1981

The use of the spline function in computer-aided type design

Kathleen Barry Albertini

Rochester Institute of Technology
School of Computer Science and Technology

The Use of the Spline Function in Computer-aided Type Design

A thesis submitted in partial fulfillment of the
Master of Science in Computer Science Degree Program

by
Kathleen Barry Albertini

Approved by: Guy Johnson

Guy Johnson

Stuart Hirshfield

Stuart Hirshfield

Peter Lutz

Peter Lutz

Archibald Provan

Archibald Provan

23 May 1981

Table of Contents

Acknowledgements

I would like to thank my thesis advisor, Guy Johnson for his expertise, his time and his patience. In addition to their insights on letterforms, Archie Provan and Hermann Zapf provided me with a great deal of background information on type design. Peter Lutz and Stuart Hirshfield provided help and critcism. Jim Jensen and Doug Sargent provided the necessary equipment to produce the hard copy output for Appendix C. My husband, John, not only introduced me to calligraphy and the appreciation of letterforms but also provided support throughout the thesis. Finally, I would like to thank the unofficial committee Lois Burton and Arn and Dorothy Albertini.

Abstract

The traditional type designer works with pen, ink and paper. The characters that he designs go through many changes before the whole font is finalized and set in the printing medium. Further alterations to the font may occur after he prints and evaluates a sample piece of text. Computer-aided type design presents the type designer with a new design tool. Computers have been used in the printing industry for typesetting; the automation of type design is a logical next step. Freeing the designer from the repetitive tasks of the design cycle is one obvious advantage of using computers. But computer-aided type design makes its real contribution when a designer can use it with the same facility that he uses traditional design tools like pen, ink and paper.

This research investigates the use of the spline curve in type design. Contemporary type designers are expected to perform a variety of tasks. On the one hand, they may design characters that will last because of their artistic merit and singular characteristics. On the other hand, they may design characters that can be recognized by machines. A computer-aided type design tool should be flexible enough to meet these demands and convenient to use. The focus of the research will be on the input requirements for drawing the spline curves that form the characters. First, the mathematics of the spline functions and their use in two dimensional graphics is discussed. Next, a brief survey of the use of the spline curve in computer-aided type design

is given. In order to investigate further the problems of designing characters, a pilot system was written to design basic character shapes, using raster based graphics. The system was designed with the objective of having input specifications which would be familiar to a type designer. This system is described in the last section.

Introduction

Typography
I am
only a poet
and you are
the flowery
play of reason,
the movement
of the chess bishops
of intelligence. ...

Pablo Neruda
from Ode to Typography

Letterforms sometimes convey information without calling attention to themselves. In other contexts the letterforms themselves draw attention before the information they convey is assimilated. The attributes of a detergent's cleansing power, the particulars of an upcoming concert, the daily news all are read and assimilated without drawing conscious attention to the different forms an A or a K can take and still be recognized as an a or a k. Type designers through the centuries have labored over letter design to allow this unconscious recognition while conveying the information contained. Readability is an abstract quality that is influenced by a single letter and "all acceptable variations" [COX 78,p.430] and the relationship of letters to each other visually. This relationship of letters to each other is in turn influenced by inter-letter spacing, size and weight of the letters and the "style and form of groups within a single

type font" [COX 78,p.430].

A type designer will have many, often conflicting, requirements in designing a typeface. Some requirements are fairly easy to quantify such as whether the typeface will be used primarily for text or display purposes, whether the purpose of the typeface may be to draw attention to itself, or whether the typeface must be read by a machine. Many of the requirements that relate to the printing medium (the press, the paper and the ink [PROVA 80]) are also quantifiable. When artistic or qualitative requirements are then added to the type design goals, the complexity of the type designer's task becomes evident. Hermann Zapf, a type designer involved in several computer-aided design projects, frequently describes the many tasks a type designer confronts especially in relation to changing technology. "The designer of a new typeface has a responsibility, not only to the past masters of type design - ... but also to the future" [ZAPF 65,p.23].

This thesis will consider the use of computers in aiding the design cycle for the type designer. Specifically of interest is the use of the spline function which allows the letterforms to be quantified and stored easily in a form usable by computers. The use of computers in the design process has the obvious advantage of eliminating many tedious design steps which are often done by technical assistants to the designer. For instance, once a designer has drawn the final version of a character, a technician then cuts the letter in rubylith. Rubylith, a brand name, is "a

red film which is coated over a plastic base" [SILVE 73,p. 63]. When a letter is cut in rubylith, it may be used as a photographic negative. The technician may cut a character several times before he is satisfied that the rubylith matches the original drawing of the character. Rubylith can be cut under computer control using the same drafting machine that can reproduce the drawings of characters [IKARU]. Machine cutting of rubylith will reproduce the letter accurately without any recuttings, unless further changes are made to the letter. In addition, uses of functions such as the spline function in conjunction with sophisticated computer graphics hardware and software give the designer a powerful tool for reproducing letter shapes.

In Section 1 the spline function and its use in two dimensional graphics will be discussed. This discussion will include mention of some of the variations on the spline function. Section 2 will discuss the use of the spline function in computer-aided type design. In this section systems being used in industry today will be considered. Section 3 will describe, TYPDESNGR, a pilot system written in PASCAL for the APPLE microcomputer. TYPDESNGR attempts to provide a design tool that is oriented toward the type designer. The special emphasis in this pilot project was on the input requirements for designing letter shapes.

Section 1: Spline Functions in Two Dimensional Computer Graphics

The "fundamental spline function" [SCHOE 69,p. 221] was introduced by a mathematician named I. J. Schoenberg in 1947. As the number of uses for Schoenberg's spline function has increased so have the variations on the original function. B-splines [GORDO 74] and Bezier curves [BEZIE 74] are two of the better known examples but the list is extensive: periodic splines, L-splines, natural splines, Q-splines, quasi splines, splines in tension, non-linear splines, encastered splines, C-splines and so on. Since its introduction, Schoenberg's spline function and its many variations have been used extensively in approximation theory [SCHOE 69], numerical differentiation [AHLBE 67], and more recently in computer graphics and computer-aided design. The concept of the mathematical spline function is derived from a tool, a loftsman's spline, used in shipbuilding. "A physical spline is a long narrow strip of wood or plastic used by a loftsman to fair in curves between specified datapoints. The splines are shaped by lead weights called 'ducks'" [ROGER 76,p.119]. In designing the hulls of ships the loftsman's spline had to be flexible in order to meet the constraints of the designer's shape. The ducks allowed the designer to control the shape of the curve as it passed through the designated datapoints.

The definition of the spline function follows the physical concept of the loftsman's spline. The analogies between the

loftsman's spline and the mathematical spline are fairly obvious. While the mathematical terminology varies somewhat throughout the spline literature, the following definitions will be applied in the present discussion. The knots and control points of the mathematical spline correspond to the datapoints and lead weights of the loftsman's spline. Knots are points which lie on the spline curve. Control points may or may not be on the curve depending on the definition of the function. In general, control points are placed in such a way that the curve's path is influenced by them. Curves that are defined by control points will often contain some kind of blending function as a part of the function's definition. The way that blending functions are chosen will describe the kind of influence a control point will have on the curve. For some curves, such as the basic spline function, the knots and control points are the same set of points. A general definition of a mathematical spline function is "a piecewise polynomial of degree K with continuity of derivatives of order K-1 at the common joints between the segments"[ROGER 76,p.119]. Joints are formed at the knots as the separate polynomials are pieced to form the composite curve.

The equation for the basic spline function is:

$$P(t) = \sum_{i=1}^{k} a_i t^{i-1} \qquad t_1 \leq t \leq t_2$$

The equation is expressed in parametric form. A vector P[x,y] would be represented by two equations of the same form:

$$x(t) = \sum_{i=1}^{k} a_i t^{i-1} \qquad t_1 \leq t \leq t_2$$

and

$$y(t) = \sum_{i=1}^{k} a_i t^{i-1} \qquad t_1 \leq t \leq t_2$$

K is the degree of the equation. The parameter t varies over intervals t1 to t2, t2 to t3 and so on. These intervals form the pieces of the curve.

The simple definition has several key words which reveal the flexibility of spline functions for computer graphics. The fact that the spline function is a polynomial implies that a curve produced from such a function could take on many shapes. Polynomials of one variable can produce a wide variety of curves. The curves that are produced depend on the degree of the polynomial and the coefficients of its terms. The fact that the polynomials are pieced allows the design of complex shapes with twists and turns that could not be achieved with a single function. When appropriate choices of the parameter range are made, coefficients can be controlled to produce minimum curvature. The requirement that the piecewise polynomial have K-1 continuous derivatives will also influence the curvature at the joints when the spline is cubic [NEWMA 79,p.313]. The curvature may vary inbetween the joints. Second order continuity restricts the way the curve is pieced by eliminating joints where two curves simply meet (zero order continuity) or are only tangent (first order continuity).

The spline function is appealing for use in computer graphics because of its flexibility in producing a variety of shapes

while also providing control over this flexibility. A particular spline function is chosen for an application based on an appropriate balance between flexibility and control. The concept of smoothness is closely related to achieving this balance. A very general definition of a smooth curve is one that can be controlled to the extent that unwanted wiggles or deviations do not appear. The balance between flexibility and control is achieved when the curve's path is not severely limited by controlling the deviations. Knuth, in discussing smoothness for METAFONT's curves, simply eliminates sharp corners in his general definition of smoothness [KNUT1 79,p.23]. The search for smoothness certainly accounts for some of the variations on Schoenberg's original spline function. Obviously, a general definition of a smooth curve allows the type designer, for instance, to be unhappy with a curve that the ship designer calls smooth.

Rogers and Adams recommend the use of the cubic spline because of the reduced computational requirements, and they also suggest that higher ordered splines are subject to "numerical instabilities" which may cause "undesirable wiggles" [ROGER 76,p.119] when several segments are connected. Chasen suggests that specifying too many points will also increase the number of oscillations in the curves [CHASE 78,p.18]. Another source of oscillations may result when a long interval between knots is followed by a short interval [AELBE 67,p.50]. Smoothness, then, will be affected by the degree of the polynomial. Following the

recommendation of Rogers and Adams this discussion will focus on cubic spline functions, eliminating one source of oscillations. In addition, smoothness will be effected by the number and placement of the knots or control points.

In discussing the relationship of knots and control points to the resulting curve, several more definitions are necessary. Relying on the definitions already developed, an interpolating spline is a spline whose control points are the same as the knots [CATMU 74,p.317]. An interpolating spline, then, will pass through each control point. On the other hand, an approximating spline may not pass through the control points. In general, the control points and the knots will be distinct, although they may coincide at some points on the curve such as the beginning and end [CATMU 74,p.317]. The standard cubic spline is interpolating while the B-spline is an approximating spline function. Some of the obvious advantages of an interpolating spline are in ease of defining the curve, especially when the exact shape of the curve is known. A designer can specify points which correspond exactly to the edge of the shape he is creating. This feature is especially useful when included in a design tool which will be used by a person with minimal knowledge of spline functions or computers for that matter. On the other hand, the designer must have certain additional knowledge to specify the design for an approximating spline function. The designer designates the positions of control points which frequently will not be on the designed

curve; rather, depending on the function used, the control points will be placed in the vicinity of the final curve. While explicit knowledge of the mathematical functions involved is not necessary, certain techniques must be acquired for getting desired curves from these control points. In an application where the final design is only approximate, an approximating spline may be useful.

In order to take advantage of the capability of allowing the designer to specify knots in an interpolating spline function, all of the knots must be specified before any segment of the curve can be generated. The enforcement of the continuity of the second derivatives requires this [HAZON 79,p.165]. In an application where a great deal of interaction is required in designing the curve shapes, there is an obvious advantage to choosing an approximating spline, like a B-spline, where the curve may be defined incrementally [HAZON 79,p.165]. The designer may specify part of the curve, modify it, place additional control points and then continue the cycle. Such interactive design is further eased when the function can be controlled locally. Local control [NEWMA 79,p.312] allows the designer to move a control point and the curve will be effected only in the area immediately surrounding the control point. With global control, moving one control point effects the shape of the entire curve. B-spline functions have local control [NEWMA 79,p.322] while the cubic spline function and Bezier functions [NEWMA 79,p.316] have global control.

1.1 The Cubic Spline Function Equation

The equation for a cubic spline segment expressed in parametric form is:

$$P(t) = \sum_{i=0}^{3} a_i t^i = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

As noted above, P(t) is the position vector P[x,y] with coordinates x(t) and y(t) [ROGER 76, p.120]. There are significant advantages to expressing the equation in parametric form. Parametric form makes the curve independent of the axis on which it is drawn [ROGER 76,p.93]. So curve segments generated in parametric form can be manipulated without altering the shape of the curve segment. In addition, the curve length is fixed by the range of the parameter, t [ROGER 76,p.93]. This characteristic simplifies the calculations for curve generation. Curve pieces can be generated over the same parameter range and then pieced together. Choosing the beginning of the range, t1, to be equal to Ø further simplifies calculations. The choice of the parameter range will determine coefficients and the smoothness of the curve.

Through experimentation with the parameter range, t1 to t2, the coefficients of the equation, ai, are determined to produce the desired smoothness. The coefficients for a curve segment are determined by the two end points of the segment and the tangent vectors at each of those points [ROGER 76,p120]. The equations and boundary conditions lend themselves to matrix manipulation,

and matrix manipulation is especially useful when many points must be specified to define the total curve. When not all tangent vectors are known, coefficients may be determined by working with the two end points, their tangent vectors, and the inbetween points.

The basic cubic spline function produces a variety of shapes based on the placement of the points, the choice of parameter range, and the tangent vectors at the points. The variations on the spline function employ constraints such as blending functions which are designed to create different kinds of control at or surrounding the control points.

1.2 Some Variations of the Basic Spline Function

The many variations on the basic spline function have evolved from a desire or need to expand some characteristic of the spline function to suit a particular application. The cubic spline allows placement of control points which are the knots of the composite curve. All of the points must be placed before the curve is drawn and moving one point effects the shape of the whole composite curve. The requirement that second derivatives exist for the curve pieces offers the possibility of controlling the curvature at the joints. Rogers and Adams use the second derivative information to provide the solution of a group of equations where only the beginning and end slopes are known

[ROGER 76, p.122]. Having the second derivative information allows unique slopes to be determined at each of the inbetween points. The advantages and disadvantages of some of the spline alternatives can be seen by looking at a few of their characteristics.

B-spline curves and Bezier curves offer the main advantage of allowing more interaction in the design process because they do not have the restriction of designating all control points before the curve can be drawn. The use of B-spline functions allows the designer the advantage of local control of the curve shape. By using a set of blending functions or B-spline basis functions the curve can be controlled in a small neighborhood surrounding each control point [NEWMA 79,p.321]. Moving a point will only alter the curve in the neighborhood of that point. But, the control points do not, in general, coincide with the knots of the curve. It is possible to repeat control points in defining a B-spline curve. The repeated control points tend to make the curve pass through or close to that repeated point [ROGER 76,p.147]. Repeating control points at the beginning and end of the curve guarantee that the resulting curve will pass through these points. B-splines also have analogous derivatives which guarantee K - 1 continuity based on the degree, K, of the polynomial.

Bezier curves were developed by P. Bezier for automobile design at Renault [BEZIE 74]. Rogers and Adams [ROGER 76,p.139]

argue that Bezier curves offer more flexibility because the derivatives of the cubic spline curve fitting techniques do not offer the intuitive "´feel´" that Bezier curves provide. This intuitivé ´feel´ is in the same category as a smoothness criterion (ie. different people will evaluate it differently); and while that quality may have been Bezier´s objective in designing the Bezier curves, there still are design applications for which Bezier curves are not appropriate. Bezier curves are determined by blending functions which correspond to the control points for the curve [NEWMA 79,p.316]. The blending function associated with a particular control point asserts the most influence in the neighborhood of that control point. But Bezier curves, like the basic spline curve, have global control and not local control, as Newman and Sproull define local control [NEWMA 79,p.316].

One of the appealing characteristics of the Bezier curves is the fact that different curve segments may be of different order [ROGER 76,p.143]. More complex shapes can be included in the composite curve by specifying more control points in this interval which will produce a higher order polynomial. Since there is no requirement of the existence of derivatives [ROGER 76,p.141], the designer may give up continuity to achieve some of the complex shapes.

At Renault the design process starts with pencil and paper. Once the curve is drawn, it is divided into sections and four points are associated with each section. The first point marks

the beginning of the curve and the second point is placed on the tangent line from the first point at the vertex of a polygon which is enclosing this section of the curve. The third and fourth points are the analogous points at the end of the curve section. A milling machine is passed over these points and the machine draws a curve. The designer may then alter the machine drawn curve by modifying the slopes and the machine will redraw the curve section [ARMIT 71,p.1192].

The generation of different spline functions continues as more applications present new problems which are not adequately solved by the existing functions. Much of the most recent literature stresses the goals of computational efficiency and ease of storage and handling [TIMME 80,p.25] [CHASE 78,p.25] as influences on the modification of a particular function. Hazony [HAZON 79] developed algorithms for quasi cubic splines and Q-splines following the influence of the hardware development of array processing. Moss and Lindgard [MOSS 79] developed integer arithmetic algorithms for creating spline curves for implementation on microprocessors. The algorithms are illustrated by drawing alphanumeric characters. The spline in or under tension was developed to remove "extraneous inflection points in the curve" [CLINE 74,p.218] by incorporating a tension factor in the polynomial. When the tension factor is very large, the spline approaches a line; and no tension factor produces the spline curve itself. Catmull and Raphael [CATMU 74] describe a spline

function which exhibits local control. And research continues to adapt spline-like functions for particular design problems.

Section 2: Uses of the Spline Function in Type Design

> Every work of Architecture, every work of Typography, depends for its success upon the clear conveyance of intentions, in words and otherwise, from one human mind to others: from the man who is supposed to know how the finished thing should look and function, to a concert of specialists who are responsible not only to the master-designer but also to the public. [WARDE 56,p.67]

The process of designing typefaces has changed as technology has changed. The technology of the printing industry has evolved from metal impressions to photographic impressions, to digital storage of typefaces and laser printing. Yet, letterforms that are exquisite in metal may not retain this quality when transferred to a photographic medium. And letterforms that have been designed specifically for phototypesetters or digital typesetters reflect new and different design demands. The type designer today continues to search for a "different kind of letter: one suited to the electronic age of typesetting..."[LATHA 77,p.189].

Type designers have always had to keep in mind the many requirements that a typeface must fulfill. Some requirements are less quantifiable than others and can only be judged after the typeface has been in use for a period of time. Perhaps one of the most difficult requirements is meeting the design goals for a typeface specification. In the 1700s, for instance, Caslon was hired to design an "exotic" [PROVA 80] typeface. When he

presented the drawings, the typeface was rejected. Instead Caslon was instructed to create a typeface based on the way he had signed his name on the drawing. Where artistic merit is essential for one face, uniqueness or novelty (e.g. Ad Lib by Freeman Craw) may be the primary goal for another. Other typefaces, such as those used in newspapers, must be legible, distinct to the eye, and readable, easy to read under perhaps less than ideal reading conditions.

Additional requirements are dictated by the medium in which the typeface will be used. These requirements are listed and addressed more easily. For instance, when using photographic printing processes typefaces have to remain in tact despite unfavorable production conditions. The designer may experiment with conditions like under exposure or light inking to see how well the letter forms hold together [LATHA 78, p. 191]. The experimentation may suggest changes to the letterforms to make them more durable throughout the photographic printing process. Many of these techniques are well documented - anticipating ink-traps within a letter or making adjustments to the design size (so that reductions or enlargements of the designed letters retain their intended shape at all sizes which they will be used [LATHA 77, p.198]). The type designer must balance the many quantifiable technical requirements of the medium with the less quantifiable requirements mentioned above.

In designing Demos, a typeface designed for a digital typesetter, Gerard Unger refers to the medium requirements as "the necessary restrictions of technology" [UNGER 79,p.134]. Unger implies that the restrictions of the machine determined his design: "Having determined the most favorable curves ... all determined through a series of tests - I started to make the final drawings." [UNGER 79,p.140]. There is a thin line between restrictions and requirements and the danger is that the designer will let the technical requirements become restrictions. Another danger is that the engineer designing the typesetting equipment designs it awkwardly, creating unnecessary restrictions for a type designer. If restrictions determine the type design, less consideration may be given to the requirements of readability, legibility and artistic quality.

The designer must always maintain a perspective which will allow evaluation of both types of requirements. F.W. Goudy designed for the Monotype Company and he described the balance that the designer must maintain to keep the design his own:

> Yet it is important to know where hand-work should end and where machine-work should begin, and especially to see that the facility of the machine does not tend to usurp or replace any of the important functions of creation and representation... its beauty or legibility is determined by the eye, and not by means employed for its production. [GOUDY 33,p.22]

Type design presents problems of shape design for which the spline function provides solutions. The type designer needs a

tool that can reproduce essentially anything that can be drawn with pen on paper. This requirement is important whether the task at hand is to design a new typeface or to adapt an existing typeface for different typesetting equipment. Spline functions provide this capability. Since an original drawing for a character may go through many revisions before it is finalized, computers are an obvious aid in this repetitive process. Significant progress has been made in the automation of the type design process, in some instances with the aid of spline functions.

The following examples illustrate some of the automated type design tools being used today. A group at the University of Cambridge in England has developed an interactive system for type design which is not based on "mathematical formulations" [PRING 79, p.63]. The designer works with a light pen and the keyboard to specify and manipulate the letterforms interactively. The system is used mainly to design new typefaces. Xerox Corporation uses spline curves to store copies of existing typefaces. Parametric cubic splines with arc length parameterization are used to define outlines of letterforms. The existing letterform is put into raster form through a link between a TV camera and an Alto. Points from the raster are chosen interactively to form the outline. Spline curves are fit to the points and correcting of the letters can then be done interactively [RAMSE 80]. Mergenthaler Linotype Company, Autologic Incorporated and Compugraphic use Ikarus a program written at Unternehmensberatung

Rubow Weber (URW), a firm with experience in the shipbuilding industry. Ikarus uses spline functions to store letter outlines [IKARU]. Ikarus has been used both to design original type faces and to copy and/or adapt existing typefaces for new typesetting equipment. METAFONT is a program which was written by Donald E. Knuth to design typefaces. The designer specifies a series of points (and optionally tangent vectors) to which spline curves are fit in the form of letter strokes [KNUTH 79]. The specifications are then compiled to produce the letterform on the chosen offline device. The following sections will look briefly at Ikarus and METAFONT.

2.1 Ikarus

Compugraphic, a firm that designs typefaces for use on their own typesetters, purchased Ikarus in 1978 [HAWKI 80]. Compugraphic finds Ikarus attractive because they are able to produce a greater number of typefaces and because the generation of the other members of a type family is greatly eased once the main typeface is designed. Some of the attractive design features of Ikarus are: its capability of designing families of type from one basic design; a menu facility which allows common features in a typeface like serifs to be designed once and then applied throughout the typeface; and the high degree of accuracy that the drafting machine has (+ or - 2/10,000 of an inch) [HAWKI 80].

At Compugraphic the Ikarus system is used in the following way. A type designer creates a new typeface in 250 point size (approximately 3.4 inches) with pencil and paper. When the typeface is completed, the individual drawings of each letter are sent to the Ikarus department. A technician marks each letter along the outline with a set of symbols which indicate: starting point, edge points, curve points, and tangent points [IKARU]. During the marking process, the technician will determine whether common features in the typeface exist for which the menu facility may be used. The marked drawings are then placed one at a time on a drafting table which contains a digitized grid. An electronic sensor is positioned over each mark on the outline of the letter and a button is pushed indicating which mark is under the sensor. The information (coordinates of the points and the type of point) is stored on a diskette. The Ikarus system runs on a PDP11/34 which then reads the diskette and fits spline curves to the letter outlines. At this point the technician may redraw the letters on a Techtronix scope or on the Aristo drafting machine to examine for gross errors. The typeface is finally cut on the Aristo machine in rubylith. This output is then returned to the designer for corrections [HALEY] [HAWKI]. Frequently, the corrections are made to the rubylith but some corrections are made with pencil on the original and the letter is redigitized and re-cut in rubylith.

Similarly, at Autologic Incorporated, Ikarus is used primarily as a digitizing tool [STONE 80]. On the other hand, Mergenthaler Linotype Company installed an Ikarus system in the summer of 1980 [BERLO 80]. While their experience with Ikarus is brief, they have treated the Ikarus system as primarily a design tool rather than a digitizing tool. Ikarus is used for digitizing as well, but Mergenthaler places a great deal of emphasis on how Ikarus is used before the letter is finally digitized. The system is used by type designers, and type designers are therefore involved through all stages of Ikarus's use. Mergenthaler's use contrasts with Compugraphics' use of Ikarus where the designer hands the drawings to a technician who does not necessarily have any experience in type design. The designer does not see the letter again until the Ikarus output (drawings or rubylith) is produced. Mergenthaler will use Ikarus primarily to convert existing typefaces for Mergenthaler equipment. In addition, Matthew Carter designed Galliard for Mergenthaler using Ikarus [BIGEL 80,p. 28]. Galliard was designed before Mergenthaler got their own Ikarus system.

2.2 METAFONT's Use of a Spline Function

Donald E. Knuth has had little or no experience in designing letterforms. He created METAFONT as a result of his frustration with the publishing of one of his books in the computer programming series (the second edition of volume 1) [KNUT1 79,p. 17].

METAFONT was created, along with Tex (Theta Epsilon Chi), so that Knuth could control the layout of his books by presenting the publisher with machine readable text. "METAFONT is a system for the design of alphabets suited to raster-based devices that print or display text" [KNUT3 79,p. 1]. The name Tex was chosen to emphasize its Greek root whose meaning is a combination of art and technology [KNUT2 79,p. 4]. Tex is a document preparation program with many features (such as spacing, including illustrations, use of different fonts etc.) that allow it to produce a manuscript that is ready for printing. Special attention is given in both Tex and METAFONT to the requirements for printing mathematical equations, formulas, diagrams etc. - or what a printer refers to as penalty text. In fact Knuth used METAFONT and Tex to design the typefaces and produce the camera ready text for the book Tex and METAFONT New Directions in Typsetting [KNUT3 79,p. 1]. Although Knuth's work on METAFONT has continued since the book was published [TUGBOA 80,p. 12] METAFONT is available for anyone who wants to use it. The book, though now somewhat out of date, contains much more detailed information on the spline functions, the user interface, and examples of METAFONT's use, than is available on Ikarus or other tools being used in industry today. Therefore, the following explanation of how the user creates letterforms and the spline functions used will also be more detailed than the discussion of Ikarus.

METAFONT is written in SAIL for a DEC-system 20 [TUGBOA 80,p. 3]. METAFONT is a declarative computer language [KNUT3 79,p. 2]. Each letter is defined by writing a program for that letter. The program is processed by METAFONT and can produce the output letter on several different kinds of output devices. The devices that Knuth used were chosen with a good deal of attention to producing high quality text. Characters may be output on an XGP or an Alphatype CRS [KNUT3 79,p. 68]. And whole fonts may be created in a form suitable for Tex input. While Tex and METAFONT were designed to work together, they can also be used independent of each other, and METAFONT is the primary interest here. The user's view of METAFONT will be described first. This description will include the input specifications for METAFONT. Then, the equations used to produce the letterforms will be described.

In order to design a letter shape, a designer must acquaint himself with some basic mathematical terminology and learn a computer language. Each letterform is defined in METAFONT with a separate program. Knuth suggests that the designer sketch the skeleton of the letter on a piece of graph paper and then choose points along the skeleton to which METAFONT will fit spline curves [KNUT3 79,p. 4]. Using a Cartesian coordinate system with some arbitrary origin the designer specifies the points that define the skeleton of the letter he has sketched. METAFONT uses a convenient naming convention for specifying the points [KNUT3 79,p. 4]. An x or y as the first character of the variable indi-

cates in which direction the measurement is taken. The number following the x or y indicates which point is indicated (x1 = 0; y1 = 0; defines point 1). In order to produce more than a skeleton of a letter METAFONT defines a series of pens and erasers [KNUT3 79,p. 22]. A pen or eraser shape may be circular, horizontal, vertical, a rectangle to the left or right of the current position or the designer may create a special shape [KNUT3 79,p. 22]. A cpen is a circular pen, for instance. A very simple METAFONT program would contain: the definition of the points that form the letter skeleton, a statement of what kind of pen will be used, and a draw command. With the draw command, the designer may specify the size or width of the pen (9 rasters for instance) and the order in which the points will be connected (9 draw 1..2..4..7..10;). The program is then compiled and run producing output on a raster based output device.

A designer would produce a fairly crude letter shape with the above simple program. In order to produce a more refined shape METAFONT provides the designer with additional arguments to the two commands cpen and draw as well as additional parameters which may be set globally or locally, and more commands. All of these allow the designer to produce a letter of high quality. In addition METAFONT provides the user with the capability of producing whole fonts. The user may write subroutines [KNUT1 79,p. 55] to describe common features within a font. The use of subroutines allows the user to call the subroutine instead of writ-

ing the same portion of code where the common part appears in each letter. The fonts may be produced at different weights by altering global parameters. In addition the designer may make individual changes to letters when the font is produced at different weights. Finally, the fonts can be produced in a form that Tex can use as input for creating documents [KNUT3 79,p. 68].

Of the many features in METAFONT the addition of direction information to the draw command will allow a discussion of the METAFONT spline function [KNUT3 79,p. 13]. Knuth allows the user to provide METAFONT with slope information by having the user specify the x and y directions at the point. Knuth suggests that the user draw a line at the point in the direction of the curve. Then, the user counts squares to indicate the x and y directions. The command: draw 1{50,40}..2..3..4..5{-50,-36} [KNUT3 79,p. 14] contains slope information for points 1 and 5. The slope information may be specified for any point in the draw command. And Knuth suggests that adding slope information to existing points is preferable to adding points, when the user wishes to improve the shape of the letter [KNUT3 79,p. 16]. When the user does not specify slope information at a point, METAFONT imagines a circle that passes through 3 points (the point before the point in question, the point and the point after). The slope of the circle at the point is taken.

The draw command directs METAFONT to create the spline pieces between the points that will form the letter. The METAFONT spline function is a cubic paramterized spline function. The coefficients are complex numbers [KNUT1 79,p. 24]. The resulting function is a complex number which produces a curve that is drawn in a complex plane with the parameter t varying from 0 to 1. The formula for a spline piece between points z1 and z2 is:

$$z(t) = z1 + (3t^2-2t)(z2-z1) + r\cdot t(1-t)^2\cdot d1 - s\cdot t^2(1-t)\,d2$$

d1 and d2 are the directions of the curve at z1 and z2 in normalized form. h is the angle at z1 and p is the angle at z2.

$$d1 = e^{ih}(z2-z1)$$

$$d2 = e^{-ip}(z2-z1)$$

The way r and s are chosen allows METAFONT to keep curves "in bounds" [KNUT3 79,p. 21]. Knuth calls r and s the "velocities" [KNUT3 79,p. 21] at z1 and z2. "... a large value of r means that the direction remains approximately equal to d1 for a long time after the curve leaves z1 and a large value of s means that the direction is approximately d2 for a long time before the curve reaches z2" [KNUT3 79,p. 21].

$$r = \left|\frac{2\sin(p)}{(1+|\cos(g)|)\sin(g)}\right| \qquad g = \frac{h+p}{2}$$

$$s = \left|\frac{2\sin(h)}{(1+|\cos(g)|)\sin(g)}\right|$$

METAFONT makes an additional check when r and s are computed to make sure that they are neither smaller than Ø.5 nor larger than 4. [KNUT3 79,p. 21].

The curves produced in Tex and METAFONT New Directions in Typsetting are evidence of METAFONT's versatility. Brady's experience with METAFONT and the resulting letter forms give evidence of how the designer can experiment with the letter form to improve the shape and test the form at different weights and shapes [BRADY 8Ø].

Section 3: TYPDESNGR

TYPDESNGR is a program for the design of letterforms. It was written in PASCAL for an APPLE II microcomputer [APPLE 79]. Input to TYPDESNGR comes from the keyboard and from a joystick. Output is presented on the 279 dots by 191 dots APPLE screen. TYPDESNGR does not attempt to solve all of the problems that need to be solved for a program that would be used in a production environment. The intention in writing TYPDESNGR was to give special attention to the input requirements for specifying letter shapes and to use spline functions to store letter strokes. Having investigated Ikarus and METAFONT in some detail and questioned a few people using the two systems [BERLO 80] [BRADY 80] [STONE 80] [HALEY 80] [HAWKI 80], consideration was also given to including features from both systems that were popular with its users. Two specific features that were incorporated in TYPDESNGR were the storage of letters as strokes (METAFONT) and interactive communication with the user (Ikarus). All of the people contacted were primarily type designers with little or no programming experience. So the reactions to using the two programs were from the designer's point of view. At Mergenthaler, where Ikarus is being used as a design tool, Berlow appreciated Ikarus's interactive facility even as he was using it to copy an existing typeface rather than design a new face. Brady had an opportunity to experiment with METAFONT in the summer of 1980 [BRADY 80]. He spent a week designing an uncial 'a' with the aid of one of

Knuth's graduate students. Brady expressed special interest in the use of the different shaped pens. TYPDESNGR stores letter strokes and draws these strokes using pens. While TYPDESNGR does not explore all the possibilities of what can be done with one designed letter, the idea of designing letters with the letter strokes is appealing because of its simplicity and because it allows the potential of manipulating the letter more easily than when the letters are stored as outline shapes (Ikarus). TYP-DESNGR is interactive. Judging from the Ikarus experiences and Brady's experience with METAFONT, the designer may be more involved with the design tool if it is interactive. The real evaluation of a design tool such as this must come from the type designer. So the more involved the designer is in its use, the more valuable the criticism.

When TYPDESNGR is executed, the user is presented with a welcoming message on the bottom line of the screen. The bottom two lines of the screen are used to communicate with the user. The bottom line contains information messages or error messages. The information messages contain arguments to the command that has been typed, offer the user choices (to change the angle or not), describe what the program is doing at points when it is incommunicative for a period of time (connecting the points) and notes when defaults have been taken. TYPDESNGR tries to make reasonable decisions when not doing so would cause the user to repeat a lot of typing. TYPDESNGR may make decisions which seem

completely unreasonable to the designer, so TYPDESNGR tells the user when these decisions have been made. This gives the user the opportunity to respecify the arguments and/or commands where necessary. Both error messages and information messages are 30 characters long. The message length was chosen to conserve memory space. Where possible in the 30 characters, the error messages try to be explicit about what the user must do to correct the noted error. Due to the message length, some messages contain abbreviated words. For instance, the user may get the message "?? OVER INSERT LIM - CONN FIRST". In this case LIM is an abbreviation for limit and CONN is an abbreviation for connect. The user has a brief description of the error and the solution is to connect the points and then continue inserting new points. All error messages begin with two question marks.

The line above the message line is used to echo what the user has typed. TYPDESNGR indicates that it is waiting for the user to type a command by placing an asterisk (*) on this line and issuing a tone through the APPLE speaker. The user indicates a command to TYPDESNGR by typing the command followed by a carriage return <CR>. After the welcoming message appears, a menu of commands appears in the upper right corner of the screen (see Appendix A: TYPDESNGR Menus). The format of the menus that appear in this area of the screen is fairly straightforward. The top line contains the menu name (sometimes abbreviated) and number. The four menus are: the specifications menu, draw menu,

joy menu and alter menu. The next line, or lines in the case of the joy and alter menus, contain a brief note about the menu. The rest of the lines contain the commands available from that menu on the left. On the right is a one or two word description of the command. The joy and alter menus contain additional information in between the lines that describe the commands. The commands from each of the four menus which may appear in the upper right hand corner of the screen will be discussed in section 3.1 Input Specifications for TYPDESNGR. The last area of the screen is the upper left hand corner of the screen. Through commands from the specification menu the user may specify a grid upon which the letter may be drawn and a pen to draw the letter with. This grid appears in the upper left hand corner of the screen. The user places points which lie along the spine of the letter by choosing commands from the joy menu and positioning the joystick in conjunction with these commands. The points are displayed within the grid. When the points have all been chosen and modified if necessary (through commands from the alter menu), the points may be connected using the pen chosen by the user to create the letter form. The letter may be saved before ending a session with TYPDESNGR to retrieve at some other time and worked on again.

3.1 Input Specifications for TYPDESNGR

The primary input device is the keyboard. The keyboard is used to enter commands, specify arguments to the commands, and

answer questions from TYPDESNGR. The joystick is the graphics input device. Although somewhat awkward to use (especially compared to a pen or pencil), it does allow the user to access positions on the screen by using his hand and eye instead of specifying coordinates. When the joystick is moved to the right or left, the user will see a dot moving to the right or left. When the joystick is moved down the dot will move up on the screen and vice versa. Because the joystick is awkward to use, input from the joystick is controlled by input from the keyboard. The user may move the joystick freely and then indicate it is in position by entering a command from the keyboard. Input from the keyboard will be discussed first. Input from the joystick will be discussed in connection with the commands on the joy menu.

TYPDESNGR commands are one or two characters long. While some commands have arguments, all commands, except the grid command, are ended by hitting the carriage return key. If the user discovers a typing error before he finishes typing the command, he may use the backspace key to backtrack to the error and then retype the rest of the command. TYPDESNGR commands are presented to the user on four different menus, two high level menus and two low level menus. The user may switch back and forth between the two high level menus, the specifications menu (menu 1) and the draw menu (menu 2). Commands on the high level menus are two characters long and some take arguments. When a high level menu is displayed in the menu area of the screen, the user may execute

a command that will cause the program to exit, conclude execution. The first low level menu, the joy menu (menu 2A) may be retrieved by issuing a command from the draw menu. The second low level menu, the alter menu (menu 2B), may be retrieved by issuing a command from the joy menu. When a low level menu is displayed, the user must issue two or three commands to cause TYPDESNGR to exit. Commands on the low level menus are one character long and most of the alter menu commands take arguments while most of the joy menu commands do not take arguments. The TYPDESNGR commands will be described as they appear on the menus with the exception of the commands that allow the user to retrieve a new menu and to exit from the program or current menu. These commands are repeated on more than one menu and will be discussed first. An example of a letter design session with TYPDESNGR is given in section 3.1.5.

The two commands that appear on several menus are change menu (CM or G) and quit (QT or Q). The change menu (CM) command occurs on both the specification menu and the draw menu. Issuing the change menu (CM) command when the specification menu is displayed causes TYPDESNGR to retrieve and display the draw menu. Issued from the draw menu, the change menu (CM) command causes the specifications menu to be retrieved and displayed. The joy menu also contains a change menu (G) command. When the change menu (G) command is issued from the joy menu, the alter menu is retrieved and displayed. The change menu commands take no argu-

ments. The quit (QT) command occurs on both the specification menu and the draw menu. When issued from either menu, TYPDESNGR will clear the APPLE screen, perform some cleanup routines and exit. The other form of the quit (Q) command occurs on both the joy and the alter menus. When quit (Q) is issued from the joy menu, TYPDESNGR will clear the menu area of the screen and retrieve and display the draw menu. The user may choose any command from the draw menu, including quit (QT). When quit (Q) is issued from the alter menu, TYPDESNGR will clear the menu area of the screen and retrieve and display the joy menu. If the alter menu is the current menu and the user wishes to exit the program, the user must issue quit three times (Q,Q,QT). The quit commands take no arguments.

3.1.1 The Specifications Menu

The specifications menu is the first menu that is presented to the user when TYPDESNGR begins execution. With commands from this menu the user describes the drawing tools. There are seven commands on this menu, including the change menu (CM) and the quit (QT) commands discussed above. Commands from this menu may be issued in any order. In addition to the two commands that define the pen size (PN) and the grid size (GD), there are three miscellaneous commands which will be described first. The user mode (UM) command allows the user to control the number of messages that TYPDESNGR issues. The user mode (UM) command may take

one of two possible arguments. By typing: UM EX <CR>, the user classifies himself as as expert. After the command is typed, the user will receive only error messages as TYPDESNGR continues execution. By typing: UM NO <CR>, the user classifies himself as a novice and will receive both error messages and information messages. When TYPDESNGR begins execution, novice mode is assumed. If the user mode (UM) command is never issued, the user will receive all messages. The two other miscellaneous commands allow the user to save (SS) and get (GS) the specifications that the grid (GD) command sets up. When the user types: SS <CR>, TYPDESNGR writes a file on APPLE2 (a floppy disk which should be placed in the lower of the two floppy drives) called SPEC.DATA. The file contains the information which is specified in the grid (GD) command. Typing: GS <CR> reads the file SPEC.DATA on APPLE2 and defines the grid parameters without the user having to reissue the command. These commands would be useful in drawing a whole alphabet. Each time TYPDESNGR is executed the get specifications (GS) command could be issued and then the user would be ready to draw.

The grid specifications (GD) command allows the user to specify a grid to define the area in which the letter will be drawn. This grid resembles the grid used in final proofs for International Typeface Corporation typeface designs. The markings on the grid provide the designer with a guide when he is drawing the letter. After the grid (GD) command is issued and

the arguments specified, TYPDESNGR will draw a box with the dimensions of the pointsize the user chooses. The top and bottom of the box will be marked with tic marks indicating the units per em that the user chooses. A baseline will be drawn across the box. And short lines will be drawn along the left and right sides of the box which indicate the descender length, x-height, capital height and ascender height. If the user does not issue a grid (GD) command, he may still draw a letter. The grid (GD) command simply provides a frame of reference.

The grid (GD) command has seven arguments. The user first types: GD <CR> to indicate that he wishes to use the grid (GD) command. TYPDESNGR then displays messages which remind the user what the seven arguments are. Then, the user is presented with an asterisk indicating that TYPDESNGR is waiting for the user to type the arguments. The arguments may be typed in any order. The argument format is: keyword=number. If all the arguments are typed, the typing will exceed what is visible on the user's screen. Arguments are separated by a comma, and a <CR> may follow the comma when the typing gets near the edge of the screen. When the last argument is typed, it is followed by a semicolon and <CR> to indicate that the user is finished specifying arguments.

The user may specify what pointsize he would like to draw at (PZ=number). Pointsize must be between 72 and 250. The pointsize argument determines the size of the box produced. The

user may specify units per em (UN=number). TYPDESNGR will accept 18, 36, or 54 for this argument. This argument produces the tic marks along the top and bottom of the box. The remaining arguments create the short lines along the left and right side of the box and the baseline across the box. These arguments are given in points. (BH=number) specifies how far up from the bottom of the box the baseline should be drawn. (XH=number) specifies the x-height of the letter. (DH=number) specifies the measurement from the baseline down to the desired length of the descender. (AH=number) specifies the ascender height and (CH=number) specifies the capital height. Ascender height and capital height are measured from the baseline.

If the user types simply: ; <CR> after specifying the grid (GD) command, TYPDESNGR will assume the following defaults: pointsize (PZ=250), units/em (UN=54), baseline (BH=80), x-height (XH=77), descender length (DH=77), ascender height (AH=154) and capital height (CH=140). If the user omits one of the arguments, TYPDESNGR will use the default value for that argument and notify the user that a default is being used (DEFAULTS USED FOR OTHER OPTIONS). All of the default values are based on 250 pointsize. After the user has typed the semicolon and carriage return indicating that he is finished specifying arguments, TYPDESNGR asks if the user wants to have the grid drawn (DRAW THE GRID?). If the user types: Y <CR>, TYPDESNGR will try to draw the grid in the drawing area of the screen. If the user has specified

options that conflict (a 300 point x-height for a 250 point letter) TYPDESNGR will notify the user (DIM CONFLICT WITH POINTSIZE). TYPDESNGR will not draw the grid and the user will have to reissue the grid (GD) command to resolve the conflict. If the user types anything but 'Y' to this question, the grid drawing will not be attempted and TYPDESNGR will wait for another command.

The pen (PN) command allows the user to choose the weight of the pen that TYPDESNGR will use to draw the letterforms. The pen is always the same shape - rectangular with the width greater than the height. The user may choose from four possible weights: light, medium, bold and a line connecting the points. The corresponding arguments are: LT, MD, BD, and LN. The command: PN BD <CR> would cause TYPDESNGR to place the largest pattern at each point it places on the screen. If the user does not issue the pen (PN) command, TYPLDESNGR will use a medium pen when it draws.

3.1.2 The Draw Menu

The user may retrieve the draw menu by issuing the change menu (CM) command from the specifications menu. The draw menu contains three new commands. At this point the change menu (CM) command would retrieve the specifications menu again and the quit (QT) command would cause the program to exit. The other commands

are save the letter (SL), get the letter (GL) and draw (DW). The save the letter (SL) and get the letter (GL) commands allow the user to work on the same letter at different sessions of TYPDESNGR. As with the specifications menu the commands may be issued in any order with some logical restrictions. There is no point in saving a letter (SL) that does not yet exist. If the user types: SL KT UA <CR>, TYPDESNGR will create a file on APPLE2: named KT.UA. Each argument must be two alphabetic characters long. The user may use the short arguments to signify some information about the letter stored. The first argument may be an abbreviation for the typeface name (Kathi's Type). The second argument may be an abbreviation for the character name (Uppercase A). The file KT.UA, in this example, would contain the points and associated angles that the user specified with the joy and alter menus. The save letter (SL) command makes sure that any inserted or deleted points are taken into account before writing the file to the floppy disk. The message (MERGING. . .) indicates that TYPDESNGR is searching for these points. The get letter (GL) command allows the user to retrieve a letter that has been saved with the save letter (SL) command. The arguments to the get letter (GL) command must be the same as to the save letter command. The command: GL KT UA <CR> would cause TYPDESNGR to retrieve the file KT.UA from APPLE2:. After the get letter (GL) command is issued, the user may issue the draw (DW) command to retrieve the joy menu and the connect (C) command from the joy menu to redraw the letter. The draw (DW) command takes no

arguments. When it is issued the joy menu is retrieved.

3.1.3 The Joy Menu

The draw command (DW) from the draw menu retrieves the joy menu. There are six new commands on the joy menu. The change menu (G) command will retrieve the alter menu. The quit (Q) command will retrieve the draw menu. The other commands allow the user to specify the points that define the letter shape (S,B,K,E,P) and connect the points (C). When this menu is presented, the user will be specifying input from both the keyboard and the joystick. This is the only menu where order is important, both the order that the points are chosen and the order in which the commands are issued. TYPDESNGR allows the user to define letter shapes by specifying points along the spine of the letter. The order that the points are given determines the order in which the points will be connected. Since it is easy to forget a point, the last menu, the alter menu, allows the user to add new points. For instance, if the user places three points on a stroke and feels that another point is needed between the first and second points, he should remember this but continue placing points which will come after the third point. When the letter is completed, he may change menu (G) and add the missing point by using the insert point (I) command from the alter menu. The order of the commands from the joy menu is important too. A letter is divided into strokes. Five of the new commands allow

the user to indicate the beginning (S) and end of the letter (P), the beginning (B) and end of each stroke (E) and the inbetween points (K). A stroke may not be started (B) until the letter has been started (S). The letter may not be ended (P) until the last stroke has been ended (E). And a new stroke may not begin (B) before the present stroke is ended (E). TYPDESNGR tries to prompt the user when commands are out of order so that the whole letter is not lost, but it is possible for TYPDESNGR to find the letter so confused that the user will have to start over. A message - "?? FATAL STROKE ERROR" will alert the user when this condition has been reached and TYPDESNGR will take the user back to the draw menu where he may start again. Since the order in which commands are issued is significant, the commands on the joy menu are numbered. In addition, the joy menu contains information about when to position the joystick and lists valid command sequences.

Before describing the commands, a general description of how the input from the keyboard and input from the joystick are coordinated will be given. Choosing points along the spine of the letter will be accomplished with ease after the user experiments with TYPDESNGR to see how the points are connected. TYPDESNGR will take points two at a time and choose the path between the points by calculating a spline function to determine the intervening points. Details of the spline function are given in Section 3.2. The user indicates the points to TYPDESNGR by posi-

tioning the joystick followed by a command from the keyboard (B,K,E) indicating what kind of a point it is. The user will have more control over the lettershape if the points are not placed too far apart. In general, points placed more than .5 inch apart will emphasize any stairstep effect produced in connecting the points. The .5 inch measurement is the result of the TYPDESNGR spline function. For some lettershapes where the letter is more angular, points should be placed further apart than .5 inch. The placement and distance between points depends heavily on how the curves of the letterform are changing. Curves that change very gradually do not require as many points to define them as curves that change alot over a small distance.

A small dot on the APPLE screen indicates the position of the joystick. In addition to locating points with the joystick, the user provides information about the curve direction at the point. After the point has been chosen, the user may specify the angle that a line tangent to the stroke at the point would form. When the shape of a stroke changes gradually, the angle at each point will also change gradually. The user is asked the question (CHANGE THE ANGLE AT THIS POINT?(Y,N)). If the user does not choose to change the angle at a point, he answers 'N' and the angle from the previous point is used. When the user begins a new stroke, the angle is set at Ø degrees. If Ø is appropriate for the beginning of the stroke, the user need not alter the angle. If the user wishes to change the angle at the first

point, he answers 'Y' to the question followed by a <CR>. Next he types a number between Ø and 359 indicating the angle desired. After the angle is determined, either through the user's specifying it or by using the angle from the previous point, TYPDESNGR draws a short arrow originating at the point and pointing in the direction indicated by the angle. The angle should aid the user in adjusting the angle at the next point. The direction of the arrow indicates the direction the curve's path will take. Figure 1 in Appendix C shows the points that describe a lower case c. Figure 2 in the same appendix shows the letter when the points are connected by drawing with a bold pen.

Some care must be taken in moving the joystick from point to point. TYPDESNGR allows the user to move the joystick anywhere within the grid until the user chooses a point to be saved. In order to do this without cluttering the screen with trails of dots, TYPDESNGR turns off dots as the joystick is moved to new positions. So the user may erase pieces of the arrows, the points themselves, and/or the grid by causing the joystick's path to touch the path of the arrows or points in the process of positioning the joystick to save a new point. The erasing is an inconvenience for the user as it will be difficult to remember previous points and angles. The letter will not be effected as the information about the points is still available to TYPDESNGR. In addition, the user should remember that the defining points of the letter are at the bases of the arrows, not at the points.

This will be especially important where two strokes should join at the same point. The accuracy of the joystick leaves a little bit to be desired where exact matches such as stroke joins are concerned. But the accuracy may be achieved with the commands from the alter menu when the joystick fails to place the point in the desired location.

The start command (S) indicates that the user is ready to position the joystick. After the user has typed the start command and a carriage return, a dot will appear on the screen to indicate where the joystick is presently pointing. The initial dot will not appear if the joystick is pointing outside the grid area. Once the start command has been issued the user may move the joystick to the point on the screen that will be the beginning point of the first stroke. While the joystick is at this point, the user will issue the begin stroke command (B) from the keyboard to indicate that this is the first point of the stroke. TYPDESNGR will then ask the user if he wishes to adjust the angle at this point. If the user answers no (N), TYPDESNGR will copy the angle from the previous point for this point. If the user answers yes (Y), TYPDESNGR will indicate how the user may adjust the angle by typing the message, "TYPE NUMBER OF DEGREES (Ø - 359)". As with the placement of the points, some experimentation will allow the user to choose appropriate angles. The angle is measured from an imaginary horizontal line originating at the point in a counterclockwise direction. While the arrows which

are drawn at the points will be less accurate for small degree differences, the spline functions will reflect these differences. To aid the user, TYPDESNGR remembers angles chosen at the two previous points and assumes that curves change gradually. When a new point is chosen with a new or copied angle, TYPDESNGR compares this angle with the two previous points and warns the user if the curve would make a sudden change as a result of the point and angle just placed (WARNING - ABRUPT CURVE CHANGE). No information is changed as a result, as there are letter strokes where an abrupt curve change is indeed what the user wanted. If the abrupt change was not intentional, the user may change the angle and/or placement of the point with commands from the alter menu.

Now the joystick may be moved to the next point on the letter's spine. All points between the beginning of the stroke and the end of the stroke are saved by using the keep (K) command. Each stroke is limited to fifty points. The angle may also be adjusted after the keep (K) command and after the end stroke command (E). The end stroke command (E) indicates the last point in a letter stroke. After adjusting the angle, a new stroke may be started (B) or the letter may be completed by issuing a stop command (P). The user may draw straight lines with TYPDESNGR by specifying the same angle for two adjacent points and making sure that the points fall on a straight line proceeding at that angle.

The last command on the joy menu is the connect (C) command. The connect (C) command takes the points and angles the user has defined and determines the shape of the letter. This command is meaningless unless the user has either retrieved a previously defined letter with the get letter (GL) command from the draw menu or defined a letter shape in the current session with TYPDESNGR. The connect (C) command causes generation of the spline pieces between pairs of points and the application of the pen shape to this path. TYPDESNGR indicates that the points are being connected by issuing an information message to the user (CONNECTING . . .). The time required to draw the letter form depends on the number of strokes in the letter and the number of points in each stroke. For instance, the uppercase K, Figures 4 and 5 in Appendix C, took 88 seconds to draw. As the letterform appears on the screen, numbers associated with the points will also appear to aid the user with further modifications. The connect (C) command may take one argument. If the user types: C - <CR> , the letter will be drawn on the screen without the point numbers. If the command is issued without the argument, the letter will be drawn on the screen with the point numbers. Figure 1 in Appendix C shows the points that define a lower case c and the tangent arrows associated with each point. Figure 2 in the same appendix shows the c from Figure 1 which was drawn as the result of a connect command with no arguments. Figure 3 shows the c drawn with the connect command and the argument. In Figure 2 the point numbers are written near each point and in

Figure 3 point numbers are not written.

3.1.4 The Alter Menu

When the change menu (G) command is issued from the joy menu, the alter menu is retrieved. There are five new commands on the alter menu. The quit (Q) command returns the user to the joy menu. All the new alter menu commands take arguments and an example of each command is given on the line which follows the command and description. The three lines following the menu title contain explanations of the abbreviations used in the example commands. The general format of the commands is the same as for the grid (GD) command on the specifications menu: the command followed by at least one space, the arguments separated by commas, and a carriage return to end the command. Since alter menu commands are short, all the arguments must be typed on one line and a semicolon is not necessary after the last argument. Commands will not be executed unless they have all the required arguments specified in the correct order. All five of the commands have stroke number and point number as the first two arguments.

The retrieve information (R) command gives the user information about the point number indicated. It has only the two arguments: stroke number and point number. Typing: R 1,2<CR> would retrieve information about the second point in the first stroke.

The command produces a line of information in the message area containing the horizontal (X) and vertical (Y) position of the point and the angle (X = 120, Y = 30, ANG = 340). This information is useful in determining arguments for the other alter menu commands and for checking to make sure that strokes meet as intended. The adjust angle (A) command allows the user to change the angle at a point. In addition to stroke number and point number, the new angle is specified in degrees. Typing: A 3,4,20 <CR> would change the angle at point 4 in stroke 3 to 20 degrees. In addition to making this change in the data structure containing the letter points, the old arrow at the point is erased and a new arrow is drawn reflecting the new angle of 20 degrees. The move point (M) command allows the user to move an existing point to a different location with a different angle. The additional arguments for the move point (M) command are: x location, y location and angle. Typing: M 1,10,100,50,200 <CR> would update the data structure for point 10 in the first stroke and erase the old arrow. A new arrow would then be drawn at location (100,50) with an angle of 200 degrees. The delete point (D) command removes a point from the letter. The stroke number and point number are the only arguments. The delete point (D) command does not permanently update the data structure, but the arrow at the deleted point is erased. The insert point (I) command will place a new point before the point indicated in the command. The arguments for the insert point (I) command are the same as for the move point (M) command. Typing: I 1,4,30,70,100 <CR> would insert a

new point before the fourth point in the first stroke at location (30,70) and an arrow would be drawn at that point with an angle of 100 degrees.

Neither the insert point (I) command nor the delete point (D) command will update the data structure containing the point information. The data structure will be updated by issuing the connect (C) command from the joy menu. These alter menu commands are not intended for extensive letter design but rather to aid the user when he forgets a point or when he places a point improperly. Only 10 points may be inserted without connecting. An inserted point must be inserted before an existing point, since the inserted points are not in the data structure until the points have been connected. In addition, deleting points will not renumber the remaining points. Again, the connect (C) command from the joy menu will accomplish this. Therefore, if a stroke has three points and point 2 is deleted (D 2,2 <CR>), points 1 and 3 remain. Subsequent commands concerning this stroke may only address points 1 and 3, until a connect (C) is done to renumber the points.

3.1.5 Example TYPDESNGR Session

In this example session the user accepts many of the defaults set up by TYPDESNGR which eliminates the need for some of the commands. The objective is to design an italic upper case

'K'. The K designed here is shown in Figure 4 of Appendix C. When the specifications menu appears on the screen, the user issues the grid (GD) command. The user chooses the default grid dimensions by typing: ; <CR> for the grid arguments, and the default grid is drawn on the screen. The defaults allow the user to draw a letter at approximately 250 point size using a pen of medium weight. Marks on the left side of the grid indicate from the bottom: descender - 77 points, baseline - 80 points, x-height - 77 points, capital height - 140 points, and ascender height - 154 points. The marks across the top and bottom of the grid are set for 54 units per em. The user will receive all messages from TYPDESNGR, those containing information as well as the error messages. The second command that the user issues would be the change menu (CM) command. When the draw menu appears, the user issues the draw (DW) command to retrieve the joy menu.

The commands from the joy menu that must be issued are indicated in the list below. The column labelled CMD is the command which the user types. It is echoed on the screen at the left above the message line. After the user issues a begin stroke (B) command, a keep (K) command or an end stroke (E) command, he is asked "ADJUST ANGLE AT THIS POINT? (Y,N)". The user answers "Y" to this question. The next column, DEG, contains the number of degrees specified for the angle at this point. The final column labelled JOYSTICK POSITION indicates where the joystick should be positioned before issuing the next keyboard command.

Joy Menu Commands

CMD	DEG	JOYSTICK POSITION
S		At the first point in the first stroke
B	40	" next "
K	30	" last "
E	0	The joystick is not moved from the previous point.
B	267	At the next point in the second stroke
K	266	" " "
K	265	" " "
K	264	" " "
K	263	" " "
K	250	" " "
K	180	" last "
E	160	At the first point in the third stroke
B	255	" next "
K	250	" " "
K	220	" " "
K	210	" last "
E	200	The joystick is not moved from the previous point.
B	350	At the next point in the fourth stroke
K	318	" " "
K	316	" " "
K	314	" " "
K	312	" " "
K	0	" last "
E	30	
P		
C		
G		

The connect (C) command causes the letter to be drawn on the screen with a number placed above and to the left of each point. After the user issues the connect (C) command, he observes that the joystick moved slightly for the first point on the second stroke and that he would like to change the angle on the last point of the last stroke. Finally, he issues the change menu (G) command to retrieve the alter menu and make the adjustments. Although the following information would not all be necessary to make these adjustments, it is provided so that the reader may duplicate this particular "K". The retrieve information (R)

command provides this information and is listed in the first column. The next columns give the horizontal position (X), vertical position (Y), and the angle at the point (ANGLE).

```
Alter Menu Commands
CMD        X      Y     ANGLE
R 1,1     23    158      40
R 1,2     37    164      30
R 1,3     51    171       0

R 2,1     51    173     267
R 2,2     50    157     266
R 2,3     48    140     265
R 2,4     45    122     264
R 2,5     40    103     263
R 2,6     34     88     250
R 2,7     23     83     180
R 2,8     10     86     160

R 3,1     93    171     255
R 3,2     80    154     250
R 3,3     70    148     220
R 3,4     58    142     210
R 3,5     47    136     200

R 4,1     47    138     350
R 4,2     58    125     318
R 4,3     71    115     316
R 4,4     86    101     314
R 4,5    100     86     312
R 4,6    112     83       0
R 4,7    123     86      30
M 2,1,51,171,267
A 4,7,20
Q
```

The move (M) command moves the first point in the second stroke to the same position that the last point in the first stroke occupies. The adjust angle (A) command changes the way the last stroke ends. When the adjustments are made, the user issues the quit (Q) command in order to return to the joy menu. The connect (C) command will draw the letter again. The user may go back and

forth between the joy and alter menus until he is pleased with the letter or until he wishes to stop. At this point (with the joy menu the current menu), the user issues the quit (Q) command which retrieves the draw menu. The user then issues the save letter (SL) command: SL KT UK. This command will save the letter in the file APPLE2:KT.UK. To exit from TYPDESNGR the user issues the quit (QT) command. TYPDESNGR then clears the screen and the program exits.

3.2 The TYPDESNGR Spline Function

In this section the TYPDESNGR spline function is described. The naming conventions used in the routine Connect are maintained here (see Appendix B: TYPDESNGR Source Listing). The TYPDESNGR equations are very similar to the general cubic spline equation given in Section 1. In this section, however, the specific equations of the vector components, x(t) and y(t), of the point P(t) are used. In order to allow reference to Appendix B and to differentiate between the coefficients for x and the coefficients for y, the coefficients for x have an x preceding them (xaØ - xa3) and the coefficients for y have a y preceding them (yaØ - ya3). The TYPDESNGR spline function is a parameterized cubic spline function with the parameter, t, varying between Ø and 1 for each spline piece. Because the user specifies slope information at each point, the TYPDESNGR spline function has local control. The vector components of each point are:

$$x(t) = xa\emptyset + xa1 \cdot t + xa2 \cdot t^2 + xa3 \cdot t^3$$
$$y(t) = ya\emptyset + ya1 \cdot t + ya2 \cdot t^2 + ya3 \cdot t^3 \qquad \text{(Equation 1)}$$
$$\emptyset \leq t \leq 1$$

A general description of how the spline function is used is given first. The supporting equations follow this description.

The user has provided the locations of the points and the angles at the points. TYPDESNGR starts with the first point in

the first stroke and proceeds in order through the last point in the last stroke. The points the user has specified are the knots of the curve. The points are taken two at a time (points 1 and 2, 2 and 3, ..., n-1 and n). The slope at each point is determined by taking the tangent of the angle at that point. For each pair of points TYPDESNGR creates points, the connecting points, which lie inbetween the two points on the path of the spline curve. A letter stroke which has been defined by n points will have n-1 spline pieces. The connecting points for each spline piece are created as the parameter t varies from Ø to 1. When t=Ø, the beginning point is plotted. The connecting points are plotted when the equations for x(t) and y(t) are solved for the intervening values of t. When t=1, the second point of the pair is plotted. Each time a point is plotted, a whole pattern is actually plotted surrounding the point. The pattern is determined by which pen the user has chosen.

With the information from a pair of points, it is possible to find the coefficients of Equation 1 for the spline piece. X1 denotes the x value at the first point, Y1 the y value; and X2 and Y2 denote the corresponding values at the second point of the piece. The first derivative provides the equation which corresponds to the slope at the point ($x'(t)$ and $y'(t)$). The slopes at the two points (SL1 and SL2) are found by taking the tangent of the angles which were supplied by the user for these points. The slopes (SL1 and SL2) represent a change in y with

respect to x, while y′(t) and x′(t) represent changes in y and x with respect to t. In order to use this information in the parameterized equation y′(Ø) is equated to SL1, x′(Ø) is equated to 1.Ø, y′(t) is equated to SL2 and x′(t) is equated to 1.Ø. This information is sufficient to provide the coefficients for the equations in this spline piece. The following equations are derived by making the substitutions in Equation 1:

$$x(\phi) = x1 = xa\phi$$
$$x'(\phi) = 1.\phi = xa1$$
$$x(t) = x2 = xa\phi + xa1 \cdot t + xa2 \cdot t^2 + xa3 \cdot t^3$$
$$x'(t) = 1.\phi = xa1 + 2 \cdot xa2 \cdot t + 3 \cdot xa3 \cdot t^2$$

$$y(\phi) = y1 = ya\phi$$
$$y'(\phi) = SL1 = ya1$$
$$y(t) = y2 = ya\phi + ya1 \cdot t + ya2 \cdot t^2 + ya3 \cdot t^3$$
$$y'(t) = SL2 = ya1 + 2 \cdot ya2 \cdot t + 3 \cdot ya3 \cdot t^2$$

These equations provide values for xaØ, xa1, yaØ and ya1 and allow TYPDESNGR to solve for xa2, xa3, ya2, and ya3.

$$xa2 = \frac{3(x2-x1)}{t^2} - \frac{2(1)}{t} - \frac{1}{t}$$

$$xa3 = \frac{1}{t^2} + \frac{1}{t^2} + \frac{2(x1-x2)}{t^3}$$

$$ya2 = \frac{3(y2-y1)}{t^2} - \frac{2(SL1)}{t} - \frac{SL2}{t}$$

$$ya3 = \frac{SL1}{t^2} + \frac{SL2}{t^2} + \frac{2(y1-y2)}{t^3}$$

Since a normalized spline was used, substituting t=1 for t in the above equations will provide all the coefficients for Equation 1. Equation 1 then provides the information necessary to create the connecting points for the spline piece. Next, t is incremented by .Ø7 and a new value of x and y is produced. The incrementing of t is repeated until t reaches the end of the interval, 1 in this case.

Some experimentation was done in choosing the increment. The objective was to be able to produce strokes without gaps without making the user wait too long to have the letter drawn. The increment, .Ø7, was also chosen to give an even rendering of the letter at each of the four weights of the pen. Compromises were made so that the light and medium pen produce the curves with a balanced overlap of pen pattern. The line pen produces the letter as a dotted line and the broad pen produces the letter with more overlap than necessary.

Additional experimentation was done with the spline function itself. The normalized spline function of Equation 1 seemed to produce the best overall curves. Many authors mention the use of chord length (the linear distance between the two points) as a good choice for parameter range. Experiments were done comparing curves using normalized parameterization with chord length parameterization. Four curves each between two points were drawn with each spline function: a curve starting in the first quadrant and ending in the fourth quadrant (curve 1), a curve starting in the

fourth quadrant and ending in the third quadrant (curve 2), a curve starting in the third quadrant and ending in the second quadrant (curve 3), and a curve starting in the second quadrant and ending in the first quadrant (curve 4). The following list contains the sets of points for the four curves.

Curve	X	Y	Angle
1	27	110	40
	70	109	340
2	56	144	290
	38	119	260
3	60	114	200
	38	121	150
4	16	130	100
	34	145	50

The curves produced with chord length parameterization contained large deviations from the smooth path between the points for curves 2 and 4. The curves overshot the ending point and then returned to the ending point. The value of the slopes for angles near 90 degrees and 270 degrees changes very rapidly. Chord length parameterization seems to emphasize these sharp changes. With some additional restraints to control the curves in these two areas (near 90 degrees and near 270 degrees), chord length parameterization is appealing as curves 1 and 3 were more attractive than the corresponding normalized curves.

Another experiment was done with the normalized curves. The magnitude of the slope was changed to see how the curves would be effected. The value of $x'(t)$ and $y'(t)$ were both multiplied by

5. The choice of 5 was fairly arbitrary. Multiplying by 2 produced insignificant differences while multiplying by 5 really emphasized the differences with the normalized curves. Multiplying both derivatives by the same number allows the slope to stay the same while its magnitude is increased. The same four curves were drawn as for the chord length parameterization. Changing the magnitude of the slope causes the curves to overshoot the ending point and not return to it. The overshooting occured on curves 2 and 4 but also on curve 1. In general, the curves produced with this change in magnitude were also more jagged than the normalized curves.

3.3 TYPDESNGR Output

The primary output that TYPDESNGR produces is the image of the letter on the APPLE screen. Figures 4 and 5 in Appendix C contain the letter K which was designed in section 3.1.5. These figures were produced using a Tektronix 4632 Video Hard Copy Unit. Figure 4 was drawn with a medium pen, and Figure 5 with a bold pen. Figures 1 through 3 contain other examples of the TYPDESNGR screen output. The image is plotted directly from the calculations of the spline pieces. The user reviews it, makes modifications, redraws it etc. until the letter is in a form that is acceptable. The save letter (SL) command creates another form of output. The save letter (SL) command creates an output file on a floppy disk. The file contains only the points that define

the letter and associated angles that a tangent line makes at each point. The letter is stored concisely so that when it is retrieved is can be easily redrawn. The third form of output is another floppy disk file created by the save specifications (SS) command. This file contains the grid specifications.

Section 4: Conclusions

> A machine-made article requires, if it is to be fit for use, just as much creative intelligence and human discretion in its making as a handmade article requires. [WARDE 53]

While this thesis has barely scratched the surface of the problems in providing a computer-aided type design tool, the appeal of the spline function for creating the desired shapes is evident. Once the long design process of a letter, or in the future a typeface, is completed, the type designer will have the capabilities to produce many variations on the original design. Additional attention to specific shape requirements would lead to a more detailed choice of coefficients. The additional effort with the coefficients would provide more control over the letter shapes. This could be accomplished internally with little extra information required from the user. The storage of letterforms in concise mathematical form allows the letter to be easily manipulated. This ease of manipulation would seem appealing to the type designer who may be required to produce a variety of family members based on one central design. While a great deal of program code was created in order to give the user a tool that he could use fairly easily, an experienced type designer will find places where he needs more information or wants another facility. With the TYPDESNGR spline function as a basis, such a modification would not disturb the basic drawing tool.

Of the list of possible major additions to TYPDESNGR, the following ideas would take priority: creating letters with smooth edges, providing an input device that can be used more like a pencil and paper (a bitpad for instance), expanding the command structure to deal with whole fonts, and adding high quality typesetting output devices for final proofs. Creating letters with smooth edges would be a logical addition to TYPDESNGR as a type designer could hardly accept a letter with ragged edges. A device with higher resolution would allow TYPDESNGR to create smooth edged letterforms. Additional pen shapes and sizes could be added at this point to allow the designer the kind of flexibility that he would get from different shaped drawing tools. With the addition of a higher resolution device and increased memory capacity, the completed letter image could be stored in a data structute for subsequent plotting. The user would then see the entire letter at once instead of having it gradually drawn before his eyes. The letter, then, would be available to the save letter command in two forms - the points and angles could be stored for further design work and the completed letter in raster form could be stored for use on typesetting equipment.

A higher resolution device would allow the designer to design in a wider range of sizes. Specifically, the designer could draw letters at smaller sizes. Finally, a higher resolution output device would allow TYPDESNGR to provide the designer with more accurate measurements. If the designer is drawing a

250. point letter, it is essential to guarantee that the size would be accurate to .0002 of an inch as Ikarus does when it cuts the final characters in rubylith with the Aristo drafting machine [HAWKI 80]. The advantages of giving TYPDESNGR the capability to have higher resolution output on both video and hard copy devices are many fold.

Replacing the joystick with a device like a bitpad would allow a type designer to make a gradual transition between traditional tools and a computer-aided design tool. The letter could actually be drawn on paper and placed on the bitpad. Then, the designer could use the pointing device to designate points and angles using the drawing as a guide. In general, drawing the letters with a bitpad would be a more familiar movement than the movement of a joystick.

TYPDESNGR does not give the designer the capability to issue global commands, commands that could effect an entire font. Such commands could provide the designer with the capability of standardizing serifs within a font, removing serifs from a font, creating an italicized version of a font, or producing a font at different weights. Finally, as TYPDESNGR is more tuned toward a production environment, it would be necessary to provide the designer with the possibility of producing output on high resolution typesetting equipment so that final adjustments could be made.

Experiences with Ikarus have shown that existing typefaces can be copied using a computer-aided design tool. New typefaces have also been designed with Ikarus, but in most cases the designer used an intermediary to transfer the drawing to Ikarus. In these cases Ikarus was being used as primarily a digitizing tool. The few experiences with METAFONT suggest that the designer is also remotely involved. In order to create letterforms that are of high quality and benefit from technological innovations, the designer must use the computer-aided design tool himself. The engineers and computer scientists must be more aware of what the type designer needs. Creating a flexible, efficient, mathematically sound computer-aided type design tool is as complicated as designing a typeface that achieves its design goals and becomes another respected typeface such as, Janson, Baskerville, or Optima in the process.

Bibliography

AHLBE 67 Ahlberg, J.H.; Nilson, E.N.; Walsh, J.L. The Theory of Splines and Their Applications, Academic Press, N.Y., 1967.

AHUJA 68 Ahuja, D.V. "Interactive Graphics in Data Processing: An Algorithm for Generating Spline-like Curves". IBM Systems Journal. Vol. 7, 1968, pp.206-217.

APPLE 79 Shillington, K.A., Ackland, G.M., Raskin, J. and Howard, B., editors. APPLE PASCAL Reference Manual. APPLE Computer Company, Cupertino, California, 1979.

ARMIT 71 Armit, A.P. and Forrest, A.R. "Interactive Surface Design", Parslow, R.D. and Green, R. Elliott, ed. Advanced Computer Graphics Economics Techniques and Applications. Plenum Press, N.Y., 1971, pp. 1179-1202.

BERLO 80 Berlow,David. Personal Communication, Mergenthaler Linotype Company, New York, New York.

BEZIE 74 Bezier, P. "Mathematical and Practical Possibilities of UNISURF", Barnhill, Robert E. and Riesenfeld, Richard F., ed., Computer Aided Geometric Design, Academic Press, N.Y., 1974, pp. 127-152.

BIGEL 80 Bigelow, Charles. "On Type: Galliard." Fine Print, Volume V, Number 1, Jan 1979; pp. 27-30.

BRADY 80 Brady, Lawrence. Personal Communication. Los Alamitos, California.

CATMU 74 Catmull, Edwin and Rom, Raphael, "A Class of Local Interpolating Splines", Barnhill, R.E. and Riesenfeld, R.F., ed. Computer Aided Geometric Design, Academic Press, N.Y., 1974, pp. 317-326.

CHASE 78 Chasen, Sylvan H. Geometric Principles and Procedures for Computer Graphic Applications, Prentice Hall Inc.,Englewood Cliffs, N.J., 1978.

CLINE 74 Cline, A.K. "Scalar- and Planar-Valued Curve Fitting Using Splines Under Tension", Communications of the

ACM. Vol 17, Number 4, April 1974, pp. 218-220.

COX 78 Cox, C.H.,III, Blesser, B.A. and Eden,M. "The Graphical Context of Printed Characters", Visible Language, Vol XII, Number 4, Autumn 1978, pp. 429-447.

GORDO 74 Gordon, William J. and Riesenfeld, Richard F., "B-spline Curves and Surfaces", Barnhill, R.E. and Riesenfeld, R.F., ed., Computer Aided Geometric Design, Academic Press, N.Y., 1974, pp. 95-125.

GOUDY 33 Goudy, Frederic W. "On Designing A Typeface" The Dolphin: A Journal of the Making of Books Number 1. Limited Editions Club, N.Y., 1933.

HALEY 80 Haley, Allan, type designer. Personal communication. Compugraphic, Wilmington, Massachusetts.

HAWKI 80 Thomas Hawkins, manager of IKARUS system. Personal communication. Compugraphic, Wilmington, Massachusetts.

HAZON 79 Hazony, Y. "Algorithms for Parallel Processing: Curve and Surface Definition with Q-Splines", Computers and Graphics, Volume 4, 1979, pp. 165-176.

IKARU A brief description of how IKARUS works.
Unternehmensberatung Rubow Weber
Peter Karow
2 Hamburg G2
Germany

KNUT1 79 Knuth, Donald E. 'Mathematical Typography' in Tex and METAFONT New Directions in Typsetting, Digital Press, Burlington, MA, 1979.

KNUT2 79 Knuth, Donald E. 'Tex a System for Technical Text' in Tex and METAFONT New Directions in Typsetting, Digital Press, Burlington, MA, 1979.

KNUT3 79 Knuth, Donald E. 'METAFONT, a System for Alphabet Design' in Tex and METAFONT New Directions in Typsetting, Digital Press, Burlington, MA, 1979.

LATHA 77 Latham, John. "Tailoring Type to Technology" Penrose Annual International Review of Graphic Arts. 1977/1978,pp.189-198.

MOSS 79 Moss, R. and Lindgard, A. "Parametric Spline Curves in Integer Arithmetic Designed for Use in

Microcomputer Controlled Plotters", Computers and Graphics. Vol 4, 1979, pp.51-61.

NEWMA 79 Newman, William M. and Sproull, Robert F. Principles of Interactive Computer Graphics, Second Edition, McGraw Hill Book Co., N.Y., 1979.

PRING 79 Pringle, Alison; Robinson, Peter and Wiseman, Neil. "Aspects of Quality in the Design and Production of Text" Computer Graphics, Vol. 13, No.12, August 1979, pp. 63-70.

PROVA 80 A. Provan. From RIT course Typography 100.

RAMSH 80 Ramshaw, Lyle. Personal communication. Xerox Corporation. Palo Alto, California.

ROGER 76 Rogers, David F, and Adams, J. Alan. Mathematical Elements for Computer Graphics, McGraw Hill Book Co., N.Y., 1976.

SCHOE 69 Schoenberg, I.J., ed. Approximations with Special Emphasis on Spline Functions, Academic Press, N.Y., 1969.

SILVE 73 Silver, Gerald A. Modern Graphic Arts Paste-up. American Technical Society, Chicago, 1973.

STONE 80 Stone, Sumner. Personal Communication. Autologic Incorporated. Newbury Park, California.

TUGBO 80 Welland, Robert, editor. The Tex Users Group Newsletter, Volume 1, Number 1, Providence, Rhode Island, October 1980.

TIMME 80 Timmer, Henry G. "Alternative Representation for Parametric Cubic Curves and Surfaces," Computer-Aided Design. Vol 12, Number 1, January 1980, pp.25-28.

UNGER 79 Unger, Gerard. "The Design of a Typeface". Visible Language. Vol. XIII,2,1979, pp.134-149.

WARDE 53 Warde, Beatric. Concerning Some Words and Types by Varied Hands. Pickering Press, Mapleshade, N.J., 1953.

WARDE 56 Warde, Beatrice. The Crystal Goblet Sixteen Essays on Typography. Henry Jacob, editor, World Publishing Co., N.Y., 1956.

ZAPF 65 Zapf, Hermann and Stauffacher, Jack Werner. Hunt Roman: The Birth of a Type. The Pittsburgh Bibliophiles, Pittsburgh, 1965.

ZAPF 68 Zapf, Hermann. "Changes in Letterforms Due to Technical Developments. Visible Language. Vol. II, October 1968, pp. 351-368.

Appendix A: TYPDESNGR Menus

During the execution of TYPDESNGR, the user is presented with four different menus which contain the commands the user may issue. The menus are presented here exactly as the user sees them on the APPLE screen.

1. The first menu is the Specifications Menu. This menu is presented to the user when TYPDESNGR begins execution. The user may also retrieve this menu by issuing the change menu (CM) command from the draw menu.

```
MENU #1 - SPECS
CMD - EXPLANATION

GD - GRID SPECS
PN - PEN SPECS
UM - USER MODE
SS - SAVE SPECS
GS - GET SPECS
CM - CHG MEN-DRAW
QT - QUIT
```

2. The second menu is the Draw Menu. The user may retrieve this menu by issuing the change menu (CM) command from the specifications menu or by issuing the quit (Q) command from the joy menu.

```
MENU #2 - DRAW
CMD - EXPLANATION

DW - DRAW LETTER
SL - SAVE LETTER
GL - GET LETTER
CM - CHG MEN-SPEC
QT - QUIT
```

3. The third menu is the Joy Menu. It is numbered 2A because it is a low level menu which is retrieved by issuing

the draw (DW) command from the draw menu. When a low level menu is the current menu, the user must issue more than one command to exit from the program. The user specifies input from the keyboard as well as from the joystick when this menu is the current menu. The joy menu is also retrieved when the quit (Q) command is issued from the alter menu.

```
MENU #2A - JOY
 SPECIFY PTS IN
 DRAWING ORDER
1. S START LETTER
2.   MOVE JOYSTIK
3. B BEGIN STROKE
4.   ADJUST ANGLE
5.   MOVE JOYSTIK
6. K KEEP POINT
7.   REPEAT 5,6,4
8. E END STROKE
9.   REPEAT 4,3-8
10.P STOP LETTER
11.C CONNECT PTS
12.G CHG MEN-ALTR
13.Q QUIT
```

4. The fourth menu is the Alter Menu. The alter menu is also a low level menu and is retrieved in reponse to the change menu (G) command from the joy menu. The alter menu contains commands which alter the point information defined with the joy menu commands.

```
MENU #2B - ALTER
  S = STROKE NUM
  P = POINT NUM
  A = ANGLE
A - ADJUST ANGLE
A S,P,A
I - INSERT POINT
I S,P,X,Y,A
D - DELETE POINT
D S,P
M - MOVE POINT
M S,P,X,Y,A
R - GET POINTINFO
R S,P
Q - QUIT
```

Appendix B: TYPDESNGR Source Listing

```
(*$s++*)
program typdesngr;
(*$c copyright 1981 kathleen barry albertini*)
(*9 dec 80*)
(*19 mar 81*)


uses transcend,turtlegraphics, applestuff;
const xmax = 279;       ymax = 191;
      lmen = 161;       bmen = 16;
      rmen = xmax;      tmen = 183;
      lmsg = 0;         bmsg = 0;
      rmsg = xmax;      tmsg = 8;
      ldeb =lmen;       bdeb = bmen;
      rdeb = rmen;      tdeb = 23;
      chght = 7;        spce = 11;
      xdim = 160;       ydim = 175;

      (* the above constants define the
         screen
         men - menu area upper left.
         msg - message area bottom 2 lines.
         and the upper left area is the
         drawing area of the screen.
      *)

      ppi = 72;  (*points per inch*)
      dpi = 46; (* dots per inch *)
      ptmin = 72;       ptmax = 250; (* pointsize range*)
      pl = 50;  (* point limit per stroke*)
      il = 10;  (* # of points that can be inserted*)

      erl = 30; (*length of error messages*)
      ner = 41; (*num of error msgs *)
      nrw = 39; (*num of resrved words *)
      rwl = 2;  (*length of resrvd wrds*)
      ic = 40;  (*width of typein *)
      bs = 8;   (*backspace*)
      nmax = 6; (*numb of digits in numbr*)

      vert = 90;        dvert = 270;
      horiz = 0;        lhoriz = 180;
      backup= -135;

type msglvl = (info,warn,deb); (*types of msgs *)
     mentype = (specmenu,drawmenu,joymenu,altermenu);
```

```
      symbol = (nul,comma,cr,specsym,drwmsym,
       number,equals,err,semicolon,minus,plus,
       quitsym,changesym,gridsym,pensym,trysym,
       umodesym,sssym,gssym,pointsym,cptsym,
       drawsym,slsym,glsym,exprtsym,novicsym,
       unsym,pzsym,bhsym,xhsym,dhsym,ahsym,
       cphsym,yesym,nosym,ltsym,mdsym,bdsym,
       linesym,abort,
       adjang,begstk,conn,delete,endstk,chgmen,
       inspt,keep,movpt,stop,qt,infop,south,start);
      umode = (albertini,zapf);
      pmode = (light,med,bold,lin);
      symtag = packed array [1..rwl] of char;

      (* storage for defining points of
         strokes of letter.
            pts[1,n] - x coordinate
            pts[2,n] - y coordinate
            pts[3,n] - angle at the point
      *)
      stkinfo = record
       pn:integer; (* number of pts in stroke*)
       pts: array [1..3,1..pl] of integer;
       slk: ^stkinfo
      end;

      letter = record
       nme:string[3];
       sn:integer;
       stks:^stkinfo
      end;
      stfil = record
       fpn:integer;
       fpts: array[1..3,1..pl] of integer;
      end;

      ptnum = packed array[0..3,1..6]of boolean;
      ptr = ^stkinfo;

var    ch: char;
       xr,ybot:integer;
       un,pz,bh,xh,dh,ah,cph: integer; (*arguments for grid*)
       base: integer;    (*baseline in pixels*)
       gridspecs: set of symbol;
       specfile: text; (*file with specs saved*)
       lettfile: file of stfil;
       xyz:stfil;
       errs: array[1..ner] of string[30];
       debmsg: array[1..5] of string[6];
       ccn,lln:integer; (*char count and line pointer*)
       line: array [1..ic] of char;
```

```
      ii,num:integer;
      wsym: array [1..nrw] of symbol;
      tags: array [1..nrw] of symtag;
      tg: symtag;
      strtg:string[2];
      schr: array[char] of symbol;
      u:umode;
      sym:symbol;

      abc:letter;
      s,t:^stkinfo;
      heap:^integer;
      heaprel,heapinit:boolean;
      ict,dct:integer;
      insert:array[1..5,1..il] of integer;
       (* x,y,angle,strk#,pt#*)
      combine:array[1..3,1..pl] of integer;
      pen:pmode;
      pns:array [0..9] of ptnum; (*point
       numbers to label points on screen*)



procedure msg(lvl:msglvl;num:integer);forward;

(*initial - initalization routines
            these routines are only called at start up time,
            so this procedure is not in core after it has
            finished.
*)
segment procedure initial;
 (*initags - initialize variables used by getsym
             for recognizing typed input
 *)
 procedure initags;
 begin
  for ch := ' ' to '''' do schr[ch] := nul;
  schr[',']  := comma;
  schr['!']  := cr;
  schr['=']  := equals;
  schr[';']  := semicolon;
  schr['-']  := minus;
  schr['+']  := plus;

  tags[1]  := 'a ';
  tags[2]  := 'ah';
  tags[3]  := 'b ';
  tags[4]  := 'bd';
```

```
 tags[5] := 'bh';
 tags[6] := 'c '; tags[7] := 'ch';
 tags[8] := 'cm';
 tags[9] := 'd ';
 tags[10]:= 'dh';
 tags[11]:= 'dw';
 tags[12]:= 'e ';
 tags[13]:= 'ex';
 tags[14]:= 'g ';
 tags[15]:= 'gd';
 tags[16]:= 'gl'; tags[17]:= 'gs';
 tags[18]:= 'i '; tags[19]:= 'k ';
 tags[20]:= 'ln'; tags[21]:= 'lt';
 tags[22]:= 'm '; tags[23]:= 'md';
 tags[24]:= 'n ';
 tags[25]:= 'no';
 tags[26]:= 'p ';
 tags[27]:= 'pn'; tags[28]:= 'pz';
 tags[29]:= 'q ';
 tags[30]:= 'qt';
 tags[31]:= 'r '; tags[32]:= 's ';
 tags[33]:= 'sl';
 tags[34]:= 'ss';
 tags[35]:= 'u ';
 tags[36]:= 'um';
 tags[37]:= 'un';
 tags[38]:= 'xh';
 tags[39]:= 'y ';
end;

(*initsym - initialize the symbols used by getsym
*)
procedure initsym;
begin
 wsym[1] := adjang;
 wsym[2] := ahsym;
 wsym[3] := begstk;
 wsym[4] := bdsym;
 wsym[5] := bhsym;
 wsym[6] := conn;      wsym[7] := cphsym;
 wsym[8] := changesym;
 wsym[9] := delete;
 wsym[10]:= dhsym;
 wsym[11]:= drawsym;
 wsym[12]:= endstk;
 wsym[13]:= exprtsym;
 wsym[14]:= chgmen;
 wsym[15]:= gridsym;
 wsym[16]:= glsym;     wsym[17]:= gssym;
 wsym[18]:= inspt;     wsym[19]:= keep;
 wsym[20]:= linesym;   wsym[21]:= ltsym;
```

```
 wsym[22]:= movpt;     wsym[23]:= mdsym;
 wsym[24]:= nosym;
 wsym[25]:= novicsym;
 wsym[26]:= stop;
 wsym[27]:= pensym;
 wsym[28]:= pzsym;
 wsym[29]:= qt;
 wsym[30]:= quitsym;
 wsym[31]:= infop;
 wsym[32]:= start;
 wsym[33]:= slsym;
 wsym[34]:= sssym;
 wsym[35]:= south;
 wsym[36]:= umodesym;
 wsym[37]:= unsym;
 wsym[38]:= xhsym;
 wsym[39]:= yesym;
end;

(*initpn - initialize an array of point numbers
           which are drawn on the screen when the
           letter strokes are connected
*)
procedure initpn;
var i1,i2,i3,k:integer;

  procedure stuff(s:string);
  var bit:boolean;
      j:integer;
  begin
   for i2 := 1 to 3 do begin
    bit := (s[i2] <> ' ');
    pns[i1,i2,i3] := bit;
   end;
   i3 := i3 - 1;
  end;

begin(*initpn*)
 i1:=0; i3:= 6;
 stuff('xxx');
 for k := 1 to 4 do stuff('x x');
 stuff('xxx');

 i1:=i1+1; i3:=6;
 stuff(' x ');
 stuff('xx ');
 for k:= 1 to 3 do stuff(' x ');
 stuff('xxx');

 i1:=i1+1; i3:=6;
 stuff(' x ');
```

```
stuff('x x');
stuff('  x');
stuff(' x ');
stuff('x  ');
stuff('xxx');

i1:=i1+1; i3:=6;
stuff('xxx');
stuff('  x');
stuff(' xx');
stuff('  x');
stuff('  x');
stuff('xxx');

i1:=i1+1; i3:=6;
stuff('  x');
stuff(' xx');
stuff('x x');
stuff('xxx');
stuff('  x');
stuff('  x');

i1:=i1+1; i3:=6;
stuff('xxx');
stuff('x  ');
stuff('xx ');
stuff('  x');
stuff('  x');
stuff('xxx');

i1:=i1+1; i3:=6;
stuff(' xx');
stuff('x  ');
stuff('xx ');
stuff('x x');
stuff('x x');
stuff(' x ');

i1:=i1+1; i3:=6;
stuff('xxx');
stuff('x x');
for k :=1 to 4 do stuff('  x');

i1:=i1+1; i3:=6;
stuff('xxx');
stuff('x x');
stuff('xxx');
stuff('x x');
stuff('x x');
stuff('xxx');
```

```
 i1:=i1+1; i3:=6;
 stuff(' x ');
 stuff('x x');
 stuff('x x');
 stuff(' xx');
 stuff('  x');
 stuff('xx ');

end;

(*initmsg1 - initialize error and information messages
*)
procedure initmsg1;
begin
 errs[1] := '?? ptsize: 72 - 250 /250 used ';
 errs[2] := '?? units/em: 54,36,18 /54 used';
 errs[3] := '?? dim conflicts with ptsize  ';
 errs[4] := '?? invalid command            ';
 errs[5] := 'typdesngr- a letter design pgm';
 errs[6] := 'choose a cmd from above menu  ';
 errs[7] := '?? max 40 char per cmd line   ';
 errs[8] := '?? unexpected end of input    ';
 errs[9] := '?? num too big -limit 999999  ';
 errs[10]:= 'another command ??            ';
 errs[11]:= '?? unrecognizable option      ';
 errs[12]:= '?? missing  = please try again';
 errs[13]:= '?? number expected            ';
 errs[14]:= '?? use a comma between options';
 errs[15]:= 'defaults used for other option';
 errs[16]:= 'draw the grid?                ';
 errs[17]:= 'un-units/em    pz-pointsize   ';
 errs[18]:= '(in points) xh-xheight,bh-base';
 errs[19]:= '(in points) ah-ascendr,dh-desc';
 errs[20]:= '(in points) ch-cap height     ';
 errs[21]:= '?? error opening file         ';
 errs[22]:= '?? error writing file         ';
 errs[23]:= '?? error reading file         ';
 errs[24]:= '                              ';
end;

(*initmsg2 - initialize error and information messages
*)
procedure initmsg2;
 begin
  errs[25]:= 'adjust angle at this pt? (y,n)';
  errs[26]:= 'type number of degrees (0-359)';
  errs[27]:= '?? point out of range         ';
  errs[28]:= '?? fatal stroke error         ';
  errs[29]:= 'connecting . . . . .          ';
  errs[30]:= '?? invalid alter parameter    ';
  errs[31]:= '?? over insert lim -conn first';
```

```
    errs[32] := 'merging . . . .                    ';
    errs[33] := '?? too many points deleted         ';
    errs[34] := '?? end of stroke befor end del     ';
    errs[35] := '??                                 ';
    errs[36] := 'initializing . . .                 ';
    errs[37] := '?? uncharted path                  ';
    errs[38] := 'warning - abrupt curve change      ';
    errs[39] := 'need e(endstrk) before p(stop)     ';
    errs[40] := '?? no points to connect            ';
    errs[41] := '??                                 ';
  end;
begin(*initial*)
 initags;
 initsym;
 initpn;
 initmsg1; initmsg2;
 lln := 0; ccn := 0;
 ch := ' '; num := 0; ii := 0;
 tg:= '  '; strtg:= '  ';
 sym := nul;
 u := albertini;
 pen := med; (*default*)
 heaprel := false; heapinit := false;
 dct := 0; ict:=0;
 gridspecs := [unsym,pzsym,bhsym,xhsym,dhsym,ahsym,cphsym];
 xr:=xdim; ybot:=bmen;
 initturtle;
 msg(info,5);
end;

procedure pause;forward;
procedure delay(n:integer);forward;
procedure erase;forward;
procedure erasgd;forward;
procedure erasmn;forward;
procedure erasmg;forward;
procedure cleanup;forward;
procedure gline(x,y,dash,dir,bound:integer);forward;
procedure typein;forward;
procedure getsym;forward;
procedure displaym(w:mentype);forward;
procedure find(n:integer;a:ptr;var s:ptr);forward;
procedure merge;forward;



 (*smenu - process commands from specifications
           menu.
           gd - grid specifications, pn - pen specifications,
```

```
          um - user mode, ss - save specifications,
          gs - get specifications, cm - change menu (draw),
          qt - quit
*)

segment procedure smenu;
var which:symbol;
  i:integer;
  c:char;
  fl:boolean;

 (* grid   - draw grid for specified dim
      un - units per em (54,36,18)
      pz - ptsize for drawings
      bh - baseline location in points
      xh - x height in points
      dh - descender length in points
      ah - ascender length in points
      cph - cap height in points
 *)
 procedure grid;
 const tic = 3; dash = 10;
       xadj = 0; yadj = 0;
 var dpz,dbh,dxh,ddh,dah,dch: integer;
   intervl,i:integer;

  function pttdt(pt:integer):integer;
  var r:real;
  begin
   r := (pt/ppi)*dpi;
   pttdt := trunc(r);
  end;

 begin
  if not (pz in [ptmin..ptmax]) then
   begin msg(info,1);  pz := 250;  end;
  dpz := pttdt(pz);
  xr := dpz + xadj;  ybot := ymax-dpz-yadj;
  if xr >= lmen then xr := lmen - 1;
  if ybot <= (tmsg+chght) then ybot := tmsg+chght+1;
  if not (un in [18,36,54]) then
   begin msg(info,2); un := 54; end;
  intervl := dpz div un;

  i:=0;
  if bh>pz then i:=1;
  if bh+xh > pz then i := 1;
  if bh+ah > pz then i := 1;
  if bh+cph > pz then i := 1;
  if bh-dh > pz then i := 1;
```

```
   if i = 1 then msg(warn,3)
   else
    begin dbh := pttdt(bh);
     dxh := pttdt(xh);  dah := pttdt(ah);
     ddh := pttdt(dh);  dch := pttdt(cph);
     erasgd; pencolor(none);
     moveto(0,ymax);
     pencolor(white);    (*frame*)
     moveto(xr,ymax);
     moveto(xr,ybot);  moveto(0,ybot);
     moveto(0,ymax);
     pencolor(none);
     base := ybot+dbh;  (*baseline*)
     moveto(0,base);    pencolor(white);
     moveto(xr,base);
     gline(0,base-ddh,dash,horiz,xr); (*desc line*)
     gline(0,base+dxh,dash,horiz,xr);  (*x height*)
     gline(0,base+dah,dash,horiz,xr); (*asc line*)
     gline(0,base+dch,dash,horiz,xr); (*cap line*)
     i := 0;
     repeat
      begin
       i := i + intervl;
       gline(i,ybot,tic,vert,ymax-ybot);
      end;
     until i + intervl > xr;
    end;
 end;
begin (*smenu*)
(* the menu area of the screen is the
  upper right corner. coordinates of
  the corners are - lmen,ymax/xmax,ymax/
  xmax,bmen/lmen,bmen. there are 24lines
  of characters available - 17 characters
  per line.
*)
 displaym(specmenu);
 typein;
 getsym;
 while not (sym in[quitsym,changesym]) do begin
  if sym = umodesym then begin
   getsym;
   if sym in [exprtsym,novicsym] then
    begin if sym = exprtsym then u := zapf
     else u := albertini;
   end else
    begin msg(warn,11); delay(200);
    end;

  end else
  if sym = gridsym then begin
```

```
un := 0; pz :=0; bh := 0; ah :=0;
xh := 0; dh :=0; cph := 0; fl := false;
if u = albertini then begin
 msg(info,17); delay(200);
 msg(info,18); delay(200);
 msg(info,19); delay(200);
 msg(info,20); delay(200);
end;
typein;  getsym;
while sym in gridspecs do begin
 if sym = cr then begin typein; getsym;
 end else
 begin which := sym; getsym;
  if sym <> equals then msg(warn,12)
  else
  begin getsym;
   if sym <> number then msg(warn,13)
   else
   case which of
    unsym: un := num;
    pzsym: pz := num;
    bhsym: bh := num;
    xhsym: xh := num;
    dhsym: dh := num;
    ahsym: ah := num;
    cphsym: cph := num;
   end;
  end;
  if sym <> err then begin
   getsym;
   if sym = comma then getsym
   else if sym <> semicolon then msg(warn,14);
  end;
 end;
end;
if sym = semicolon then begin
 if un = 0 then
  begin un := 54; fl :=true; end;
 if pz = 0 then
  begin pz := 250; fl :=true; end;
 if bh = 0 then
  begin bh := 80; fl :=true; end;
 if xh = 0 then
  begin xh := 77; fl :=true; end;
 if dh = 0 then
  begin dh := 77; fl :=true; end;
 if ah = 0 then
  begin ah := 154; fl :=true; end;
 if cph = 0 then
  begin cph := 140; fl :=true; end;
 if fl = true then msg(info,15);
```

```
    delay(200);
    msg(info,16);
    typein;
    getsym;
    if sym = yesym then grid;
   end;
  end else
  if sym = pensym then begin
   getsym;
   case sym of
    ltsym: pen:=light;
    mdsym: pen:=med;
    bdsym: pen:=bold;
    linesym: pen:=lin;
   end;
  end else
   if sym = sssym then
    begin
     rewrite(specfile,'apple2:spec.data');
     i := ioresult;
     if i = 0 then
      begin  writeln(specfile,'un=',un,'pz=',pz,'xh=',xh);
       i := i+ioresult;
       writeln(specfile,'bh=',bh,'dh=',dh,'ah=',ah,'ch=',cph);
       i := i+ioresult;
       if i > 0 then msg(warn,22);
       close(specfile,lock);
      end else msg(warn,21);
   end else
   if sym = gssym then
    begin
     reset(specfile,'apple2:spec.data');
     i := ioresult;
     if i = 0 then
      begin readln(specfile,c,c,c,un,c,c,c,pz,c,c,c,xh);
       i := i+ioresult;
       readln(specfile,c,c,c,bh,c,c,c,dh,c,c,c,ah,c,c,c,cph);
       i := i+ioresult;
       if i > 0 then msg(warn,23);
       close(specfile);
      end else msg(warn,21);
   end else msg(warn,4);
   if u = zapf then msg(warn,24)
   else msg(info,10);
   typein;
   getsym;
  end;
  if sym = changesym then sym := drwmsym;
end;
```

```
(*dmenu - process commands from menus
          2 (draw), 2a (joy), and 2b
          (alter).
          draw menu (2) commands -
          dw - draw, retrieve the joy menu.
          sl - save the letter
          gl - get the letter
          cm - change menu (specifications)
          qt - quit (exit the program)
*)
segment procedure dmenu;
const fflop = 'apple2:';
var ang,ptcnt,strkcnt:integer;
    prevang,prevx,prevy,xpad,ypad:integer;
    prevb,strkerr:boolean;
    i:integer;
    fnme,lnme:string[2];
    str12:string[12];

 (*draw - process the draw command,
          control the flow between
          the joy menu and the alter menu,
          and process the joy menu commands.
          s - start letter
          b - begin stroke
          k - keep point - keeppt
          e - end stroke
          p - stop letter
          c - connect points - conn
          g - change menu - alter menu
          q - quit - return to draw menu
 *)
 procedure draw;
  (*readjoy - read coordinates from joystik
  *)
  procedure readjoy;
  var i:integer;
  begin
   xpad := paddle(0);
   for i := 0 to 3 do;
   ypad := paddle(1);
  end;

  (*valid - make sure point is within grid
            xpad,ypad designates point
  *)
  function valid(xpad,ypad:integer):boolean;
  begin
    if (ypad > ymax) or (ypad < ybot) or
       (xpad > xr) or (xpad < 0) then valid := false
    else valid := true;
```

```
end;

(*chkang - check to make sure the angle specified
           is between 0 and 359 degrees.
*)
procedure chkang;
begin
 if num < 0 then num := 360 + num
 else if num >= 360 then num := num mod 360;
end;

(*scanns - check and collect angle
           adjustment parameters
           scanns collects the two previous angles
           so that a sudden change in the curve
           direction can be detected and the user
           is then warned about this.
           if the user has specified a nonnummerical angle,
           the function returns a false condition, otherwise
           it returns true.
*)
function scanns:boolean;
var change:boolean;
begin
 scanns := true;
 change := false;
 getsym;
 if sym <> number then  scanns := false
 else begin
  chkang;
  if (num>=0) and (num<=90) then begin (*1st quad*)
   if (num>ang) or (ang>=270) then (*counter*)
    if (prevang>ang) and (prevang<num)
     then change := true
   else if (num<ang) and ((ang>=0) and (ang <=180)) then (*cloc
    if (prevang<ang) or (prevang>=270)
     then change := true
  end else
  if (num>90) and (num<=270) then begin (*2nd and 3rd quad*)
   if (num>ang) then (*counter*)
    if (prevang>ang) then change := true
   else if (num<ang) then (*clockwise*)
    if (prevang<ang) and (prevang>90)
     then change := true
  end else
  if (num>270) and (num<=359) then begin (*4th quad*)
   if (num>ang) and (ang>=180) then (*counter*)
    if (prevang>ang) or (prevang<=90)
     then change := true
   else if (num<ang) or (ang<=90) then (*clock*)
    if (prevang<ang) then change := true
```

```
  end;
  if change = true then msg(warn,38);
  ang := num;
 end;
end;

(*drawcurs - draw an arrow emanating from
            the point (x,y) with angle
            s. the p argument tells what
            color to draw the arrow -
            black(erase) or white.
*)
procedure drawcurs(x,y,s:integer;p:screencolor);
const lgth = 6;  plgth = 3;
begin
  pencolor(none); moveto(x,y);
  turnto(s);
  pencolor(p);    move(lgth);
  turn(backup);   move(plgth);
  turn(lhoriz);   move(plgth);
  turn(vert);     move(plgth);
end;

(*keeppt - keep this point. save
           coordinates. update ptcnt.
           adjust angle and save. draw
           cursor.
*)
procedure keeppt;
begin(*keeppt*)
 ptcnt := ptcnt +1;
 if ptcnt <= pl then begin
  if ptcnt = 1 then begin
   prevang := 0;  ang := 0;
  end else if ptcnt = 2 then prevang := ang
  else if ptcnt > 2 then begin
   prevang := s^.pts[3,ptcnt-2];
   ang := s^.pts[3,ptcnt-1];
  end;
  readjoy;
  if valid(xpad,ypad) then begin
   s^.pts[1,ptcnt] := xpad;
   s^.pts[2,ptcnt] := ypad;
   msg(info,25);
   typein; getsym;
   if sym = yesym then begin
    msg(info,26);
    typein;
    if not scanns then begin
     msg(warn,4);
     strkerr := true;
```

```
    end;
   end;
   if strkerr = false then begin
    s^.pts[3,ptcnt] := ang;
    drawcurs(xpad,ypad,ang,white);
   end else begin
    strkerr := false; (*minor error*)
    ptcnt := ptcnt - 1;
   end;
  end else begin
   ptcnt := ptcnt - 1;
   msg(warn,27);
  end;
 end else begin
  strkerr := true;
  msg(warn,28);
  ptcnt := ptcnt - 1;
 end;
end;

(*strokes - collect the points that
           form the strokes of the letter
*)
procedure strokes;
var v,strkend:boolean;

 (*lite - lite points on the screen as
          user moves joystick. only light
          if different from previous point
          and within grid, return true
          if point was lit
 *)
 function lite:boolean;
  begin
   if (not valid(xpad,ypad)) or ((xpad=prevx) and (ypad=prevy))
    then lite := false
   else begin
    pencolor(none); moveto(xpad,ypad);
    pencolor(white);moveto(xpad,ypad);
    lite := true;
   end;
   prevx := xpad; prevy := ypad;
  end;

 (*unlight - turnoff previous point
 *)
 procedure unlight;
  begin
   pencolor(none); moveto(prevx,prevy);
   pencolor(black);moveto(prevx,prevy);
  end;
```

```
    (*position - allow joystik to be moved
                 without remembering the points
    *)
     procedure position;
      begin
       readjoy;
       if prevb then unlight;
       prevb := lite;
      end;

   begin (*strokes*)
    prevb := false;
    strkend := false;
    msg(warn,24); (*clear off msg area*)
    repeat position until keypress;
    typein; getsym;
    if sym = begstk then begin
     ptcnt := 0;
     s := t; (* next stroke link*)
     new(t);
     s^.slk := t;
     strkcnt := strkcnt +1;
     keeppt;
     if ptcnt = 1 then repeat
      prevb := false;
      msg(warn,24); (*clear off msg area*)
      repeat position until keypress;
      typein; getsym;
      if sym = keep then keeppt
      else if sym = endstk then begin
       keeppt;  strkend := true;
       s^.pn := ptcnt;
      end else if sym <> stop then msg(warn,4);
      if (sym = stop) and (strkend = false) then begin
        sym := err;
        msg(warn,39);
      end;
     until (strkerr=true) or (strkend=true) or (sym = stop)
     else strkerr := true;
    end else if sym <> stop then msg(warn,4);
   end;

(*connect - connect the points
    7-feb-81
   26-feb-81
*)

   procedure connect;
   var ptn:integer;
       nonum:boolean;
```

```
(*store - store pen pattern and point number(n)
          on the screen at the position indicated by
          x,y.
*)
procedure store(x,y,n:integer);
var v:boolean;
  xn,yn:integer;

 (*stpoint - store the point in white
             if b is true, in black if
             b is false
 *)
 procedure stpoint(x,y:integer;b:boolean);
 begin
  if (valid(x,y)) then begin
   pencolor(none);  moveto(x,y);
   if b then begin
    pencolor(white); moveto(x,y);
   end else begin
    pencolor(black); moveto(x,y);
   end;
  end else note(1,1);
 end;

 (*stptnum - store the point number next
             to the point
 *)
 procedure stptnum(x,y,n:integer);
 var i2,i3,a,b:integer;
     bit: boolean;
 begin
  i3 := 6;
  repeat
   for i2:= 1 to 3 do begin
    a := i2-1;  b := i3-1;
    bit := pns[n,i2,i3];
    stpoint(x+a,y+b,bit);
   end;
   i3 := i3-1;
  until i3 = 0;
 end;

 (*dbldig - store the 2nd digit of point
            number
 *)
 procedure dbldig(x,y,n:integer);
 var m:integer;
 begin
  m := n div 10;
  stptnum(x,y,m);
  m := n mod 10;
```

```
  stptnum(x+4,y,m);
 end;

begin(*store*)
 (*store pen pattern at point x,y*)
 v:= true;
 stpoint(x,y,v);
 if pen <> lin then begin
  (*light pen pattern*)
  stpoint(x+1,y,v);
  stpoint(x-1,y,v);
  if pen <> light then begin
   (*medium pen pattern*)
   stpoint(x-2,y,v);
   stpoint(x+2,y,v);
   stpoint(x,y+1,v);
   stpoint(x+1,y+1,v);
   stpoint(x-1,y+1,v);
   stpoint(x+2,y+1,v);
   stpoint(x-2,y+1,v);
   stpoint(x,y-1,v);
   stpoint(x-1,y-1,v);
   stpoint(x+1,y-1,v);
   stpoint(x+2,y-1,v);
   stpoint(x-2,y-1,v);
   if pen <> med then begin
    stpoint(x-3,y,v);
    stpoint(x+3,y,v);
    stpoint(x+3,y+1,v);
    stpoint(x-3,y+1,v);
    stpoint(x+3,y-1,v);
    stpoint(x-3,y-1,v);
    xn :=x+5; yn :=y+2;
   end else begin xn :=x+4; yn :=y+2 end
  end else begin xn :=x+3; yn :=y+1 end
 end else begin xn :=x+2; yn :=y+1 end;
 if n > 9 then dbldig(xn,yn,n)
 else if n <> Ø then stptnum(xn,yn,n);
end;


(*piece - piece the letter together
          taking 2 points at a time
          n - pt # of x1,y1
          x1,y1 - coordinates of point 1
          x2,y2 - coordinates of point 2
          tmax - upper limit of parameter range
          sl1 - slope at point 1
          sl2 - slope at point 2
          d1,d2 are the angles associated
          with sl1,sl2
```

```
*)
procedure piece(x1,y1,sl1,tmax,x2,y2,sl2:real;n,d1,d2:integer);
var x,y:integer;
    top,bot:real;
    xa0,xa1,xa2,xa3,ya0,ya1,ya2,ya3:real;
    t,rx,ry,tsq,tcub,t4,tmsq,tm3,tm4:real;

(*racoeff - calculate the coefficients for the function
           for this pair of points
*)
procedure racoeff;
begin
 tmsq := tmax*tmax; tm3 := tmsq*tmax;
 tm4 := tm3*tmax;
 xa0 := x1;
 xa1 := 1.0;
 ya0 := y1;
 ya1 := sl1;
 (*equations for coefficients xa2,xa3 and ya2, ya3
 *)
 top := 1.0/tmax;
 bot := (2*1.0)/tmax;
 xa2:= ((3*(x2-x1))/tmsq)- bot- top;
 top := sl2/tmax;
 bot := (2*sl1)/tmax;
 ya2:= ((3*(y2-y1))/tmsq)- bot- top;
 top := 1.0/tmsq;
 bot := 1.0/tmsq;
 xa3:= ((2*(x1-x2))/tm3)+ bot+ top;
 top := sl2/tmsq;
 bot := sl1/tmsq;
 ya3:= ((2*(y1-y2))/tm3)+ bot+ top;
end;

(*straitlin - if the user specifies the same angle between two
             points, a straight line is attempted between the
             two points.
*)
procedure straitlin;
var tx,ty,quad:integer;
    exit:boolean;
begin
 quad:=0;  exit:=false;
 (*reposition turtle to point x1,y1*)
 pencolor(none);
 moveto(x,y);
 turnto(d1);

 x:=round(x2);
 y:=round(y2);
 if d1 <= 90 then quad:=1
```

```
 else if (d1>90) and (d1<=180) then quad:=2
 else if (d1>180)and (d1<=270) then quad:=3
 else quad:=4;
 repeat
  move(1);
  tx:=turtlex;
  ty:=turtley;
  case quad of
   1: if (tx>x) or (ty>y) then exit:=true;
   2: if (tx<x) or (ty>y) then exit:=true;
   3: if (tx<x) or (ty<y) then exit:=true;
   4: if (tx>x) or (ty<y) then exit:=true;
  end;
  if exit=false then begin
   store(tx,ty,0);
   pencolor(none);
   moveto(tx,ty);
   turnto(d1);
  end;
 until exit=true;
end;

begin(*piece*)
 x := round(x1);
 y := round(y1);
 if nonum = true then store(x,y,0)
 else store(x,y,n);

 if d1=d2 then straitlin
 else begin
  racoeff;
  t := 0.0;
  top := 0.07;

  repeat
   tsq := t*t; tcub := tsq*t; t4 := tcub*t;
   rx := xa0 + xa1*t + xa2*tsq + xa3*tcub;
   x := round(rx);
   ry := ya0 + ya1*t + ya2*tsq + ya3*tcub;
   y := round(ry);
   store(x,y,0);
   t := top + t;
   if t>tmax then
    if (t-top) < tmax then t := tmax;
  until t > tmax;
 end;

 if (n=ptn-1) then begin
  x := round(x2);
  y := round(y2);
  if nonum = true then store(x,y,0)
```

```
  else store(x,y,ptn);
 end;
end;

(*convrt - convert angles to tangent slopes.
*)
function convrt(a:real):real;
const rad = 57.2957795;
var n1,n2,n3:real;
begin
 if (a=90.0) or (a=270.0) then begin
  if a=90.0 then convrt := 344.0
  else convrt := -344.0;
 end else begin
  n1 := a/rad;
  n2 := sin(n1);
  n3 := cos(n1);
  convrt := n2/n3;
 end;
end;

(*spline - called for each stroke.
          s points to the link for the
          stroke.
          spline passes points two at a time
          to piece to produce a spline piece.
          tmax is 1, the maximum of the parameter range.
*)
procedure spline;
var i,d1,d2:integer;
    sqx,sqy:real;
    x1,y1,sl1,tmax,x2,y2,sl2:real;
begin
 ptn := s^.pn;
 for i := 1 to (ptn-1) do begin
  x1:=s^.pts[1,i]; y1:=s^.pts[2,i];
  sl1:=s^.pts[3,i];
  d1 :=s^.pts[3,i];
  sl1:=convrt(sl1);
  x2:=s^.pts[1,i+1]; y2:=s^.pts[2,i+1];
  sl2:=s^.pts[3,i+1];
  d2 :=s^.pts[3,i+1];
  sl2:=convrt(sl2);
  (*tmax - upper end of each intrval for t*)
  tmax := 1.0;
  piece(x1,y1,sl1,tmax,x2,y2,sl2,i,d1,d2);
 end;
end;

begin(*connect*)
 getsym;
```

```
    if sym = minus then nonum := true
    else nonum := false;
    if (dct > 0) or (ict > 0) then merge;
    if strkcnt = 0 then msg(warn,40)
    else begin
     msg(info,29);
     t := abc.stks;
     while (t^.slk <> nil) do begin
      s := t^.slk;
      spline;
      t := s;
     end;
    end;
   end;

(*alter - alter the points.
         these routines process the commands from the
         alter menu.
         alter angle (a), insert point(i),
         delete point (d), move point (m)
         retreive point info (r), quit (q)
         (the quit command returns control to the joy menu)
  27 jan 81
  5 feb 81
*)
  procedure alter;
  var st,pt:integer;
      alterr:boolean;
      ix,iy,ia:integer;

   (*scanstpt - scan typein for - stroke#,
                point#. ch at exit contains
                the character after point#.
                return true if stroke # is
                valid and point# is at least
                within point limit else
                return false.
   *)
   function scanstpt:boolean;
   begin
    getsym;
    if sym = number then begin
     st := num;
     if st > abc.sn then scanstpt := false
     else begin
      getsym;
      if sym <> comma then scanstpt := false
      else begin
       getsym;
       if sym = number then begin
        pt := num;
```

```
      if pt > pl then scanstpt := false
      else scanstpt := true
     end else scanstpt := false;
    end;
   end;
 end else scanstpt := false;
end;

(*scanxy - read input to get x,y values
*)
function scanxy:boolean;
var ix,iy:integer;
begin
 getsym;
 if sym = number then begin
  xpad := num;
  getsym;
  if sym = comma then begin
   getsym;
   if sym = number then begin
    ypad := num;
    scanxy := valid(xpad,ypad);
   end else scanxy := false;
  end else scanxy := false;
 end else scanxy := false;
end;

(*scana - read input to get angle value
*)
function scana:boolean;
begin
 getsym;
 if sym = comma then begin
  ang := s^.pts[3,pt];
  if pt > 2 then prevang := s^.pts[3,pt-1]
  else prevang := 0;
  if scanns = false then scana := false
  else scana := true;
 end else scana := false;
end;

(*typeit - type the information retrieved by the retrieve info
          command. the message at the bottom line of the screen
          x = 124,  y = 93, angle = 120.
*)
procedure typeit;
var s3:string[5];

 (*numstr - convert a number to a typeable form
 *)
 procedure numstr(i:integer);
```

```
  var q,r,j:integer;
  begin
   s3 := '      '; j := 5;
   while (i > 9) and (j > Ø) do begin
    q := i div 1Ø;
    r := i mod 1Ø;
    s3[j] := chr(r + ord('Ø'));
    j := j - 1;
    i := q;
   end;
   if j = Ø then msg(deb,1)
   else if (j=5) and (i=Ø) then s3[j] := chr(ord('Ø'))
   else if i > Ø then s3[j] := chr(i +ord('Ø'));
  end;

 begin
  numstr(ix);
  pencolor(none); moveto(lmsg,bmsg);
  pencolor(white);
  wstring('x =');
  wstring(s3);
  numstr(iy);
  wstring(', y =');
  wstring(s3);
  numstr(ia);
  wstring(', ang =');
  wstring(s3);
 end;

 (*erascurs - erase the arrow at the current position
 *)
 procedure erascurs;
 begin
  ix := s^.pts[1,pt];
  iy := s^.pts[2,pt];
  ia := s^.pts[3,pt];
  drawcurs(ix,iy,ia,black);
 end;

begin(*alter*)
 displaym(altermenu);
 typein; getsym;
 while sym <> qt do begin
  alterr := false;
  msg(warn,24);
  if sym = adjang then begin
   if scanstpt = true then begin
    find(st,abc.stks,s);
    if pt > s^.pn then alterr := true
    else begin
     if scana = false then alterr := true
```

```
    else begin
     erascurs;
     drawcurs(ix,iy,ang,white);
     s^.pts[3,pt] := ang;
    end;
   end;
  end else alterr := true;
end else

if sym = inspt then begin
 if ict < il then begin
  if scanstpt = true then begin
   getsym;
   if sym = comma then begin
    find(st,abc.stks,s);
    if pt > s^.pn then alterr := true
    else begin
     if scanxy = true then begin
      if scana = true then begin
       ict := ict + 1;
       insert[1,ict] := xpad;
       insert[2,ict] := ypad;
       insert[3,ict] := ang;
       insert[4,ict] := pt;
       insert[5,ict] := st;
       drawcurs(xpad,ypad,ang,white);
      end else alterr := true;
     end else alterr := true;
    end;
   end else alterr := true;
  end else alterr := true;
 end else msg(warn,31);
end else

if sym = delete then begin
 if scanstpt = true then begin
  find(st,abc.stks,s);
  if pt > s^.pn then alterr := true
  else begin
   if dct + 1> s^.pn then msg(warn,33)
   else begin
    erascurs;
    s^.pts[1,pt] := 0;
    s^.pts[2,pt] := 0;
    s^.pts[3,pt] := 0;
    dct := dct + 1;
   end;
  end;
 end else alterr := true;
end else
```

```
  if sym = movpt then begin
   if scanstpt = true then begin
    getsym;
    if sym = comma then begin
     find(st,abc.stks,s);
     if pt>s^.pn then alterr := true
     else begin
      if scanxy = true then begin
       if scana = true then begin
        erascurs;
        s^.pts[1,pt] := xpad;
        s^.pts[2,pt] := ypad;
        s^.pts[3,pt] := ang;
        drawcurs(xpad,ypad,ang,white);
       end else alterr := true;
      end else alterr := true;
     end;
    end else alterr := true;
   end else alterr := true;
  end else

  if sym = infop then begin
   if scanstpt = true then begin
    find(st,abc.stks,s);
    if pt > s^.pn then alterr := true
    else begin
     ix := s^.pts[1,pt];
     iy := s^.pts[2,pt];
     ia := s^.pts[3,pt];
     typeit;
    end;
   end else alterr := true;
  end else msg(warn,4);

  if alterr = true then begin
   msg(warn,30);
   alterr := false;
  end;
  typein;
  getsym;
 end; (*while*)
end;


begin(*draw*)
 prevx := 0; prevy := 0; prevb := false;
 ict := 0; dct := 0;
 strkerr := false;
 displaym(joymenu);
 typein; getsym;
```

```
  while not (sym in [qt,conn,chgmen]) do begin
   if sym = start then begin
    mark(heap); heapinit := true;
    new(s);
    abc.stks := s; (*header*)
    new(t);
    s^.slk := t;
    strkcnt := 0;
    repeat strokes until (strkerr=true) or (sym=stop);
    s^.slk := nil;  abc.sn := strkcnt;
    if sym <> stop then sym := qt;
   end else begin
    msg(warn,4);
    typein; getsym;
   end;
   if sym = stop then begin
    typein; getsym;
   end;
  end;(*while*)
  while sym <> qt do begin
   if sym = conn then connect
   else  if sym = chgmen then begin
    alter;
    displaym(joymenu);
   end else msg(warn,4);
   typein; getsym;
  end;
  if (strkerr = true) and (heaprel = false)
     and (heapinit =true) then begin
   release(heap);
   heaprel := true;
   msg(warn,28);
  end;
 end;

begin(*dmenu*)
 displaym(drawmenu);
 typein;
 getsym;
 while not (sym in [quitsym,changesym]) do begin
  if sym = drawsym then begin
   draw;
   displaym(drawmenu);
  end else
  if sym = slsym then begin
   getsym;
   fnme := strtg;
   getsym;
   lnme := strtg;
   str12:= concat(fflop,fnme,'.',lnme);
   rewrite(lettfile,str12);
```

```
 i:=ioresult;
 if i=0 then begin
  merge;
  t:=abc.stks;
  repeat
   s:=t^.slk;
   xyz.fpn := s^.pn;
   xyz.fpts:= s^.pts;
   lettfile^:=xyz;
   put(lettfile);
   i:=ioresult;
   if i>0 then msg(warn,22);
   t:=s;
  until (t^.slk=nil);
  close(lettfile,lock);
 end else msg(warn,21);
end else
if sym = glsym then begin
 getsym;
 fnme:=strtg;
 getsym;
 lnme:=strtg;
 abc.nme:=lnme;
 str12:=concat(fflop,fnme,'.',lnme);
 reset(lettfile,str12);
 i:=ioresult;
 if i=0 then begin
  if heapinit=true then release(heap);
  mark(heap);
  heapinit:=true;
  new(s);
  abc.stks:=s;
  new(t);
  s^.slk:=t;
  strkcnt:=0;
  while not eof(lettfile) do begin
   s:=t; new(t); s^.slk:=t;
   xyz:= lettfile^;
   strkcnt:=strkcnt+1;
   s^.pn:=xyz.fpn;
   s^.pts:=xyz.fpts;
   get(lettfile);
   i:=ioresult;
   if i>0 then msg(warn,23);
  end;
  s^.slk:=nil;
  abc.sn:=strkcnt;
  close(lettfile,lock);
 end else msg(warn,21);
end else msg(warn,4);
if u=zapf then msg(warn,24)
```

```
    else msg(info,10);
    typein;
    getsym;
   end;
   if sym = changesym then sym := specsym;
  end;


(* pause  -  wait for input from keyboard *)
procedure pause;
var c: integer;
begin
 c:= 1;
 repeat c:= c + 1;
 until keypress;
end;

(* delay - allow user to read msg before
           writing over it
*)
procedure delay;
begin
 while n > 0 do
  n := n-1;
end;

(* erase  -  erases whole screen *)
procedure erase;
begin
 viewport(0,xmax,0,ymax);
 fillscreen(black);
end;

(* erasgd -  erases grid area of screen *)
procedure erasgd;
var r,b:integer;
begin
 r := lmen-1;  b := tmsg+8;
 viewport(0,r,b,ymax);
 fillscreen(black);
 viewport(0,xmax,0,ymax);
end;

(* erasmn -  erases menu area of screen *)
procedure erasmn;
begin
 viewport(lmen,rmen,bmen,tmen);
 fillscreen(black);
 viewport(0,xmax,0,ymax);
end;
```

```
(* erasmg -  erases msg area of screen *)
procedure erasmg;
begin
 viewport(lmsg,rmsg,bmsg,tmsg+chght);
 fillscreen(black);
 viewport(0,xmax,0,ymax);
end;

(* cleanup - reset and exit     *)
procedure cleanup;
begin
 erase;
 textmode;
 writeln('letters are things, not pictures of things');
end;

(* gline  -  puts short lines on grid *)
(*           x,y  - left for horizontal lines*)
(*                  bottom for vertical*)
(*           dash - length of lines*)
(*           dir  - horizontal or vertical*)
(*           bound- maximum x or y, dep on direction of line*)

procedure gline;
const twoquad = 180;
begin
 pencolor(none);
 moveto(x,y);
 turnto(dir);
 pencolor(white);
 move(dash);
 pencolor(none);
 move(bound-dash);
 turnto(dir+twoquad);
 pencolor(white);
 move(dash);
end;

(*typein - procedure to prompt user
           and pick up subsequent
           type in
*)
procedure typein;
var l:integer;
    b:boolean;
begin
 note(10,10);    (*a little noise too*)
 msg(info,0);    (*put out prompt*)
 lln := 0;  l:= 0; b := false;

 pause;          (*wait for typing*)
```

```
while l < ic do
 begin read(keyboard,ch);
  if eoln(keyboard) then l := ic
  else
   begin pencolor(none);
    if ch = chr(bs) then
     begin lln := lln - 1;
      if lln < 0 then
       begin note(10,10); lln := 0; end;
      ch := ' '; b := true;
     end;
    moveto(lln*chght+lmsg,tmsg);
    pencolor(white); wchar(ch);
    if b = true then b := false
    else begin
     lln := lln + 1;
     line[lln] := ch;
    end;
    l := lln;
   end;
  end;
  if l = lln then msg(warn,7); (*tried to type too much*)
  ccn := 1;
  if (lln = 0) and (l = ic) then ch := '!'
  else ch := line[ccn];
end;

(*getsym - procedure to pick up symbols
     from input line
*)
procedure getsym;
var i,j,k:integer;

 procedure getch;
 begin
  if ccn = lln then ch := '!'
  else
   begin ccn := ccn + 1;
    ch := line[ccn];
   end;
 end;
begin (*getsym*)
 while ch = ' ' do getch;
 if ch in ['a'..'z'] then
  begin i := 0;
  (*pick up a symbol*)
   repeat if i < rwl then
    begin i := i +1; tg[i] := ch;
     strtg[i]:=ch;
    end;
```

```
    getch;
   until not (ch in ['a'..'z']);
  (*blank out unused portions*)
   if i >= ii then ii := i else
    repeat tg[ii] := ' '; ii := ii -1;
    until ii = i;
   j := 1; k := nrw;
  (*search for the symbol*)
   repeat i := (j+k) div 2;
    if tg <= tags[i] then k := i-1;
    if tg >= tags[i] then j := i+1;
   until j > k;
   if j-1 > k then sym := wsym[i] else sym := nul;
 end else
 if ch in ['0'..'9'] then
  begin i := 0; num := 0; sym := number;
  (*pick up a number*)
   repeat num := 10*num + (ord(ch) - ord('0'));
    i := i+1; getch;
   until not (ch in ['0'..'9']);
   if i > nmax then msg(warn,9);
 end else
  begin sym := schr[ch];
   getch;
  end;
end;

(* msg - routine to put messages out for
       the user. the message area is the
       bottom two lines of the screen. the
       bottom line will contain error mesages
       from the program. the line above
       will contain the prompt and echo
       the users type in.  lvl - is the
       severity of the message, num - is
       an index into the errmsg table
*)
procedure msg;
var emsg:string[40];
  procedure wmsg;
  begin
   moveto(lmsg,bmsg);  pencolor(white);
   emsg := errs[num];  wstring(emsg);
  end;
begin
 pencolor(none);
 if lvl = info then begin
  if num = 0 then begin
   moveto(lmsg,tmsg);
   pencolor(white);
   wstring('*                                                    ');
```

```
   end else if u = albertini then wmsg;
  end else
  if lvl = deb then begin
   moveto(ldeb,tdeb); pencolor(white);
   emsg := debmsg[num];
   wstring(emsg);
   delay(80);
  end else
  if lvl = warn then begin
   wmsg;
   if num <> 24 then sym := err;
  end;
end;


(* displaym - display the menu indicated
        by the menutype
*)
procedure displaym;
var curs:integer;
 procedure skip;
 begin
  pencolor(none);
  curs := curs - spce;
  moveto(lmen,curs);
  pencolor(white);
 end;

 procedure menu1;
 begin
   wstring('menu #1 - specs  '); skip;
   wstring('cmd - explanation'); skip;
   skip;
   wstring('gd - grid specs  '); skip;
   wstring('pn - pen specs   '); skip;
   wstring('um - user mode   '); skip;
   wstring('ss - save specs  '); skip;
   wstring('gs - get specs   '); skip;
   wstring('cm - chg men-draw'); skip;
   wstring('qt - quit        ');
 end;

begin
 erasmn; (*clear menu area*)
 curs := tmen + spce;
 skip;
 if w = specmenu then menu1
 else if w = drawmenu then
  begin
   wstring('menu #2 - draw   '); skip;
```

```
    wstring('cmd - explanation'); skip;
    skip;
    wstring('dw - draw letter '); skip;
    wstring('sl - save letter '); skip;
    wstring('gl - get letter  '); skip;
    wstring('cm - chg men-spec'); skip;
    wstring('qt - quit        ');
  end else if w = joymenu then
   begin
    wstring('menu #2a - joy   '); skip;
    wstring(' specify pts in  '); skip;
    wstring(' drawing order   '); skip;
    wstring('1. s start letter'); skip;
    wstring('2.   move joystik'); skip;
    wstring('3. b begin stroke'); skip;
    wstring('4.   adjust angle'); skip;
    wstring('5.   move joystik'); skip;
    wstring('6. k keep point  '); skip;
    wstring('7.   repeat 5,6,4'); skip;
    wstring('8. e end stroke  '); skip;
    wstring('9.   repeat 4,3-8'); skip;
    wstring('10.p stop letter '); skip;
    wstring('11.c connect pts '); skip;
    wstring('12.g chg men-altr'); skip;
    wstring('13.q quit        ');
  end else
   begin
    wstring('menu #2b - alter '); skip;
    wstring('  s = stroke num '); skip;
    wstring('  p = point num  '); skip;
    wstring('  a = angle      '); skip;
    skip;
    wstring('a - adjust angle '); skip;
    wstring('a s,p,a          '); skip;
    wstring('i - insert point '); skip;
    wstring('i s,p,x,y,a      '); skip;
    wstring('d - delete point '); skip;
    wstring('d s,p            '); skip;
    wstring('m - move point   '); skip;
    wstring('m s,p,x,y,a      '); skip;
    wstring('r - get pointinfo'); skip;
    wstring('r s,p            '); skip;
    wstring('q - quit         ');
   end;
  msg(info,6);
end;

(*find - find stroke #n starting from
        pointer a and return the
        found stroke in the pointer s
*)
```

```
procedure find;
var i:integer;
begin
 for i := 1 to n do begin
  s := a^.slk;
  a := s;
 end;
end;

(*merge - merge letter points with points
          deleted and points inserted.
          first compress array by removing
          deleted points (x=Ø,y=Ø,ang=Ø).
          then combine inserted points
          with original points.
          called by connect before connecting
          and by save letter before
          saving.
*)
procedure merge;
var st,pt,i,j:integer;
    mergerr:boolean;
begin
 msg(info,32);
 mergerr := false;
 if dct <> Ø then begin
  t := abc.stks;
  i := 1;
  repeat
   s := t^.slk;
   for j := 1 to s^.pn do begin
    if (s^.pts[1,j] = Ø) and
       (s^.pts[2,j] = Ø) and
       (s^.pts[3,j] = Ø) then dct := dct -1
    else begin
     combine[1,i] := s^.pts[1,j];
     combine[2,i] := s^.pts[2,j];
     combine[3,i] := s^.pts[3,j];
     i := i + 1;
    end;
   end;
   s^.pts := combine;
   s^.pn := i;
   i := 1;
   t := s;
  until (dct = Ø) or (t^.slk = nil);
  if dct <> Ø then begin
   mergerr := true;
   msg(warn,34);
  end;
 end; (*dealing with deletes*)
```

```
if (mergerr = false) and (ict <> 0) then begin
 repeat
  st := insert[5,ict];
  pt := insert[4,ict];
  find(st,abc.stks,s);
  if pt = 1 then begin
   combine[1,pt] :=insert[1,ict];
   combine[2,pt] :=insert[2,ict];
   combine[3,pt] :=insert[3,ict];
  end else begin
   for i := 1 to (pt-1) do begin
    combine[1,i] :=s^.pts[1,i];
    combine[2,i] :=s^.pts[2,i];
    combine[3,i] :=s^.pts[3,i];
   end;
   combine[1,pt] :=insert[1,ict];
   combine[2,pt] :=insert[2,ict];
   combine[3,pt] :=insert[3,ict];
  end;
  if pt <= s^.pn then begin
  i := pt; j := pt+1;
   repeat
    combine[1,j] :=s^.pts[1,i];
    combine[2,j] :=s^.pts[2,i];
    combine[3,j] :=s^.pts[3,i];
    j := j + 1;
    if j > pl then mergerr := true;
    i := i + 1;
   until (i > s^.pn) or (mergerr = true);
  end;
  s^.pts := combine;
  s^.pn := j-1;
  ict := ict - 1;
 until (ict = 0) or (mergerr = true);
 if mergerr = true then msg(warn,28);
end;(*dealing with inserts*)
end;
```

```
begin(*typdesngr*)
 initial;
 smenu;
 while sym <> quitsym do
  if sym=specsym then smenu
  else dmenu;
 if (heaprel=false) and (heapinit=true) then release(heap);
```

```
  cleanup;
end.
```

Appendix C: Sample Letters

This appendix contains some sample letters designed using TYPDESNGR and copied from the APPLE screen with a Tektronix 4632 Hard Copy Video Unit. The letters included in this appendix were chosen to illustrate some of TYPDESNGR's capabilities. Each figure is described briefly below.

Figure 1: Lower case c - points only.
This figure shows a set of points that were placed through commands from the Joy menu. The points that will be connected to form the lower case c are at the base of the arrows.
The arrows reflect the angle that the tangent line makes with the curve at the point. The grid is the default grid.

Figure 2: Lower case c - points connected, point numbers shown, medium pen.
This figure shows how TYPDESNGR connects the points of Figure 1 with a medium pen. The connect command was issued with no arguments, therefore, the point numbers appear near the defining points to aid the user with further modifications. This letter uses the default grid and pen settings.

Figure 3: Lower case c - no point numbers, bold pen.
This figure shows the c of figures 1 and 2 without the point numbers. This letter was drawn with a bold pen and the default grid settings were used. Design time was about 1/2 hour.

Figure 4: Upper case K - medium pen.
This figure shows the italic K designed in section 3.1.5. This was the first letter designed with TYPDESNGR and benefited from extensive criticism. Total design time was probably about 3 hours. Default pen and grid settings were used.

Figure 5: Upper case K - bold pen.
This figure shows the upper case K of Figure 4 drawn with a bold pen.

Figure 1: Lower case c - points only.

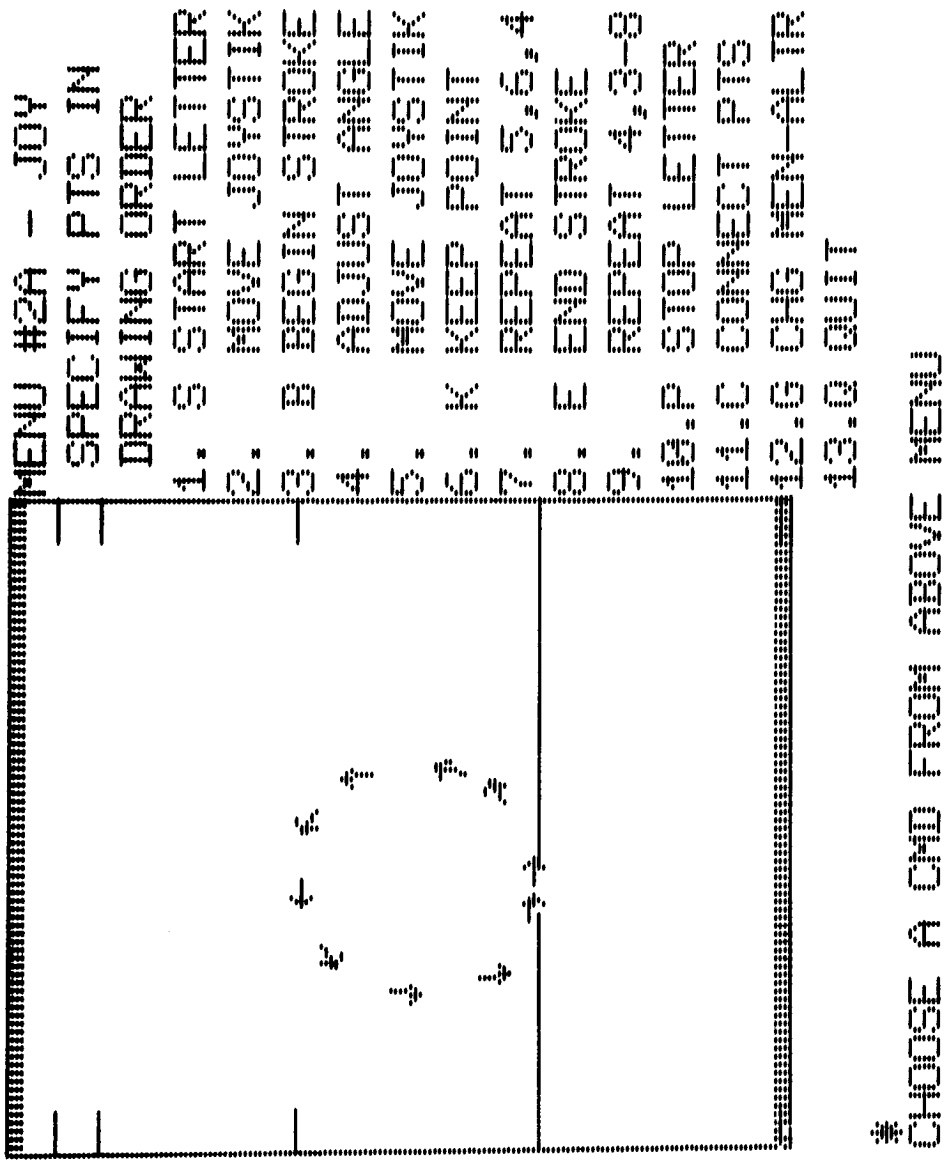

Figure 2: Lower case c - points connected, point numbers, medium pen.

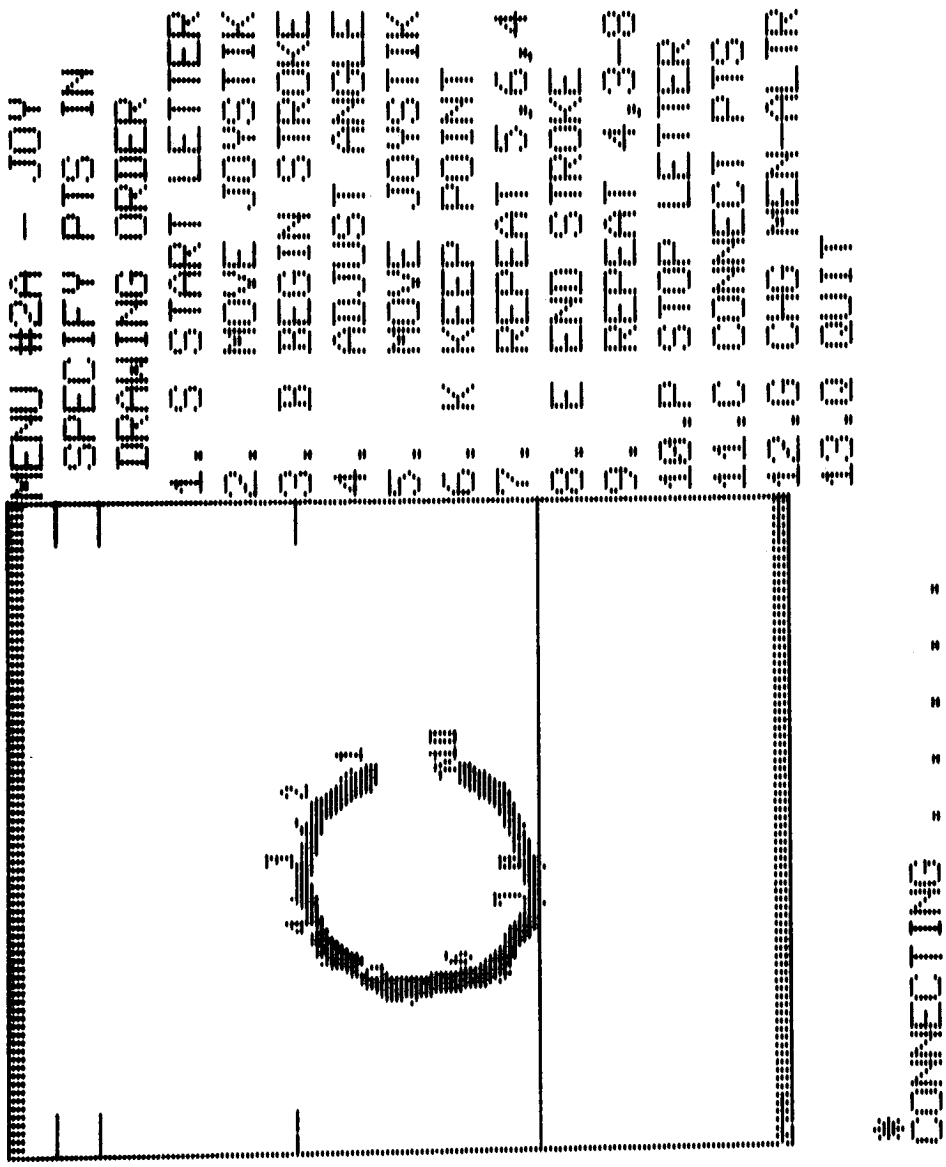

Figure 3: Lower case c - points connected, bold pen.

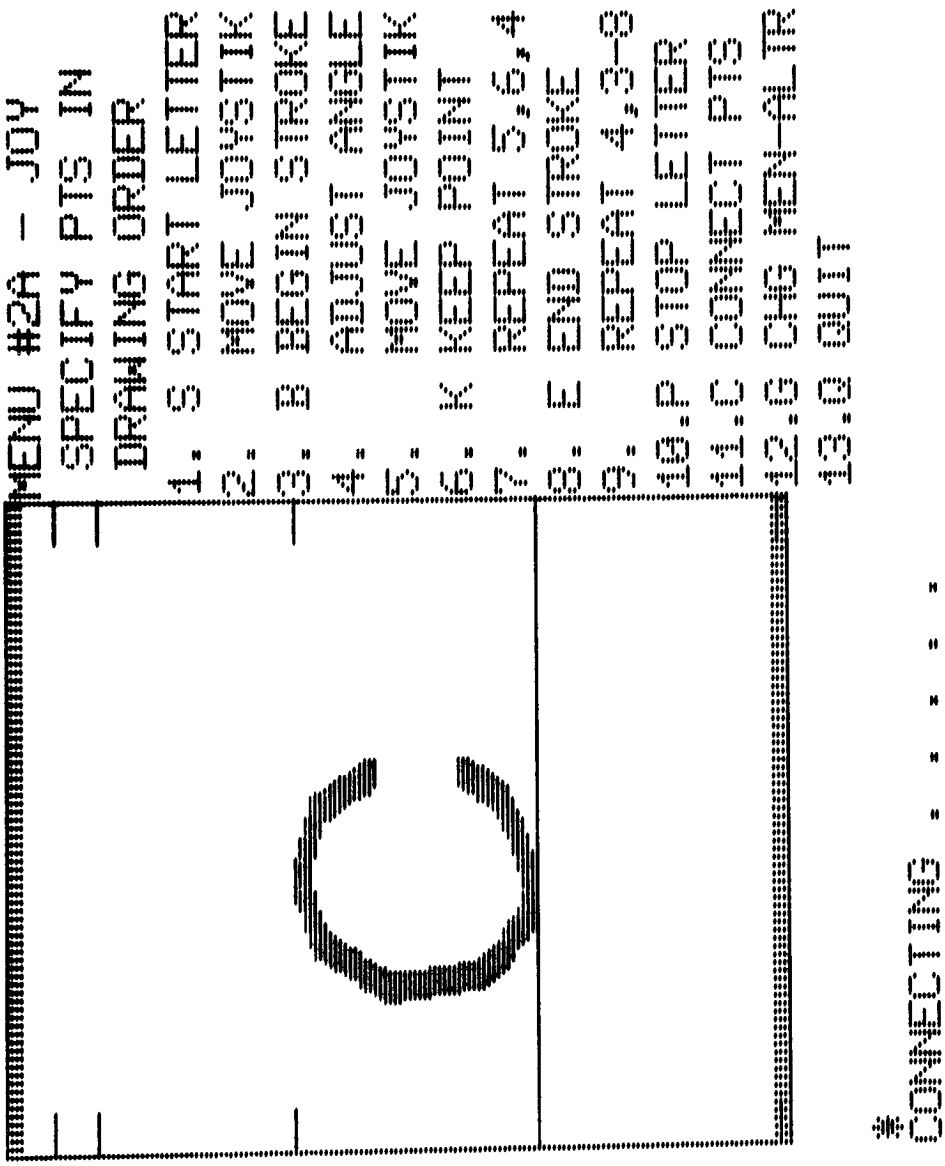

Figure 4: Upper case X - medium pen.

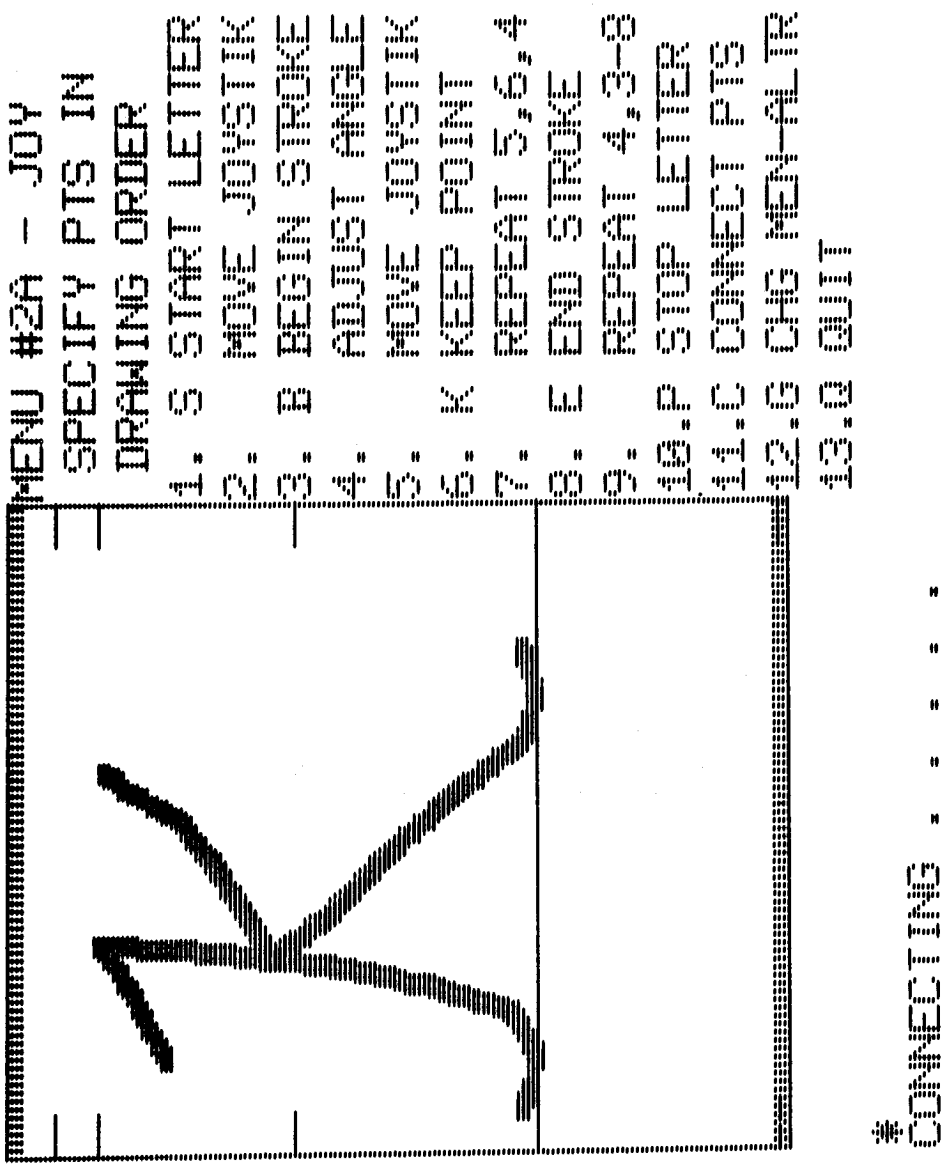

Figure 8: Upper case X - bold pen.