: the first 'harmonized' type, *The Penrose Annual*, London, 1968, pp. 65–74.

DROS85 Drost, Henk, Punchcutting demonstration, *Visible Language* 19(1):99–105, 1985.

DWIG39 Dwiggins, W.A., *Caledonia, A New Printing Type*, Mergenthaler Linotype Company, New York, 1939.

DWIG40 Dwiggins, W.A., *WAD to RR: a letter about designing type*, Cambridge, Mass., Harvard College Library, 1940.

ELLI32 Elliott, R.C., The Monotype from infancy to maturity, *Monotype Recorder*, 31(243):8–39, 1932.

ELSN80 Elsner, V., Ikarus: computergesteuerte Vorlagernerstellung für Foto-, CRT- und Lasersatz, *Typographische Monatsblaetter*, No. 2, 1980, pp. 69–79.

ELSN81 Elsner, V., Ikarus: Computer-controlled production of fonts for photo/CRT/laser composition, *Baseline*, No. 3, Typographic Systems International, Ltd., London, 1981.

FINZ84 Finzer, William and Laura Gould, *Programming by Rehersal*, Byte Magazine, June 1984, pp. 187–210.

FLOW84 Flowers, Jim, Digital Type Manufacture: An Interactive Approach, *Computer*, May 1984, pp. 40–48.

FOLE74 Foley, James D., and Victor Wallace, The Art of Natural Graphic Man-Machine Conversation, *Proceedings of the IEEE*, April 1974.

FOLE82 Foley, James D., and Andries Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.

FOLE84 Foley, James D., Victor L. Wallace, and Peggy Chan, The Human Factors of Computer Graphics Interaction Techniques, *IEEE Computer Graphics and Applications*, November 1984, pp.13–48.

FONT87 *Fontographer, Professional Font Editor User's Guide*, Altsys Corporation, Plano, Texas, 1987.

FOUR30 Fournier, Simon Pierre, *Fournier on typefounding: the text of the Manuel Typographique (1764-1766)*, translated and edited by Harry Carter, London, Soncino Press, 1930.

FRUT67 Frutiger, A., OCR-B; A Standardized Character for Optical Recognition, *Journal of Typographic Research*, 1(2):137–146, April 1967.

FRUT70 Frutiger, A., Letterforms in photo-typography, *Journal of Typographic Research*, 4(4):327–335, 1970.

FRUT80 Frutiger, Adrien, *Type sign symbol*, ABC Verlag, Zurich, 1980.

GASK76 Gaskell, Philip, A Nomenclature for the Letterforms of Roman Type, *Visible Language* 10(1):41–51, 1976.

GILL36 Gill, Eric, *An Essay on Typography*, Sheed and Ward, London, 1936.

GILL78 Gill, Eric, *Letterforms and Type Designs*, Typophiles, 1978.

GILL80 Gilliam, Barbara, Geometrical Illusions, *Scientific American*, January, 1980, pp. 102–111.

GOLD68 Goldring, Maurice, Typographic design and the computer, *Penrose Annual*, 61:41–46, 1968.

GOLD83 Goldberg, Adele, and David Robson, *Smalltalk-80, The Language and its Implementation*, Addison Wesley Publishing Company, Menlo Park, California, 1983.

GORD74 Gordon, William J., and Richard F. Riesenfeld, B-Spline Curves and Surfaces, in *Computer Aided Geometric Design*, Academic Press, New York, 1974.

GOUD33 Goudy, Frederic W., Designing a Typeface, *The Dolphin*, The Limited Editions Club, New York, 1933.

GOUD36 Goudy, Frederic W., *Types of the Past, type revivals with a few words on type design in general*, School of Journalism, Syracuse University, Syracuse, New York, 1936.

GOUD40 Goudy, Frederic W., *Typologia*, University of California Press, Berkeley, California, 1940.

GOUL84 Gould, Laura, and William Finzer, *Programming by Rehersal*, Xerox Palo Alto Research Center, Palo Alto, California, 1984.

GRAB49 Grabhorn, Robert, *The Story of Centaur Type*, New Harmony Press, San Francisco, 1949.

GRAY76 Gray, Nicolete and Ray Nash, *19th century ornamented type faces*, University of California, Berkeley, 1976.

GRAY82 Gray, Nicolete, *Lettering as Drawing*, Taplinger Publishing Co., Inc., New York, 1982.

GUIB82 Guibas, Leo J., and Jorge Stolfi, A Language for Bitmap Manipulation, *ACM Transactions on Graphics*, 1(3):191–214, July 1982.

GÜRT68 Gürtler, André, The Design of Egyptian 505, *Journal of Typographic Research*, 2(1):27–42, 1968.

GÜRT77a Gürtler, André, Christian Mengelt, and Erich Gschwind, *From Helvetica to Hass Unica*, Haas Typefoundry, Munchenstein, Switzerland, 1977.

GÜRT77b Gürtler, André, and Christian Mengelt, Cyrillic Gothic: Formal Modifications in the Design of a Russian Sans-serif Typeface, *Journal of Typographic Research*, 11(1):25–36, 1977.

GÜRT85 Gürtler, André, and Christian Mengelt, Fundamental Research Methods and Form Innovations in Type Design Compared to Technological Developments in Type Production, *Visible Language*, 19(1):123–147, 1985.

HART58 Hartz, S.L., An approach to type designing, *Penrose Annual*, London, 52:39–42, 1958.

HATT73 Hattery, Lowell H., and George P. Bush, editors, *Technological Change in Printing and Publishing*, Hayden Book Co. Inc. New Jersey, 1973.

HAYE81 Hayes, Phil, Eugene Ball, and Raj Reddy, Breaking the Man-Machine Communication Barrier, *Computer (IEEE)*, March 1981, pp. 19–30.

HERO76 Herot, Christopher F., Sketch Recognition for Computer Aided Design, *User Oriented Design of Interactive Graphics Systems*, ACM/SIGGRAPH Workshop, October 1976, pp. 31–35.

HERS67 Hershey, Allen V., *Calligraphy for Computers*, Technical Report No. 2101, Computation and Analysis Laboratory, United States Naval Weapons Laboratory, Dahlgren, Virginia, August 1, 1967.

HERS72 Hershey, Allen V., A Computer System for Scientific Typography, *Computer Graphics and Image Processing*, 4:373–385, December 1972.

HESS37 Hess, Sol, *Janson, an authentic revival of a classic face adapted to the Monotype*, Lanston Monotype Machine Co., Philadelphia, 1937.

HESS Hess, Sol, *The Origin and Development of Printing Types*, Lanston Monotype Machine Company, Philadelphia, (no date).

HOBB85a Hobby, John, *Smooth, Easy to Compute Interpolating Splines*, Department of Computer Science, Stanford University Technical Report No. STAN-CS-85-1047, January 1985.

HOBB85b Hobby, John Douglas, *Digitized Brush Trajectories*, Department of Computer Science, Stanford University Technical Report No. STAN-CS-85-1070, August 1985.

HOLL67 Holland, F.C., Typographical Effects by Cathode Ray Tube Typesetting Systems, *Journal of Typographic Research*, 1(1):69-79, 1967.

HOLM80 Holmes, Kris, On Type: ITC Zapf Chancery, *Fine Print*, 6(1):26-30, January 1980.

HOLM85 Holmes, Kris, On Type: Isadora, *Fine Print*, 11(3):148-152, July 1985.

HOPG83 Hopgood, F.R.A., et al., *Introduction to the Graphical Kernal System, GKS*, Academic Press, New York, 1983.

HUSS73 Huss, Richard, *The Development of Printers' Mechanical Typesetting Methods, 1822-1925*, University of Virginia Press, Charlottesville, 1973.

HUTC69 Hutchins, Michael, *Typographis, A designer's handbook of printing techniques*, Reinhold Book Corp., New York, 1969.

JACK70 Jacks, Edwin L., Designer's Choice at the Console, in *Picture Language Machines*, ed. by S. Kaneff, Academic Press, New York, 1970.

JOHN30 Johnson, A.F., The Evolution of the Modern Face Roman, *The Library*, Oxford University Press, Oxford, Series 4, 11:353-377, December 1930.

JOHN34 Johnson, A.F., *Type Designs: their history and development*, Grafton & Co., London, 1934.

JOHN87 Johnson, Bridget Lynn, *A Model for Automatic Optical Scaling of Type Designs for Conventionsl and Digital Technology*, MS Thesis, Rochester Institute of Technology, May, 1987.

JOHN73 Johnson, John R., On Certain Improvements in the Manufacture of Printing Types, *Journal of the Society of Arts*, March 21, 1873, pp. 330-338.

JOHN13 Johnston, Edward, Decoration and its Uses, *The Imprint*, Jan.-Nov.1913, 1:7-14, 2:128-133, 3:201-205, 4:252-256, 5:345-352, 6:428-430, 7:79-84.

JUST72 Justis, Paul E., There is more to Typesetting than Setting Type, *Transactions on Professional Communication*, PC-15(1), March 1972, pp. 13-16.

KARO87 Karow, Peter, *Digital Formats for Typefaces*, URW Verlag, Hamburg, West Germany, 1987.

KIND69 Kindersley, David, Optical Letter Spacing, *The Penrose Annual*, 62:167-176, 1969.

KIND73 Kindersley, David, Space Craft, *Visible Language*, 7(4):311-324, 1973.

KIND76 Kindersley, David, *Optical Letter Spacing for New Printing Systems*, The Wynkyn de Worde Society, London, 1976.

KIND79 Kindersley, David and Neil Wiseman, Computer Aided Letter Design, *Printing World*, October 31, 1979.

KIND81 Kindersley, David and Lida Lopez Cardozo, *Letters Slate Cut*, Pentalic Corp, New York, 1981.

KLEM77 Klemke, Werner, *Leben und Werk des Typographen Jan Tschichold*, VEB Verlag der Kunst, Dresden, 1977.

KNOW72 Knowlton, Ken, Collaboration with Artists – A Programmer's Reflections, in *Graphic Languages, Proceedings of the IFIP Working Conference on Graphical Languages*, ed. F. Nake, A. Rosenfeld, North Holland Publishing Co., Amsterdam, 1972, pp. 399-403.

KNUT79 Knuth, Donald E., *Tex and Metafont, New Directions in Typesetting*, Digital Press and the American Mathematical Society, Bedford, Massachusetts, 1979.

KNUT80a Knuth, Donald E., *The Computer Modern family of typefaces*, Stanford University Technical Report No. STAN-CS-80-780, 1980.

KNUT80b Knuth, D.E., The Letter S, *The Mathematical Intelligencer*, 2:114–122, 1978.

KNUT82 Knuth, Donald E., The Concept of a Meta-Font, *Visible Language*, 16(1):3–27, 1982.

KNUT85 Knuth, Donald E., Lessons Learned from Metafont, *Visible Language* 19(1):35–53, 1985.

KNUT86 Knuth, D.E., *The Metafontbook*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

KOCH33 Koch, Paul, The Making of Printer's Types, *The Dolphin*, Limited Editions Club, New York, No. 1, 1933, pp. 24–57.

KOHE85 Kohen, Eliyezer, *An Interactive Method for Middle Resolution Font Design on Personal Workstations*, Eidgenössische Technische Hochschule Zürich, Institut Für Informatik, Zürich Switzerland, 1985.

KOHE85 Kohen, Eliyezer, A simple and efficient way to design middle resolution fonts, in *Workshop on Font Design Systems*, INRIA/Irisa, May 1987, pp. 3–19.

KUHN79 Kuhn, Larry, and Robert A. Myers, Ink-Jet Printing, *Scientific American*, June 1979, pp. 120–132.

LAMP79 Lampson, Butler et al., *Alto User's Handbook*, Xerox Palo Alto Research Center, September 1979.

LANG71 Lange, Gunter G., On the Technical Typographical Changes of the Letterform in Metal- and Photo-composition, *Der Druckspiegel*, 26(1):17–28, January 1971.

LAWS71 Lawson, Alexander, *Printing Types: an Introduction*, Beacon Press, Boston, 1971.

LEGR16 Legros, L.A. and J.C. Grant, *Typographical Printing-surfaces*, Longmans, Green and Co., London, 1916.

LEGR22 Legros, Lucien Alphonse, *A Note on the Legibility of Printed Matter*, His Majesty's Stationery Office, 1922.

LICK76 Licklider, J.C.R., User-Oriented Interactive Computer Graphics, *User Oriented Design of Interactive Graphics Systems*, ACM/SIGGRAPH Workshop, October 1976, pp. 89–96.

LIEB83 Lieberman, Henry, Designing Interactive Systems from the User's Viewpoint, *Integrated Interactive Computing Systems*, Degano, P., and E. Sandewall, ed., North-Holland Publishing Company, New York, 1983, pp. 45–59.

LIEB85 Lieberman, Henry, There's More to Menu Systems than Meets the Screen, *SIGGRAPH '85*, 19(3):181–190, July 1985.

LIES78 Lieser, G., Production of high-quality arabic texts on a CRT filmsetting machine, Paper presented at the 5th symposium 'Computers in literary and linguistic research', Birmingham, England, 1978.

LOEL81 Loeliger, R.G., *Threaded Interpretive Languages, their Design and Implementation*, Byte Books, Peterborough, New Hampshire, 1981.

MACK71 MacKellar, Thomas, *The American Printer: A Manual of Typography*, MacKellar, Smith and Jordan, Philadelphia, 1871.

MACL83 MacLennan, Bruce J., *Principles of Programming Languages: Design, Evaluation, and Implementation*, Holt, Rinehart and Winston, New York, 1983.

MCMU21 McMurtrie, Douglas C., *Proofreading in the 15th Century*, Condé Nast Press, Greenwich, Connecticut, 1921.

MCMU27 McMurtrie, Douglas C., *Type Design; an essay on American type design with specimens of the outstanding types*, John Lane & the Bodley Head Ltd., London, 1927.

MCPH78 McPherson, Michael, *Electronic Textsetting, the impact of revolutions in composition on typography and type design*, Graphic Design Department, Rhode Island School of Design, 1978.

MCRA20 McRae, J.F., *Two Centuries of Typefounding*, London, 1920.

MAIE80 Maier, Manfred, *Basic Principles of Design*, Van Nostrand Reinhold Company, New York, 1980.

MALL83 Mallgren, William R., *Formal Specification of Interactive Graphics Programming Languages*, The MIT Press, Cambridge, 1983.

MATH54 *Mathematics in Type*, The William Byrd Press, Richmond, Virginia, 1954.

MATH67 Mathews, M.V., C. Lochbaum, and J.A. Moss, Three Fonts of Computer Drawn Letters, *Journal of Typographic Research*, 1(4):345–356, 1967.

MEGG83 Meggs, Philip B., *A History of Graphic Design*, Van Nostrand Reinhold Company, New York, 1983.

MENG54 Mengel, Willi, *Ottmar Mergenthaler and the Printing Revolution*, Mergenthaler Linotype Co., Brookline, New York, 1954.

MERG35 Mergenthaler Linotype Co., *The Legibility of Type*, 1935.

MERG20 Mergenthaler Linotype Co., *One-Line Specimens*, Mergenthaler Linotype Co., New York, 1920.

MERG68 Mergler, H.W., and P.M. Vargo, One Approach to Computer Assisted Letter Design, *Journal of Typographic Research*, 2(4):299–322, 1968.

MIDD49 Middleton, Robert Hunter, *Creating Type*, Diamant Typographic Services, New York, 1949.

MINT81 Mintz, Patricia Barnes, *Dictionary of Graphic Arts Terms, A communication tool for people who buy type and printing*, Van Nostrand Reinhold Company, New York, 1981.

MONO32 Monotype Corporation, An Approach to Type Design in the Twentieth Century, *Monotype Recorder*, 246:17–21, September/October 1932.

MORA74 Moran, James, ed., *Printing in the 20th Century, A Penrose Anthology*, Visual Communication Books, Hastings House Publishers, New York, 1974.

MORA78 Moran, James, *Printing Presses, History and Development from the 15th Century to Modern Times*, University of California Press, Berkeley, California, 1978.

MORI23 Morison, Stanley, *On Type Faces, examples of the use of type for the printing of books*, The Medici Society and The Fleuron, London and Westminster, 1923.

MORI24 Morison, Stanley, Towards an Ideal Type, *The Fleuron*, London, 2:57–75, 1924.

MORI30 Morison, Stanley, First Principles of Typography, *The Fleuron*, London, 7:61–72, 1930.

MORI32 Morison, S., The design and cutting of The Times new roman: how matrices are made, *Monotype Recorder*, 31(246):12–15, 1932.

MORI36 Morison, Stanley, *First Principles of Typography*, The MacMillian Company, New York, 1936.

MORI50 Morison, Stanley, *The Typographic Arts*, Harvard University Press, Cambridge, Massachusetts, 1950.

MORI53 Morison, S., *A tally of types cut for machine composition and introduced at the University Press*, University Printing House, Cambridge, England, 1953.

MORI57 Morison, Stanley, *Four Centuries of Fine Printing*, Farrar, Straus and Cudahy, New York, 1957.

MORI59 Morison, S., *Typographic Design in Relation to Photographic Composition*, The Book Club of California, San Francisco, 1959.

MORI62 Morison, Stanley, *On Type Designs*, E. Benn, London, 1962.

MORI68 Morison, Stanley, *Letterforms Typographic and Scriptorial: two essays on their Classification, History and Bibliography*, The Typophiles, New York, 1968.

MOSL82 Mosley, James, Eric Gill's Perpetua Type, *Fine Print*, 8(3):90–95, July 1982.

MOXO58 Moxon, J., *Mechanick exercises on the whole art of printing (1683)*, edited by Herbert Davis and Harry Carter, London, Oxford University Press, 1958.

MOYR65 Moyroud, L., Les machines a composer Photon Lumitype, in 1er congres international d'information technique: composition automatique des textes Paris, Compagnie Francaise d'Editions, 1965.

NAIM85 Naiman, Avi C., High-Quality Text for Raster Displays, MS Thesis, Department of Computer Science, University of Toronto, January 1985.

NEGR76 Negroponte, Nicholas, An Idiosyncratic Systems Approach to Interactive Graphics, *User Oriented Design of Interactive Graphics Systems*, ACM/SIGGRAPH Workshop, October 1976, pp. 53–60.

NELS85 Nelson, Stanley, Mould making, matrix fitting and hand casting, *Visible Language*, 19(1):107–120, 1985.

NEWM79 Newman, William M., and Robert F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill Book Company, New York, 1979.

NEWT21 Newton, A. Edward, *A Noble Fragment being a Leaf of the Gutenberg Bible, 1450-1455*, Gabriel Wells, New York, 1921.

OBRI75 O'Brien, C. D., and Bown, H. G., Image: a language for the Interactive Manipulation of a Graphics Environment, *SIGGRAPH '75 Conference Proceedings*, 9(1):53–60, June 1975.

OVIN80 Ovink, W., Review of P.H. Raedisch, A tot Z, *Quaerendo*, 10(2):164–72, 1980.

PAVL83 Pavlidis, Theo, Curve Fitting with Conic Splines, *ACM Transactions on Graphics*, 2(1):1–31, 1983.

PHIL68 Phillips, A. H., *Computer peripherals and typesetting*, HM Stationery Office, London, 1968.

PLAS83 Plass, Michael and Maureen Stone, *Curve-Fitting with Piecewise Parametric Cubics*, Imaging Sciences Laboratory, Xerox Palo Alto Research Center, March 1983.

POLL71 Pollak, Michael, Durability of Fifteenth-century Type, *Visible Language*, 5(2):159–188, 1971.

PRAT85 Pratt, Vaughn, Techniques for Conic Splines, *SIGGRAPH '85*, 19(3):151–159, July 1985.

PRIN79 Pringle, A.M., P. Robinson, and N.E. Wiseman, Aspects of Quality in the Design and Production of Text, *SIGGRAPH '79*, 13(2):63–70, July 1979.

PURD78 Purdy, Peter and Ronald McIntosh, PM Digital Spiral, (Journal Unknown), *Britain in Print, Forward Thinking*, 1978.

RAMS80 Ramshaw, Lyle and Kerry A. LaPrade, *Prepress Manual*, version 2.1, Xerox Corporation, September 1980.

REED52 Reed, Talbot Baines, *Old English Letter Foundries*, Faber and Faber Ltd., London, 1952.

RICH82 Richmond, Wendy, Letter Input Processor, Internal Memo, Camex Corporation, October 7, 1982.

ROGE76 Rogers, David F., and J. Alan Adams, *Mathematical Elements for Computer Graphics*, McGraw-Hill Book Co., New York, 1976.

ROSE68 Rosenblum, Louis, Harmonic System for Computer-Controlled Typesetting, *The Penrose Annual*, 61:257–266, London, 1968.

RUGG83 Ruggles, Lynn, Letterform Design Systems, Department of Computer Science, Stanford University, Technical Report No. STAN–CS–83–971, April 1983.

RUGG87 Ruggles, Lynn, Paragon, a Type Design System for Designers, in *Workshop on Font Design Systems*, INRIA/Irisa, May 1987, pp. 58–67.

SCHU70 Schulz-Anker, Erich, *Syntax-Antiqua, a Sans Serif on a New Basis*, Gebrauchsgraphik, No. 7, 1970.

SCOT82 Scott, Joan E., *Introduction to Interactive Computer Graphics*, John Wiley and Sons, New York, 1982.

SEYB65 Seybold, Jonathan, Between Here and Someday, *Automation and Electronics in Publishing*, Hattery and Bush, ed., Spartan Books, New York, 1965, pp. 149–156.

SEYB68 Seybold, John W., Esthetic Values in Computerized Photocomposition, *Journal of Typographic Research*, 2(4):341–350, October 1968.

SEYB79a Seybold, Jonathan, Digitized Type: What is it? What does it mean for typesetting and word processing?, *The Seybold Report*, 8(24):3–17, August 27, 1979.

SEYB79b Seybold, J.W., *Fundamentals of modern photocomposition*, Media, Pennsylvania, Seybold Publications, 1979.

SEYB81 Seybold, Jonathan, Xerox's Star, *The Seybold Report*, 10(16):3–18, April 27, 1981.

SEYB84a Seybold, Jonathan, Apple's Macintosh & Lisa II: The Alternative to IBM, *The Seybold Report on Professional Computing*, 2(6), February 6, 1984.

SEYB84b Seybold, John W., *The World of Digital Typesetting*, Seybold Publications, Inc., Media, PA, 1984.

SHEI83 Sheil, B.A., Power Tools for Programmers, in *Interactive Programming Environments*, McGraw-Hill Book Co., New York, 1984.

SIEG85 Siegel, David R., *The Euler Project at Stanford*, Department of Computer Science, Stanford, California, 1985.

SIMO28 Simon, Oliver and Julius Rosenberg, *Printing of Today*, 1928.

SIMO45 Simon, Oliver, *Introduction to Typography*, Faber and Faber, London, 1945.

SMIT75 Smith, David Canfield, *PYGMALION: A Creative Programming Environment*, Department of Computer Science, Stanford University Technical Report No. STAN–CS–75–499, June 1975.

SMIT82 Smith, David Canfield, et al., Designing the Star User Interface, *BYTE Magazine*, April 1982.

SOUT85 Southall, Richard, *Designing new typefaces with Metafont*, Department of Computer Science, Stanford University Technical Report No. STAN–CS–85–1074, September 1985.

SOUT95 Southward, John, Machines for Composing Letterpress Printing Surfaces, *Journal of the Society of Arts*, December 20, 1895, pp. 74–82.

SPAN73 Spangler, Kathleen, New Technologies Challenge Classical Typography, *Technological Change in Printing and Publishing*, Hattery and Bush, ed., Hayden Book Co., New Jersey, 1973.

SPEN69 Spencer, Herbert, *The Visible Word*, Hastings House, New York, 1969.

STAL81 Stallman, Richard M., EMACS: The Extensible, Customizable, Self-Documenting Display Editor, *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, June 1981, pp. 147–156.

STEI55 Steinberg, S.H., *Five Hundred Years of Printing*, Penguin Books, Harmondsworth, Middlesex, 1955.

STON79 Stone, Sumner, Hans Eduard Meier's Syntax Antiqua, *Fine Print*, 5(4):120–123, October 1979.

SUTH63 Sutherland, I.E., *Sketchpad: A Man-Machine Graphical Communication System*, Massachusetts Institute of Technology Lincoln Laboratory, Technical Report No. 296, January 30, 1963.

THAC79 Thacker, C.P. et al, *Alto - A Personal Computer*, Technical Report No. CSL 79–11, Xerox Palo Alto Research Center, August, 1979.

THOM36 Thomas, David, *Type for Print*, 1936.

THOM72 Thompson, John Smith, *History of Composing Machines*, Arno Press, New York, 1972.

TIME30 The Times, *Printing in the Twentieth Century, a survey*, The Times Publishing Co., London, 1930.

TIME32 The Times, *Printing the Times, a record of the changes introduced in the issue for Oct. 3, 1932*, The Times Publishing Co., London, 1932.

TRAC81 Tracy, Walter, The Types of Jan van Krimpen, Part I, *Fine Print*, 7(2):51–56, April 1981.

TREU76 Treu, Siegfried, A Framework of Characteristics Applicable to Graphical User-Computer Interaction, *User Oriented Design of Interactive Graphics Systems*, ACM/SIGGRAPH Workshop, October 1976, pp. 61–71.

TSCH48 Tschichold, Jan, *An Illustrated History of Writing and Lettering*, Holbein Publishing Company, Basle, 1948.

TSCH51 Tschichold, Jan, *Designing Books*, Wittenborn Schultz, New York, 1951.

TSCH52 Tschichold, Jan, *Meister Buch der Schrift*, Otto Maier Verlag, Ravensburg, 1952.

TSCH66 Tschichold, Jan, *Treasury of Alphabets and Lettering*, Reinhold Publishing Co., New York, 1966 (English translation of *Meister Buch der Schrift*).

TSCH67 Tschichold, Jan, *Asymetric Typography*, Reinhold Publishing Corporation, New York, 1967.

UNGE79 Unger, Gerard, The Design of a Typeface, *Visible Language*, 13(2):134–149, 1979.

UPDI37 Updike, Daniel Berkeley, *Printing Types: Their History, Forms and Use*, Harvard University Press, Cambridge, 1937.

VANK57 van Krimpen, Jan, *On designing and devising typefaces*, New York, The Typophiles, 1957.

VANK72 van Krimpen, Jan, *A letter to Philip Hofer on certain problems connected with the mechanical cutting of punches*, Cambridge, Mass., Harvard College Library, 1972.

VANW80 Van Wyk, Christopher John, *A Language for Typesetting Graphics*, Department of Computer Science, Stanford, University, Technical Report No. STAN-CS-80-803, June, 1980.

VIRG57 Publii Virgilii Maronis, *Bucolica, Georgica, et Aeneis*, Johannis Baskerville, Birmingham, England, 1757.

WAKE67 Wakefield, R.J., Print Layout and Design with a Computer CRT System, *Journal of Typographic Research*, 1(2):165–168, 1967.

WARD35 Warde, Beatrice, Cutting Types for Machines: A Layman's Account, *The Dolphin*, Limited Editions Club, New York, No. 2, 1935.

WARD53 Warde, Beatrice, *Quotations from the Writings of Beatrice Warde*, Lanston Monotype Company, Philadelphia, 1953.

WARD56 Warde, Beatrice, *The Crystal Goblet*, The World Publishing Company, New York, 1956.

WARN80 Warnock, John, The Display of Characters Using Gray Level Sample Arrays, *SIGGRAPH '80*, 14(3):302–307, July 1980.

WARN82 Warnock, John, and Douglas K.Wyatt, A Device Independent Graphics Imaging Model for Use with Raster Devices, *SIGGRAPH '82*, 16(3):313–319, July 1982.

WEIN81 Weinrab, D. and Moon, D., *Lisp Machine Manual*, Massachusetts Institute of Technology, 1981.

WILK84 Wilkes, A.J., et al., The Rainbow Workstation, *The Computer Journal*, 27(2):112–120, 1984.

WINO73 Winograd, Terry, Breaking the Complexity Barrier (Again), *Proceedings of the ACM SIGPLAN/SIGIR Interface Meeting on Programming Languages-Information Retrieval*, November 1973, pp. 13–30.

WISE78a Wiseman, N.E., C.I.C. Campbell, and J. Harradine, On making graphic arts quality output by computer, *The Computer Journal*, 21(1):2–6, 1978.

WISE78b Wiseman, N.E., *Use ELF for your Elfabets*, University of Cambridge Computer Laboratory, Rainbow Memo 162, July 18, 1978.

WISE80 Wiseman, N.E., *Computing Aspects of Text and Graphics Processing*, University of Cambridge Computer Laboratory, 1980.

WROT38 Wroth, Lawrence C., The Eighteenth Century, *The Dolphin*, Limited Editions Club, New York, (3):208–232, 1938.

ZACH65 Zachrisson, Bror, *Studies in the Legibility of Printed Text*, Almqvist & Wiksells Boktryckerie AB, Stockholm, 1965.

ZAPF52 Zapf, Hermann, *Pen and Graver*, Museum Books, New York, 1952.

ZAPF65 Zapf, Hermann, *Hunt Roman, the birth of a type*, Pittsburg Bibliophiles, Pittsburg, 1965.

ZAPF68a Zapf, Hermann, Changes in Letterforms due to Technical Developments, *Journal of Typographic Research*, 2(4):351–368, 1968.

ZAPF68b Zapf, Hermann, The Many Faces of Zapf, *Lithopinion*, 3(2):57–65, 1968.

ZAPF70 Zapf, Hermann, *About alphabets: some marginal notes on type design*, Cambridge, Massachusetts, MIT Press, 1970.

ZAPF83 Zapf, Hermann, Zapf on Tomorrow's Type, *Communication World*, November, 1983.

This dissertation is printed in Lucida,
designed by Charles Bigelow and Kris Holmes,
and the first typeface family created for laser printers.
It was printed on an Imagen 8/300 printer.