July 1983

Report No. STAN-CS-83-974

A Chinese Meta-Font

by

John Hobby and Gu Guoan

Department of Computer Science

Stanford University
Stanford, CA 94305

A Chinese Meta-Font

John Hobby and Gu Guoan

Abstract

METRFONT is Donald E. Knuth's system for alphabet design. The system allows an entire family of fonts or "meta-fonts" to be specified precisely and mathematically so that it can be produced in different sizes and styles for different raster devices.

We present a new technique for defining Chinese characters hierarchically with METAFONT. We define METAFONT subroutines for commonly used portions of strokes and then combine some of these into routines for drawing complete strokes. Parameters describe the skeletons of the strokes and the stroke routines are carefully designed to transform themselves appropriately. This allows us to handle all of the basic strokes with only 14 different **routines.**

The stroke routines in turn are used to build up groups of strokes and radicals. Special routines for positioning control points ensure that the strokes will join properly in a variety of different styles. The radical routines are parameterized to allow them to be placed at different locations in the typeface and to allow for adjusting their size **and shape.** Key points are positioned relative to the bounding box for the radical, and the special positioning routines find other points that must be passed to the stroke routines.

We use this method to design high quality Song style characters. Global parameters control the style, and we show how these can be used to create Song and Long Song from the same designs. Other settings can produce other familiar styles or even new styles. We show how it is possible to create completely different styles, such as Bold style, **merely** by substituting different stroke routines. The global parameters can be used to augment simple scaling by altering stroke width and other details to account for changes in size. We can adjust stroke widths to help even out the overall darkness of the characters. We also show how it is possible to experiment with new ideas such as adjusting character widths individually.

While many of our characters are based on existing designs, the stroke routines facilitate the design of new characters without the need to refer to detailed drawings. The skeletal parameters and special positioning routines make it easy to position the strokes properly. In our previous paper, in contrast to this, we parameterized the strokes according to their boundaries and copied an existing design. The previous approach made it very difficult to create different styles with the same METAFONT program.

This research was supported in part by National Science Foundation grant IST-820.1926 and by the Systems Development Foundation.

0. Introduction

Chinese character generation is a very important part of Chinese language computer systems, and it is complicated by the number and complexity of Chinese characters. Even simplified characters contain an average of about twelve strokes each, and a good printing system requires all four standard styles in different sizes, with at least 8,000 characters in each. The designs can be digitized using optical scanning, but this is expensive and the resulting characters must be edited individually.

METRFONT is a system for designing alphabets for raster devices so that a single mathematical description can be used for different sizes and styles of fonts on different devices [1]. While not designed explicitly for Chinese characters, METRFONT is a general system with features useful for Chinese character design.

Knuth's idea of a "meta-font" is to describe alphabets parametrically so that one routine can produce different styles of letters. In [2], Knuth explains how this can be done for Roman alphabets. We apply the same concept to Chinese characters, except that we also describe the radicals and the strokes parametrically. While there are really only a few kinds of strokes that are fundamentally different, similar strokes can vary significantly. We can therefore produce better strokes with fewer routines by parameterizing these differences. A similar type of parameterization also applies to radicals. We create a complete hierarchy starting with eight routines for parts of strokes and thirteen routines for complete strokes. This hierarchical organization not only simplifies the design process, it leads to more uniformity in the designs. The more complicated strokes are formed in the radical routines by combining the basic strokes, and more complicated radicals call routines that draw simple combinations of strokes. The basic strokes have special parameters that specify how they are being joined together so that they can draw the special features that appear near such joins. There are also support routines for calculating points used in the constructions and for positioning certain combinations of strokes, taking into account style parameters such as the stroke width.

Designing an entire set of 8,000 characters would be a rather large project. Instead we have designed a representative sample of about 140 radicals and 128 characters. Many of the radicals appear in more than one character, and many more characters could be formed from these radicals. This work is an extension of the ideas presented in [6], which showed how METRFONT could be used to copy specific Song style designs. Here we also make use of these designs as well as Bold style and Long Song style from the same source [5], but we only use them to get an idea of how the strokes should be placed and to determine how to set the parameters to the METRFONT programs so that they can produce the various styles. The Song and Bold style designs consist of a carefully chosen set of about 125 large characters superimposed on graph paper, and the Long Song designs were taken from large scale photographs.

The global font parameters that affect size and style are used mainly in the stroke routines themselves. For Song style, there are 68 global parameters that control the slant and aspect ratio, the stroke widths and amount of taper, the size and shape of the stroke end features, and various special properties of certain strokes.

The stroke routines have to be designed carefully to work properly for all reasonable settings of these parameters, and to join together properly even when stroke widths and shapes change. To achieve this, we use control points on the skeletons of the strokes and join strokes by placing the ending control point of one on the skeleton of the other. This is simpler and more flexible than the technique used in [6], where the parameters. described the edge of the stroke. The stroke routines also take fewer parameters, so that all details except the placement of the skeleton are controlled by the global font parameters.

Completely different styles can be produced from the same radical and character routines by substituting different stroke routines. It is apparent from [5] that the stroke skeletons are essentially the same in Bold style as in Song style, and many of the differences can be characterized in terms of a few simple rules that are used by the Bold stroke routines. The few differences that remain can be achieved by adding METRFONT conditional statements to key radical routines. There is more work to be done on how to describe the skeletal differences between styles, so we only present preliminary results here.

In section **1** we introduce METAFONT and LCCD. In section 2 we examine some of the basic stroke drawing routines to see how the style parameters are used. In section 3 we see how to combine the basic strokes into radicals; and in section **4** we discuss the choice of style parameters, adjustments for changing point size, and experiments with new styles. Appendix **1** presents examples of all the basic strokes in the Song, Long Song, and Bold styles. In appendix 2 we show all **128** characters at **10** and 18 points and in an example of actual Chinese text. Finally, in appendix 3 we give a sample of actual METRFONT programs for Chinese characters.

1. METRFONT and LCCD

METRFONT is an algebraic language with subroutines, variables, equations, conditional statements, and commands for describing letterforms. Equations are used in a declarative way to define the numerical values of variables and the coordinates of points. METAFONT will solve systems of linear equations, keeping track of linear constraints between variables until enough equations are given to determine their values.

Letterforms are actually created with "draw commands" that refer to points whose coordinates have been determined in the above manner. Draw commands work by moving a discrete "pen" along a path through the points and turning on all the pixels covered by the pen. The actual curve used is a piecewise cubic spline. The section of this spline between any two points i and j is defined by

$$x(t) = x_i + (3t^2 - 2t^3)(x_j - x_i) + rt(1-t)^2\delta_{ix} - st^2(1-t)\delta_{jx}$$
$$y(t) = y_i + (3t^2 - 2t^3)(y_j - y_i) + rt(1-t)^2\delta_{iy} - st^2(1-t)\delta_{jy}$$

for $0 \leq t \leq 1$ where (δ_{ix}, δ_{iy}) and (δ_{jx}, δ_{jy}) are the direction of the spline at points i and j respectively, and r and s are additional parameters that METAFONT calculates. METAFONT has a rule for determining the directions (δ_{ix}, δ_{iy}) at each point, but it is possible for the user to give them explicitly, and this is the approach that works best for Chinese characters.

All coordinates given in a METAFONT program are in absolute raster units so that rounding to the nearest integer corresponds to rounding to the nearest pixel. This allows a METAFONT program to make rounding adjustments to help fit the characters to the raster. When a cubic spline is rounded to the raster, the curves look much better if the points where the spline is vertical occur at integer s-coordinates.

Drawing with a circular pen or "cpen" produces a constant width line with rounded ends. METRFONT also has elliptical pens and special pens that can be an almost arbitrary shape, but the most general way to draw a shape is to use the "ddraw" command to specify both sides independently and have METRFONT fill in between them.

Complex mathematical constructions can be performed in METRFONT by taking advantage of subroutines and the ability to solve linear equations. We use such constructions to define the points and directions in various subroutines that draw strokes and other parts of characters. In this way, the subroutines can have a few parameters that control what they draw, and there can also be various global parameters that control overall properties of the font and allow for differences in point size and device resolution. **A** good illustration of this can be found in [2], where Knuth describes in detail the constructions and parameterizations used in his Computer Modern family of typefaces. See also [7].

METAFONT subroutines have two types of parameters. Index parameters are point numbers from the calling routine and may be used as point numbers in the routine that is being called. Ordinarily, point numbers have purely local significance, but in this way, it is possible to use points that are defined in the calling routine or to define points for use in the calling routines. It is even possible to define a point partly in one routine and partly in another, by giving additional constraints that allow METAFONT to solve for the coordinates. The second type of parameters are arbitrary numeric expressions that should be "known" at the time of the call. These may be used exactly as in any programming language.

Tung Yun Mei's LCCD system for designing Chinese characters [4] is based on METAFONT. It has draw statements, variables, and pens as METAFONT does, but it does not solve implicit equations, and subroutine parameters have a different meaning. LCCD has taken the ability of METRFONT to do affine transforms and incorporated it into subroutines. Each subroutine has transformation parameters that apply to everything it draws, and the transformations are composed when one subroutine calls another. METRFONT, on the other hand, applies a global transformation matrix to each point before actually plotting it.

LCCD makes it very convenient to apply affine transformations to subroutines, but since subroutines are limited to transformation parameters, it is difficult to parameterize subroutine results in any other way. Lack of conditional statements also makes complex constructions very difficult.

Another feature of LCCD is that it has another type of pens called "tear drops," which are intended for drawing dot strokes. However, it is difficult to draw high quality dot strokes of all styles with these.

2. Stroke Drawing Routines

The routines we have constructed are carefully parameterized with just enough information to describe the skeletons of the strokes and how they are joined to adjacent strokes. Mathematical constructions are used to adapt the stroke to the length, orientation, and shape determined by its parameters and the global font parameters. These constructions can get very complex, but since a small number of routines suffice for an entire family of fonts, the time spent writing and debugging them is quite small in comparison to the whole project, even when designing just 128 charactera.

The approach suggested in [4] is quite different. Tung suggests that the strokes should be drawn as affine transforms of canonical strokes. Despite its simplicity, this approach has a number of disadvantages. If the canonical stroke is rotated or stretched more than a small amount it acquires an undesirable shape. Examples in [3] show how this problem is solved by having many different versions of each stroke so that it is only necessary to transform them by small amounts. In [3] there are 108 different routines for drawing the strokes referred to here as horizontal strokes, vertical strokes, pie strokes, dot strokes, triangle strokes, f-strokes, and j-strokes. (See appendix 1.) In spite of this, the results obtainable are not as good as with the new method, where we have just a few routines that transform themselves properly.

2.1 The Pie Stroke

In Song style, the pie stroke is controlled by three point parameters. These are points 10, 11, and 12 in figure 1. The stroke goes from point 10 to point 12, and point 11 gives the initial and final directions: from point 10 it heads toward point 11 and it approaches point 12 from the direction of point 11. Notice that the stroke overlaps points 10 and 12 by an amount equal to half the stroke width. This helps the design transform properly when stroke width changes. The exact location of the curve is determined by the sharpness parameter. This is used to determine a point 7 on the skeleton and another point 8 giving the tangent there. The parameter gives the ratio between the distance from point 11 to point 8 and the distance from point .11 to point 12. Similarly, it also determines point 0 where the tangent line crosses the line between points 10 and 11. Point 7 is then located so that the distance between points 7 and 8 divided by the distance between points 0 and 7 is the same as the ratio of the 11-12 distance to the 10-11 distance. The purpose of the sharpness parameter is to control how close the stroke gets to point 11. The construction for point 7 tends to place it near point 11 except in extreme cases where this would not yield a smooth curve.

Since there are other tapering curved strokes in Song style, most of the pie stroke is drawn by a separate subroutine. This takes as parameters the three control points for the stroke, the width near point 10, a special taper parameter that determines how the width is changing, the slope of the line between points 1 and 2, and the size of the flares on each side of the top part of the stroke. The width of the narrow portion near point 2 is a global style parameter so it does not need to be passed as a parameter. The routine fits a quadratic equation to the width as a function of distance along the stroke to the specified widths and taper. The function is then used to find points on the edge of the stroke. The equation gives the distance between the pairs of points 1 and 2, 3 and 4, 5, and 6. The derivative of the width function determines vanishing points that give the spline directions at each pair of points. The vanishing points are too far away to show in the figure, but they are not hard to calculate. Suppose point 7 is at distance x from point 10. Then the distance between points 3 and 4 is w(x) and and the distance to the vanishing point is $-w(x)/w'(x)$. The direction of the curve at both points 3 and 4 is toward this vanishing point. There are similar vanishing points for points 1 and 2 and for points 5 and 6.

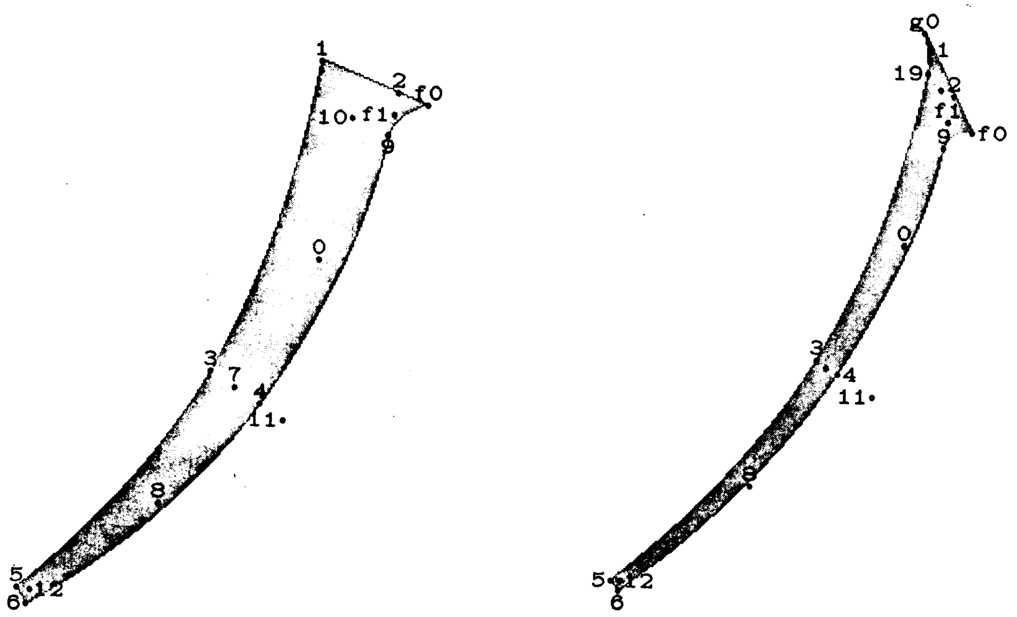


Figure **1.** The pie stroke construction in Song style and in Long Song.

The maximum width of a pie stroke is a linear function of its length, and the coefficients of this function are style parameters. The ***taper*** that is used for pie strokes is also a global style parameter. It is chosen to be somewhat less than 1 so that the rate of taper at the upper end of the stroke is less than that at the lower end.

The size of the flares at points 1 and f 0 are also given by style parameters. These flares are also drawn by a separate routine since similar flares also appear in other strokes. To draw the right side flare we pass points 1, 2, and 11 to the flare routine and let it draw the flare and return the point 9 where the flare stops.

2.2 The Dot Stroke

The dot stroke shown in figure 2 is halfway between the Song style and the Long Song. This should help to explain the effect of the ***dotrnd*** parameter that is used to interpolate between the two styles. This is a rather extreme example, because most style parameters are not so drastic in their effect or so complex in their implementation.

The basic construction is very similar to that used in [6], so we will not dwell on it here. Points 80 and 90 are parameters that control the position of the stroke. In ordinary Song style, points 10, 12, and 6 define the lower end of the stroke. Their placement relative to point 90 is fixed except for a scale factor used to control the overall width of the stroke. Lines 10-11-o and 6-2-8 are tangent to the stroke near the lower end; lines 3-8-7-5 and 4-O-9 are tangent to the stroke at the upper end. Point 8 is a fixed fraction of the way from point 7 to point 3 and point 0 is a fixed fraction of the way from point 11 to point 4. The distance between points 5 and 6 determines the curvature of the stroke and depends on the stroke length and the ***dotrnd*** parameter. Finally, the distance between points 7 and 9 is fixed in terms of the scale factor that was mentioned previously.

In Long Song style, the end of the stroke should be more triangular than in Song style. Points 106 and 112 are versions of points 6 and 12 that would be more appropriate for the Long Song. Point 112 is on the tangent from point 11 and point 106 is on the tangent from

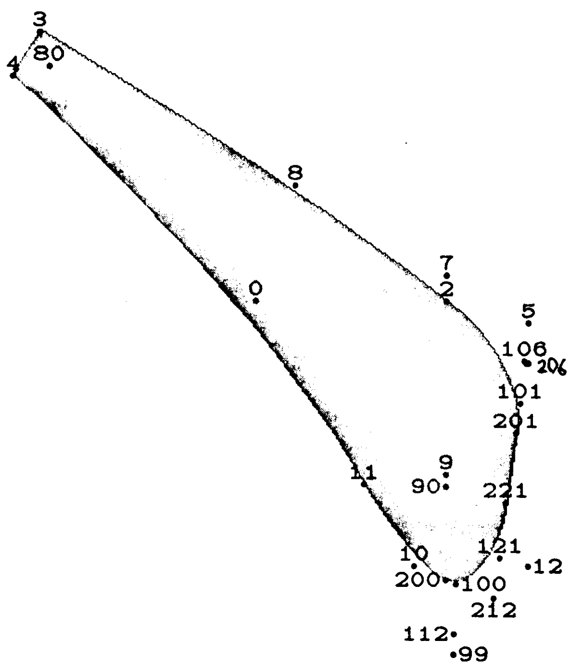


Figure 2 **A** dot stroke half way between the Song and Long Song styles

point 2. These points are placed as close to points 10 and 2 as possible without violating a certain minimum separation defined by the style parameter' ***dotcrv.*** Furthermore, we constrain the angle 11,112,106 to be 45". The primary effect of the ***dotrnd*** parameter is that we place point 212 this fraction of the way from point 12 to point 112 and similarly for point 206 between points 6 and 106 (not shown). The edge of the stroke is tangent to the line between points 206 and 212 in two places and to the line between points 10 and 212 in one place. In ordinary Song style, there would be only one point of tangency between points 206 and 212, but we split it so that there can be a large flat spot here in the Long Song style. The ***dotrnd*** parameter interpolates between two sets of placements for these points of tangency. Points 100, 101, and 121 are the placements for Long Song where ***dotrnd*** = 1; the placements for dotrnd = 0 are not shown but points 200, 201 and 221 are halfway between these and Long Song placements.

2.3 the Pie Stroke in Bold Style

In Bold style, the pie stroke is controlled by three parameter points exactly as it is in Song style. In figure 3, points 10, 11, and 12 have exactly the same meaning as they do in figure 1, except that the stroke must stop somewhat before point 12 in order to have the same apparent length. For the main body of the stroke essentially the same construction is used here as for the Song style except that the width function is very different. The width appears to be constant over the length of the stroke, but actually, there is significant variation. This is controlled by two style parameters that we shall refer to as δ and cr. The widths relative to a global stroke width parameter for curved strokes are $1 + \sigma + \delta/2$ near point **10,** $1 + \sigma - \delta/2$ near point 12; halfway in between, the width is equal to the parameter. As for the Song style, this determines a parabolic function that gives all the widths and locates the vanishing points.

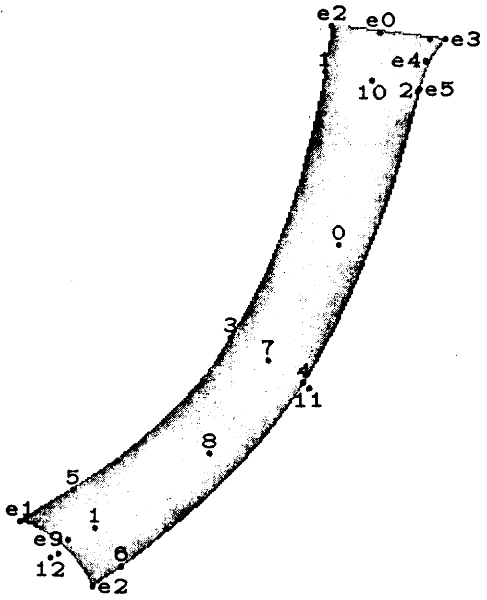


Figure 3. The pie stroke in Bold style

The upper and lower ends of the stroke are drawn by a separate subroutine that draws almost all the stroke ends in Bold style. The routine takes as parameters the stroke end parameter point, the associated vanishing point, the maximum width of the stroke, the widths of the flares on either side, and two more parameters that determine the concavity of the end of the stroke and the angle at which it is cut off. The cutoff angle near point **10** is a linear function of the angle that the **10-11** line forms with the x-axis and similarly the cutoff angle near point **12** depends on the angle of the **11-12** line. The coefficients of these linear functions are determined by style parameters.

3. Combining Strokes into Radicals

The stroke routines are designed to be as easy to combine as possible. In general, we join two strokes by placing their control points in some simple geometric relationship with each other, and by passing additional information to each of them indicating how it must adapt itself. In both Song style and Bold style, the routines have the same names and parameters but their actions are different. We will examine the problem of joining basic strokes together in Song style, since this is the more interesting of the two.

3.1 Positioning Strokes

Appendix **1** shows all the basic stroke routines along with the parameter points that control the position of each. With few exceptions each of the strokes has only enough parameters to determine its overall size and shape. This means that the radical routines only have to determine the placement of the strokes and the stroke routines handle all the other details and produce uniformly good looking strokes.

Certain stroke routines do have extra parameters to allow more generality in special cases and to simplify the process of joining basic strokes. In figure 9, the size and shape of the hooks near points **20** and **26** at the ends of e-strokes and I-strokes are determined by font parameters, so these stroke routines have no control points on the ends of the hooks. For j-strokes, the situation is similar except that the length of the hook often depends on how much room for it there is in the radical. We solve this problem by having point 29 partially determined by the stroke routine and partially determined by the radical. The radical routine sets $x_{26} - x_{29}$ to be a constant times a font parameter or a linear function of the width of the radical.

The curved strokes all use a special guide point parameter to give the initial and final directions of the curve as explained in section **2.1.** When the curve is to be symmetrical it is more natural to just give a single number specifying how much the stroke curves, but sometimes a highly asymmetrical curve is desired and this requires the extra degree of freedom that is provided by the guide point. We solve this problem by having a special subroutine to calculate the guide point for symmetrical curved strokes based on the endpoints of the curve and the amount of curvature desired.

The guide point for curved strokes is also used by routines that help join strokes together. Figure $4a$ shows how a vertical stroke might be joined to a pie stroke. The joining routine takes points **1, 2,** and 3 defining the pie stroke and points 4 and 5 defining the vertical stroke, and finds a new point 6 where the line defined by points 4 and 5 crosses the skeleton of the pie stroke. This computation is necessary to insure that the vertical stroke will always touch the pie stroke without crossing through it completely.

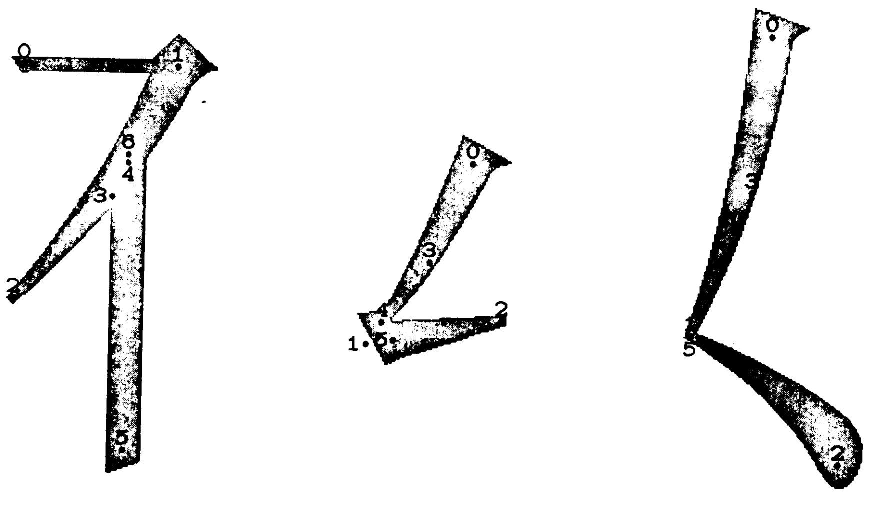


Figure 4. Special subroutines insure proper positioning when basic strokes are joined.

Figure $4b$ shows how a special routine is used to join pie strokes to triangle strokes. Here again, exact positioning is required to insure that the strokes meet without crossing through each other. The position of the join between strokes is controlled by point 1 where the extensions of their skeletons cross. The positioning routine takes points 1, 2, and 3 as arguments and sets points 4 and 5 so that they can be used as control points to the stroke routines. The routine also fills in a small area above point 1 in order to smooth out the corner where the strokes join and to provide an optical correction by thickening the end of the triangle stroke. The amount of this thickening depends on the width of the end of the pie stroke and this in turn is a font parameter that also controls the width of the thin portions of other strokes.

Figure *4c* shows another stroke combination where positioning is critical. The lower left corner of the pie stroke must exactly match the upper left corner of the dot stroke. **A** special routine takes points **1, 2,** and 3 and finds point 4 on the line between points **1** and 3, and point 5 on the line between points **1** and 2, so that the strokes will join properly if point 4 is used as the control point for the, pie stroke and point 5 is used as the control point for the dot stroke. In ordinary Song style, it turns out that points 4 and 5 are almost on top of point **1** so that there is no room to show point 4 in the figure. In other styles this is not the case, but the special routine guarantees that the strokes will always join properly.

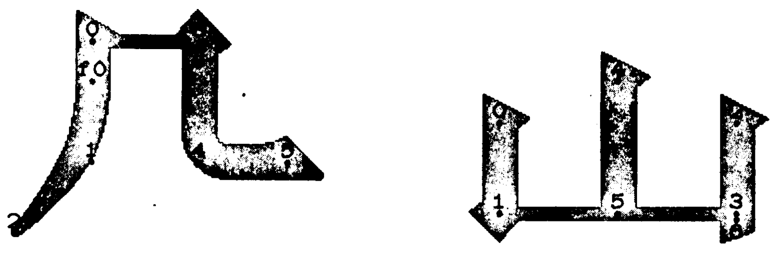
3.2 Endpoint Parameters

When strokes are joined together, the ends near the join have to change shape to adapt to the different possibilities. Fortunately, most of the basic strokes can be joined to other strokes only in a very limited number of ways. Most of the them have two special parameters that determine how each end is to be joined with surrounding strokes and thus what form it should take. These parameters range over a small set of integral values that we refer to symbolically as norm, join, corner, etc. For most strokes, not all of these values are used and some of those that are used in Song style become synonymous in Bold style and vice versa. In general, the parameters are norm for isolated strokes as shown in appendix **1.**

The radical in figure *5a* is constructed from three basic strokes and it illustrates two of the most common types of joins between strokes. The relative positioning of the strokes is very simple: point **0** is passed to the f-stroke routine as well as the horizontal stroke routine, and similarly, point 3 is also used by the I-stroke routine. The turning feature near point 3 where the horizontal stroke joins the Z-stroke is handled exactly the same way as similar features where horizontal strokes join vertical strokes, e-strokes, and j-strokes.

It is most convenient to draw the turning feature with a separate routine since it depends on the control points for both the horizontal stroke and the I-stroke. An added benefit is that this same routine can be used to draw the similar but slightly smaller feature found near point 1 in figure *5b.* End point parameters tell the strokes not to draw their usual ending features and cause the I-stroke to stop short of point 3 so that it cannot cross outside of the turning feature.

. The join between the f-stroke and the horizontal stroke is somewhat simpler, since the feature near point **0** in figure *5a* is part of the f-stroke. In fact, this f-stroke is almost identical to isolated f-strokes. **As** before, the situation would be equivalent if the f-stroke were replaced by any other basic stroke having similar structure on top. The complicating factor is that there is an optical correction to help balance the height of the feature near point 0 with that of the feature near point 3, and this requires the f-stroke routine to know the size of the feature. This size depends on the length of the horizontal stroke and whether or not it is joined to another stroke as in the figure, so it would be awkward to provide enough information to the f-stroke to enable it to calculate such a quantity. We solve this problem by always drawing the horizontal stroke first and saving the information



(*a*) (*b*)

Figure 5. Two simple radicals that illustrate the basic ways of joining horizontal and vertical strokes.

in a global variable. This is the only place where it is necessary to violate the usual stroke order, and in fact, it is the only place where stroke order matters **at all.**

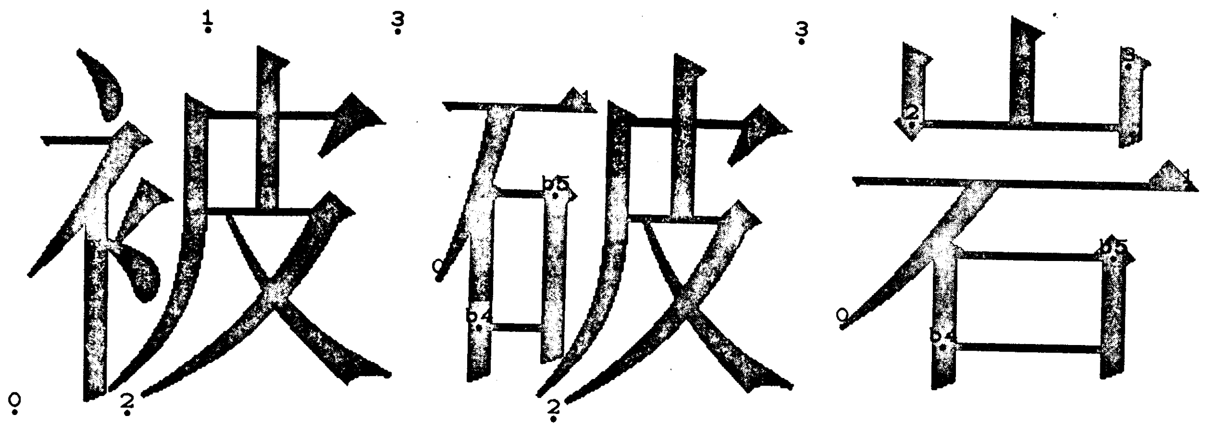
Figure *5b* illustrates the other main ways in which basic strokes are joined. The feature near point 1 is handled exactly the same way as the similar feature in figure $5a$, except that the lower end of the vertical stroke must be shortened and the feature is somewhat smaller.

When a vertical stroke is joined to the middle of an horizontal stroke or vice versa, the control point for the abutting stroke should lie on the skeleton line of the stroke being joined. This means that point 5 should lie on the line between points 1 and 3 and that point 3 should lie on the line between points 2 and 6. The special end point parameters are used to tell the routine for the joining stroke not to draw its usual ending feature. In the case of a vertical stroke joining a horizontal stroke as at point 5, the vertical stroke must have a squared off end flush with its control point. This is required because the vertical stroke can be much wider than the horizontal stroke that it joins and we have to guarantee that it will abut properly without crossing the horizontal stroke. The situation would be similar if the vertical stroke were to join the horizontal stroke from below, or even if the vertical stroke were replaced by a pie stroke.

It is also convenient to use end point parameters for variations not related to joining strokes together. For instance, the lower end of the I-stroke near point 5 in figure $5a$ is different from the more common version shown in appendix 1. The l-stroke routine has a parameter that tells it which form of lower end to draw.

3.3 Radical Routines

Since radicals can change size and shape when they are combined in different ways in different characters, it is necessary to design radical routines that are parameterized to allow this. The basic technique for doing this is to apply a simple geometric transformation to the control points of all the strokes. The size and shape of a radical is controlled by two points that are passed to the radical routine. Typically, these points will be two opposite corners of an imaginary box in which the radical lies. In figure $6a$, for instance, the left radical is controlled by points 0 and 1 and the right radical is controlled by points 2 and 3. In the radical routines, all coordinates are relative to the box defined by the control points. METAFONT has a convenient syntax for this: for instance, the x-coordinate of the vertical stroke in the left radical is refered to as $.42[x_0, x_1]$ which means $(1 - .42)x_0 + .42x_1$. (Actually, the specification is slightly more complicated than this because of the need for



(a) (b) (c)

Figure 6. Examples of how radicals change their shape when they are used in different characters.

rounding instructions.)

The right radical in figures 6*a* and 6*b* appears to be the same size and shape in both characters, but it is actually **10%** narrower in figure ***6b.*** Differences of this magnitude are very common in Chinese characters and they can easily by handled by simple geometric transformations as described above.

Sometimes additional corrections are required when the amount of stretching or shrinking in each coordinate is very large. Figures ***6b*** and 6*c* show how a radical undergoes such a change. The main radical routine is controlled by points **0** and **1** and it in turn calls a simpler radical that is controlled by points ***64*** and ***b5.*** Note that the control points are key points on the strokes in the radicals rather than the corners of surrounding boxes. This is basically an arbitrary choice, but it tends to facilitate joining additional strokes onto simple radicals to form more complicated ones. The choice of points 0 and **1** is convenient because it allows all the coordinates except y_{b4} to **be** specified by METRFONT expressions of the form $\alpha[x_0, x_1]$ or of the form $\beta[y0,$ **yl].**

The y-coordinate of point $b4$ depends on more than just y_0 and y_1. If we write y_{b4} as a METRFONT expression of the form $\gamma[y_0, y_1]$, we find that **7** $\approx -.29$ in figure ***6b*** and 7 $\approx -.13$ in figure 6*c*. Another way to look at the problem is that if we use y_{b4} and y_1 for the control box, then points 0 and $b5$ will be too high when a radical designed for figure ***6b*** is used in figure 6*c*. The solution we adopt is to make **7** a linear function of $(y_1 - y_0)/(x_1 - x_0)$. This is easy to do because of METAFONT's ability to solve implicit linear equations, and the result is a much more flexible radical routine.

4. Font Parameters and Different Styles

Our goal is to have one program that can create a whole family of Chinese fonts by just changing **a** few parameters. It is desirable to have these parameters relatively free in the sense that, within limits, the parameters can be set arbitrarily and still produce **a** reasonable font. This is especially difficult for Chinese because it takes several parameters to describe each basic stroke and it is not obvious what relationships have to hold in order for all the strokes to look reasonable and appear as if they belong to the same font. Here, we emphasize the need for sufficient variability while still trying to keep the number of parameters required to a minimum. We have enough parameters to allow us to obtain all three styles from [5], but further study is still necessary to determine exactly what degrees of freedom a Chinese "meta-font" should have.

4.1 Parameterization of the Font

Figure 7 shows how five different characters appear in ordinary Song style, Long Song, and Bold style. The basic positioning of the strokes is almost the same in all three styles, but the strokes themselves are very different. The Song and Long Song style characters in the first two lines all use the same stroke routines, but a completely different set of stroke routines was used for the Bold style characters in the last line. The variation between the ordinary Song style in the first line and the Long Song in the second line is entirely due to changes in the font parameters.

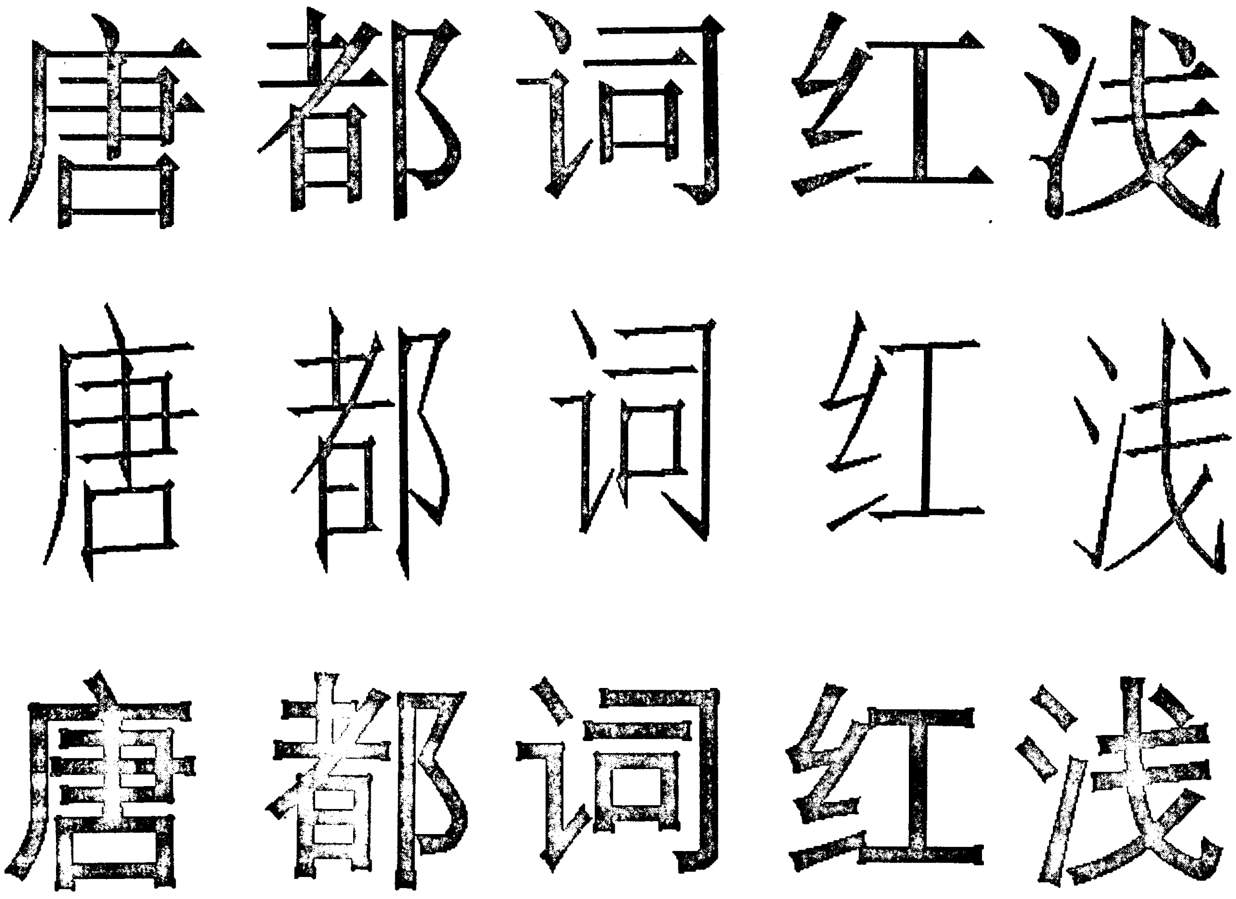


Figure 7. Five characters in Song style, Long Song style, and Bold style.

The character shapes are smooth functions of most of the font parameters, although there are two cases where conditional tests are used to produce changes in structure. Notice that the character in the second column of the figure has a horizontal stroke that joins a vertical stroke in the ordinary Song style but not in the Long Song. There is a special **gap** parameter that controls the degree of shortfall in such cases. For the ordinary Song style, this parameter is zero and horizontal strokes join in the usual way. When the end point parameter is set appropriately, the horizontal stroke routine tests this parameter and shortens the stroke by the distance ***gap*** and draws the usual ending feature.

The characters in the first three columns show the effects of another font parameter that must be treated specially. All three characters have four basic strokes that form a rectangle. We will refer to this combination of strokes as the square radical. In the Long Song style, the horizontal stroke extends beyond the vertical stroke in the lower right corner of this radical, but in the ordinary Song style, the vertical stroke extends beyond the horizontal stroke. This is handled partly by the stroke routines and partly by the radical routines and this is controlled by the font parameter ***overshoot.*** End point parameters are used to tell the vertical stroke and horizontal stroke routines that they are joining in this way. In this case, if the ***overshoot*** parameter is non-zero, the horizontal stroke **is** lengthened by a distance of ***overshoot*** and the vertical stroke is cut off at the bottom like the center one in figure $5b$. If the ***overshoot*** parameter is zero, then the horizontal stroke is cut off instead. Unfortunately, the radical routine also has to test the ***overshoot*** parameter because the lower control point for the vertical stroke should be on the horizontal stroke in order for them to join properly in the Long Song style. Very few different radical routines have to make this test, however, because most of them just call the square radical.

Other font parameters effect stroke widths, the sizes of various features and certain critical angles such as those that control the slopes of the ends of the strokes. Other prominent parameters are the overall height to width ratio of the characters and the slant parameter that makes the horizontal strokes slightly sloped in the Long Song style.

The Bold style stroke routines use a different, smaller set of font parameters. There are fewer special features to control, but stroke widths undergo subtle variations and there are other features such as the concavity of the ends of the strokes and the curvature of the dot strokes. Since we design the radical routines based on the Song style, stroke lengths have to be corrected slightly so that the strokes that are much thinner in the Song style will not appear too long in the Bold style. Figure 9 in appendix 1 shows how many of the Bold style stroke routines do not draw all the way to their ending control points. The magnitudes of these corrections are not true font parameters because they are calculated by the Bold style stroke routines based on the stroke width.

4.2 Adjusting Font Parameters

We have already seen how font parameters can be used to create different styles of characters approximating existing designs. Minor adjustments can made to change qualities like slant and boldness and to augment simple scaling to improve the appearance in small point sizes. It is also possible to experiment with new ideas and even go to ridiculous extremes.

In small point sizes it is desirable to keep stroke widths more uniform so that the thin strokes will not be too hard to see and the thick strokes will not encroach upon the white space too much. In Song style, there are three main parameters that control stroke width and a few more for the thick portions of strokes that vary in width. For the 18 point characters shown in appendix 2 and the large diagrams shown in the other figures, the basic widths of the vertical strokes, horizontal strokes, and the thin parts of tapered strokes are respectively 6%, 2.2%, and 1.8% of the type size. For the 10 point characters, however, we use **6%, 3.3%,** and 2.7%. This has the virtue that it tends to correct for the limitations of the printing device. The characters in the appendix were printed on a DOVER printer with a resolution of 384 dots to the inch. This means that for a 10 point font, all the characters are at most 53 pixels high. With the correction the horizontal strokes come out to be two pixels wide and the vertical strokes are three pixels wide. Without the correction the horizontal strokes would be only one pixel wide and they would hardly show up at **all.**

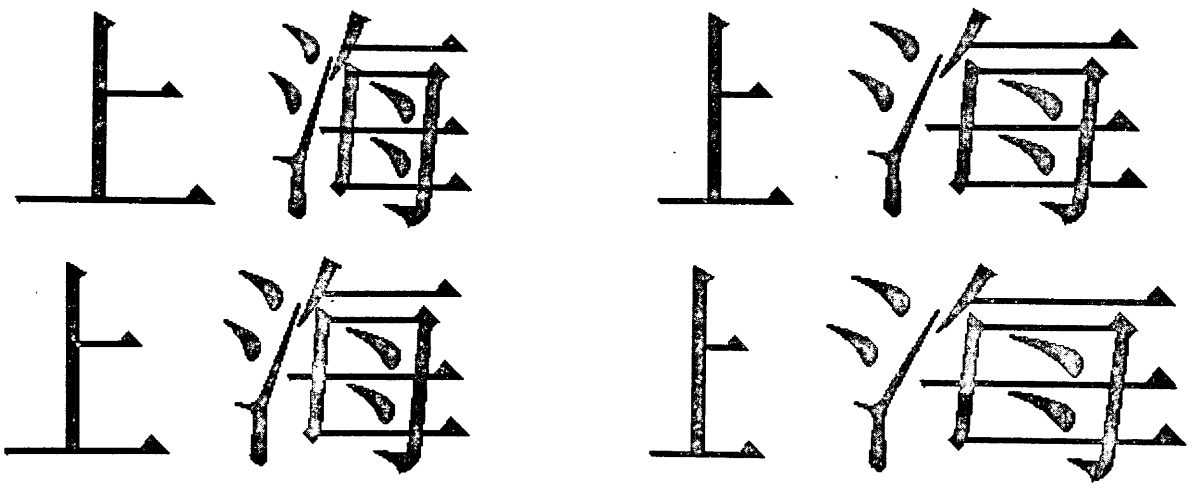


Figure 8. One possible way to experiment with a "meta-font." With each step the first character is compressed 18% and the second one is expanded by 22%.

Figure 8 shows one possible way to experiment with a "meta-font." We start with the characters the same width and progressively compress the simpler character while expanding the more complicated one. Note that this is not a simple geometric transformation, but a more complicated one as described in section 3.3 where the stroke widths are preserved and certain adjustments can take place. The stroke routines that cause the ending features on the.horizontal strokes to become slightly larger as the character is expanded. It is possible to carry such experiments to great extremes, but milder versions may be desirable in some applications. There are innumerable possible variations to be explored.

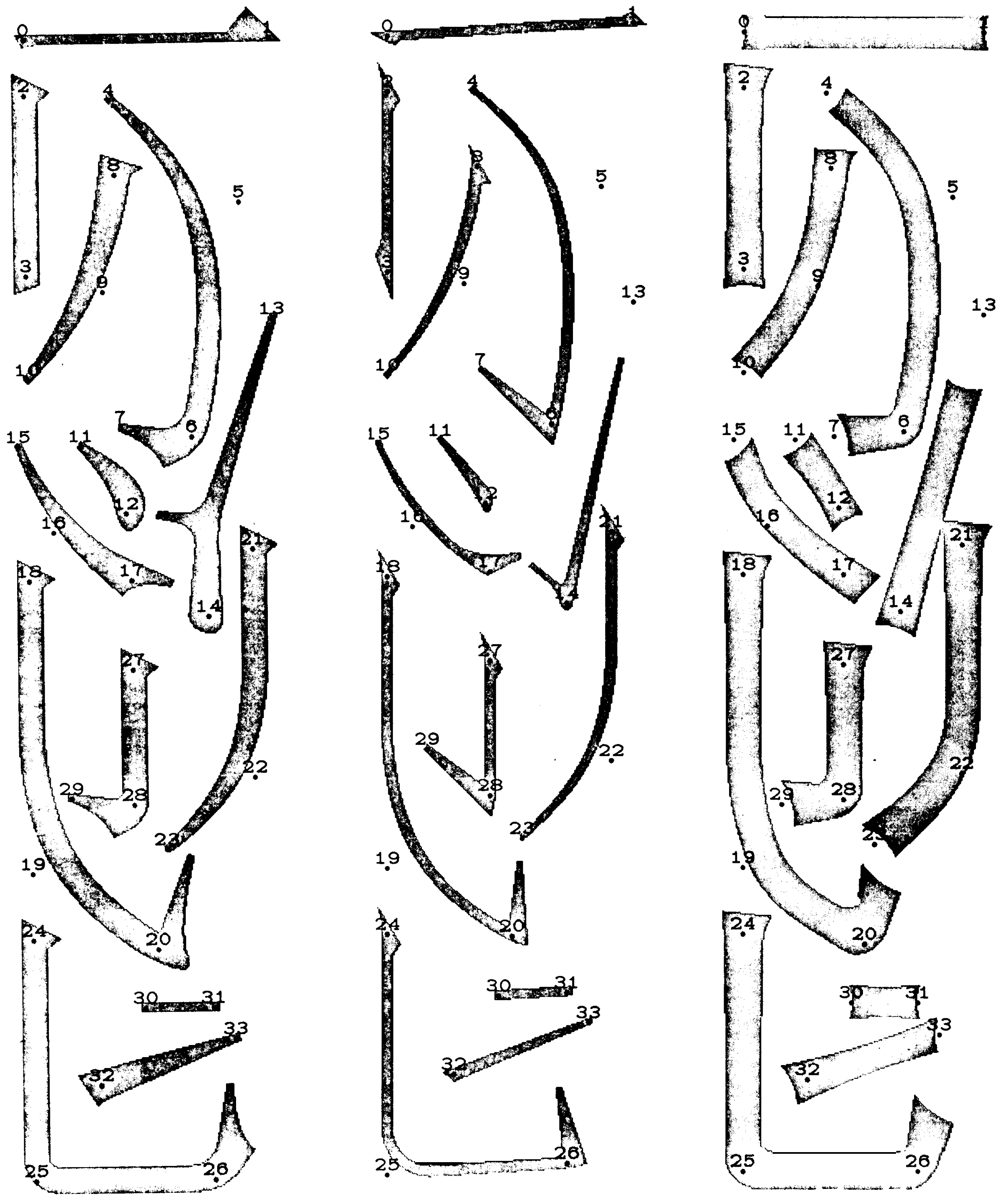
5. Conclusion

We have shown how it is possible to use METAFONT to design Chinese characters, and to obtain many different styles from the same program just by changing a few parameters. It is possible to build up a hierarchical structure so that most of the work is not too difficult and the resulting quality can be very high.

The authors are not expert font designers. Although we had access to high quality professional designs, they did not encompass the full range of characters discussed in this work and some judgment is required in order to best adapt them to the new medium. Details such as the exact relative positioning of the radicals could probably be improved. Our goal is to provide the groundwork for further research.

Appendix 1

For convenience, the basic strokes have been given somewhat arbitrary names with unique first letters. Figure 9 shows all the basic strokes and their control points for the ordinary Song, Long Song, and Bold styles.



(a) Song (b) Long Song (c) Bold
Figure 9. The basic strokes in all three styles.

The strokes on the left in the figure are in ordinary Song style and the Long Song and Bold style versions are shown in the middle and on the right. The correspondence between strokes and control points is as follows: horizontal stroke (0, 1), vertical stroke (2,3), u-stroke (4,5,6,7), pie stroke (8,9, 10), dot stroke (11,12), *k*-**stroke** (13,14), na stroke (15,16,17), e-stroke (18,19,20), f-stroke (21,22,23), Z-stroke (24,25,26), j-stroke (27,28,29), bar stroke (30,31), triangle stroke (32,33).

Appendix 2

Figure 10 shows how our fonts might be used in actual Chinese text.

忆 江 南

白居易

(唐朝 772-846)

江南好,风景旧曾谙:
日出江花红胜火,春来江水绿如蓝.
能不忆江南!.

Figure 10. An example of actual Chinese text. (唐宋词一百首 上海古籍出版社)

Figure 11 lists all 128 characters in Song style at 10 and 18 points and in Bold style at 18 points. Notice that many radicals appear in several different characters. Each radical is produced by one METAFONT subroutine, and all of the characters using a radical need only call the subroutine. This provides a substantial saving in labor and helps build uniformity into the font.

景 旧 曾 谙 日 出 花 红 胜 火 不 来 蓝 春 水 如
忆 绿 能 白 居 易 唐 朝 宋 司 词 百 首 上 海 古
籍 版 社 一 破 迄 高 纳 德 枕 蜀 都 珞 略 明 膘
意 慓 盆 旻 分 须 杉 幔 缦 镘 漫 熳 谩 慢 鳗 清
请 鲭 情 赗 蜻 锖 倩 残 浅 钱 乾 绍 谁 秒 利 稞
祼 被 衫 钐 裡 袼 褡 锘 褚 浬 理 锂 俚 狸 犹 拘
辅 鸟 纷 粉 粝 堆 唯 呻 吩 咯 鸣 惟 帷 淮 珅 绅
伸 砷 神 祜 汶 沈 捕 哺 埔 棵 枯 赐 江 南 好 风

景 旧 曾 谙 日 出 花 红 胜 火 不 来 蓝 春 水 如
忆 绿 能 白 居 易 唐 朝 宋 司 词 百 首 上 海 古
籍 版 社 一 破 迄 高 纳 德 枕 蜀 都 珞 略 明 膘
意 慓 盆 旻 分 须 杉 幔 缦 镘 漫 熳 谩 慢 鳗 清
请 鲭 情 赗 蜻 锖 倩 残 浅 钱 乾 绍 谁 秒 利 稞
祼 被 衫 钐 裡 袼 褡 锘 褚 浬 理 锂 俚 狸 犹 拘
辅 鸟 纷 粉 粝 堆 唯 呻 吩 咯 鸣 惟 帷 淮 珅 绅
伸 砷 神 祜 汶 沈 捕 哺 埔 棵 枯 赐 江 南 好 风

景 旧 曾 谙 日 出 花 红 胜 火 不 来 蓝 春 水 如 忆 绿 能 白 居 易 唐 朝 宋 司 词 百 首 上 海 古
籍 版 社 一 破 迄 高 纳 德 枕 蜀 都 珞 略 明 膘 意 慓 盆 旻 分 须 杉 幔 缦 镘 漫 熳 谩 慢 鳗 清
请 鲭 情 赗 蜻 锖 倩 残 浅 钱 乾 绍 谁 秒 利 稞 祼 被 衫 钐 裡 袼 褡 锘 褚 浬 理 锂 俚 狸 犹 拘
辅 鸟 纷 粉 粝 堆 唯 呻 吩 咯 鸣 惟 帷 淮 珅 绅 伸 砷 神 祜 汶 沈 捕 哺 埔 棵 枯 赐 江 南 好 风

Figure 11. Three fonts that were created from the same METAFONT program.

Appendix 3

Here, we show some of the actual METAFONT code. The dot stroke routine for Song style is an example of one of the most complex stroke routines. Once working, however, it is very easy to use and it provides an enormous degree of flexibility.

First we have some of the support routines from which the elaborate constructions in the stroke routines are built, then we have the stroke routine itself, and finally one of the radical routines that were used to create the last column of figure 7.

```
% Set slope = $(y_i - y_j)/(x_i - x_j)$
% and also find derivatives of arc length with respect to $x$ and $y$
subroutine fslope (index $i$, index j):
new slope; new dsdy; new dsdx;
if $x_i = x_j$ : slope = 7423.16;                                  % a large random number
     else: if $y_i = y_j$: slope = 117423.17;
          else: slope = $(y_i - y_j)/(x_i - x_j)$;
     fi;
fi;
dsdx = sqrt (1 + slope . slope);
dsdy = sqrt(1 + 1/(slope . slope)).

% Set dist to the distance between points $i$ and j, sqrdist to the square
% of the distance, and also set dx and dy to the x and y components.
subroutine fdist (index $i$, index j):
new dx, dy , dist, sqrdist ;
dx = $x_j - x_i$;
dy = $y_j - y_i$;
sqrdist = dx . dx + dy . dy ;
dist = sqrt sqrdist.

% Specify that point $k$ is distance $d$ to the right of the line from $i$ to j.
% Points $i$ and j should be known.
subroutine dtoright (var d, index $i$, index $j$)(index $k$):
call fdist ($i$,j);
$x_k \cdot (y_j - y_i) + y_k \cdot (x_i - x_j) = dist \cdot d + x_i \cdot (y_j - y_i) + y_i \cdot (x_i - x_j)$.

% Make a square end of width $d$ near $i$ for a stroke heading toward j. Facing from
% $i$ to j, $l$ is in the left and $r$ is on the right.
subroutine sqend (var d, index $i$, index j)(index $l$, index $r$):
no proofmode;
call toward (- (d - 1)/2, $i$, j, 0);
call right (j, 0, l);
call right (j, 0, r);
call dtoright ((d - 1)/2, j, $i$, $l$);
call dtoright ((d - 1)/2, $i$, j, $r$).

% Find point $k$, distance $d$ of the way from $i$ to j
subroutine toward(var d, index $i$, index $j$)(index k)
call fdist ($i$,j);
$x_k = x_i + (d/dist) \cdot dx$;
$y_k = y_i + (d/dist) \cdot dy$.
```

```
% Specify that point $k$ is on the line between points $i$ and j, which should be known,
subroutine online (index $i$, index j)( index k):
$x_k \cdot (y_j - y_i) - y_k \cdot (x_i - x_j) = x_i \cdot (y_j - y_i) - y_i \cdot \ldots - x_j)$.

% Find point $r$ at distance "dist" from $k$ so that the following lines from points
% $i$ and $r$ will have length ratio "rat." The lines will be tangents of a curve at
% $i$ and $r$. The tangent from $i$ passes through j and that from $r$ passes through $k$.
subroutine fspoint (index $i$, index j, index $k$, index $r$, var dist, var rat ):
no proofmode;
call intersect (k, i, j, 1);
new bsqr; $bsqr = (xi - x_1) \cdot (xi - x_1) + (y_i - y_1) \cdot (y_i - y_1)$;
new csqr; $csqr = (x_k - x_1) \cdot (x_k - x_1) + (y_k - y_1) \cdot (y_k - y_1)$;
call fslope $(i, j)$;
new tmpa; $tmpa = dist . rat$ + sqrt bsqr ;
new tmpb;
$tmpb = (\mathrm{sqrt}(tmpa . tmpa + (rut . rat - 1) \cdot (bsqr + csqr - dist . dist)) - tmpa)/(rat \cdot rat - 1)/dsdy$;
$y_0 - y_i = (x_0 - x_i) \cdot slope$ ;
if $y_j > y_i$: $y_0 - y_i = tmpb$;
      else: $y_i - y_0 = tmpb'$;
fi;
call fslope (0, k);
$y_r - y_k = ( x_r - x_k) \cdot dope$ ;
if $y_0 > y_k$: $y_r - y_k = dist / dsdy$;
      else: $y_k - y_r = dist / dsdy$ ;
A.

% This draws a dot stroke compromising between two styles and changes point n
% by rounding its x-coordinate.
subroutine dotstroke (index n, index t):
new tmpj;
 if $x_n < x_t$:
      call fdist (t, n);
      $y_5 - y_6 = dotrnd \cdot 0.13 . dist$ ;
      $y_{10} = y_{12} = \mathrm{round}(y_t - (.4 . cf . dotw - 1/2))$;
      $x_5 = x_6 = x_{12} = \mathrm{round}(x_t + (.4 \cdot cf . dotw - 1/2))$;
      $y_6 - y_{12} = cf \cdot dotw - .5$;
      $x_{12} - x_{10} = 0.56. cf \cdot dotw - .5$;
      $x_2 = x_7 = x_t$;
      $tmpf = .4$;
 else:
     . call roundx$(n, 1)$;
      $x_5 - x_{66} = 0.13 \cdot (y_n - y_t)$;
      $x_{10} = x_{12} = \mathrm{round}(x_t - (0.4 \cdot cf . dotw - 1/2))$;
      $y_5 = y_6 = y_{12} = \mathrm{round}(y_t - (0.4. cf . dotw - 1/2))$;
      $x_{66} - x_{12} = cf . dotw - .5$;
      call fslope (t, n);
      $x_6 = ((.6 slope + 2)/( slope + 2))[x_{12}, x_{66}]$;
      $y_{10} - y_{12} = 0.56 \cdot cf \cdot dotw - .5$;
      $y_2 = y_7 = y_t$;                          % at slope > 2, 6 moves 0.4 of the way to 12
      $tmpj = .54$;
 fi;
```

```
call sqend (w 1, n, 6, 3, 4);                                % 6 is almost the right direction
call online (3, 5, 7);
$x_8 = .37[x_7, x_3]$;
$y_8 = .37[y_7, y_3]$;                                         % 6 to 8 is tangent on top curve
call online (6, 8, 2);                                       % 2 is the tangent point
call toward((dotrnd[1, .57])($y_2 - y_{12}$) − 1, 7, t, 9);     % 9 is 4 tangent (divergence near n)
call fspoint (4, 9, 10, 11, .48 · cf . dotw − .5, .64);      % 11 is point of inflection

if $x_n < x_t$:
    $x_{111} = x_{11}$; $y_{111} = y_{11}$;                               % we are not going to move point 11
    call toward( cf . dotcru, 2, 6, 106);                     % 106 is new version of 6
    call toward(−cf (dotcrv + .19dotw), 10, 11, 112);         % 112 is new version of 12
    call sin (106, 112, 111);
    if acc > (sqrt .5): new $x_{112}$, $y_{112}$;                    % free either 106 or 112 to move out
        call online (10, 11,112);
    else: new $x_{106}$, $y_{106}$;
        call online (2, 6, 106);
    fi;
    $(x_{10} - x_{11} + y_{11} - y_{10})(x_{106} - x_{112})$
     $+ (x_{10} - x_{11} + y_{10} - y_{11})(y_{106} - y_{112}) = 0$;       % make 106 112 10 a 45 degree angle
    $x_{206}$ = dotrnd [$x_{106}, x_6$]; $y_{206}$ = dotrnd [$y_{106}, y_6$];     % 206 is compromise version of 6
    $x_{212}$ = dotrnd [$x_{112}, x_{12}$]; $y_{212}$ = dotrnd[$y_{112}, y_{12}$];  % 212 is compromise version of 12
    call toward( cf . dotcru, 212, 10, 100);                  % 100 is new version of 0
    $x_{120} = x_{100}$; $y_{120} = y_{100}$;                              % 120=100 since point 0 doesn't split
    call toward(cf . dotcrv, 212, 206, 121);
    call toward( cf . dotcru, 206, 212, 101);                 % 101 and 121 are new versions of 1
else:
    call toward (cf . dotcru, 10, 11,111);                    % 111 is new version of 11;
    call intersect (10, 6, 8, 50);
    call fdist (10, 50);
    call toward (−dist, 50, 8, 112);                          % 112 is new version of 12
    $x_{206} = x_6$; $y_{206} = y_6$;                                      % make 206 same as the original 6
    $x_{212}$ = dotrnd[$x_{112}, x_{12}$]; $y_{212}$ := dotrnd[$y_{112}, y_{12}$];  % 212 is compromise version of 12
    call toward (cf · dotcru, 10, 212, 100);
    call toward (cf . dotcru, 212, 10, 120);                  % 100 and 120 acre new versions of 0
    call toward (cf . dotcru, 212, 206, 101);                 % 101 is new version of 1
    $x_{121} = x_{101}$; $y_{121} = y_{101}$;                              % 121=101 since point 1 doesn't split
fi;

$x_{200}$ = dotrnd[$x_{100}$, (2/7)[$x_{10}, x_{212}$]];
$y_{200}$ = dotrnd [$y_{100}$, (2/7) [$y_{10}, y_{212}$]];
$x_{220}$ = dotrnd[$x_{120}$, (2/7)[$x_{10}, x_{212}$]];
$y_{220}$ = dotrnd[$y_{120}$, (2/7)[$y_{10}, y_{212}$]];                   % 200 and 220 are compromises for 0
$x_{201}$ = dotrnd[$x_{101}$, tmpf [$x_{206}, x_{212}$]];
$y_{201}$ = dotrnd[$y_{101}$, tmpf [$y_{206}, y_{212}$]];
$x_{221}$ = dotrnd [$x_{121}$, tmpf [$x_{206}, x_{212}$]];
$y_{221}$ = dotrnd[$y_{121}$, tmpf [$y_{206}, y_{212}$]];                  % 201 and 221 are compromises for 1

1 ddraw 3{$x_8 - x_3, y_8 - y_3$} . . 2{$x_6 - x_2, y_6 - y_2$} . . 201{$x_212 - x_201, y_{212} - y_{201}$}
    . . 221{$x_{221} - x_{206}, y_{221} - y_{206}$} . . 221,
    4{$x_9 - x_4, y_9 - y_4$} . . 111{$x_{10} - x_{111}, y_{10} - y_{111}$} . . 200{$x_{212} - x_{200}, y_{212} - y_{200}$}
    . . 220{$x_{220} - x_{10}, y_{220} - y_{10}$} . . 221{$x_{206} - x_{221}, y_{206} - y_{221}$}.
```

```
subroutine rzhe (index ll, index ur):                                   % 70% of x-side
$x_0 = (19/70)[x_{ll}, x_{ur}]$;                 $y_0 = 0.66[y_{ll}, y_{ur}]$;
$x_1 = (62/70)[x_{ll}, x_{ur}]$;                 $y_1 = 0.71[y_{ll}, y_{ur}]$;
$x_2 = (16/70)[x_{ll}, x_{ur}]$;                 $y_2 = 0.50[y_{ll}, y_{ur}]$;
$x_3 = (64/72)[x_{ll}, x_{ur}]$;                 $y_3 = 0.55[y_{ll}, y_{ur}]$;
$x_4 = good_3(32/70)[x_{ll}, x_{ur}]$;           $y_4 = 0.91[y_{ll}, y_{ur}]$;
$x_5 = good_3(33/70)[x_{ll}, x_{ur}]$;           $y_5 = 0.26[y_{ll}, y_{ur}]$;
$x_6 = (60/70)[x_{ll}, x_{ur}]$;                 $y_6 = 0.09[y_{ll}, y_{ur}]$;
$x_7 = (54/70)[x_{ll}, x_{ur}]$;                 $y_7 = 0.42[y_{ll}, y_{ur}]$;
$x_8 = (5/70)[x_{ll}, x_{ur}]$;                  $y_8 = 0.09[y_{ll}, y_{ur}]$;
$x_9 = (41/70)[x_{ll}, x_{ur}]$;                 $y_9 = 0.90[y_{ll}, y_{ur}]$;
$x_{10} = (52/70)[x_{ll}, x_{ur}]$;              $y_{10} = 0.78[y_{ll}, y_{ur}]$;
call roundy (0, 2);
call roundy (1, 2);
call roundy (2, 2);
call roundy (3, 2);
call 𝔥 hstroke (0, 1, norm, norm);
call 𝔥 hstroke (2, 3, norm, norm);
call 𝔢 estroke (4, 5, 6, norm);
call bendpie (7, 11, 8, 0.15);
call 𝔭 piestroke (7, 11, 8, 7, norm, norm);
call 𝔡 dotstroke (9, 10).
```

References

1. Knuth, Donald E., *TEX and METAFONT, New directions in typesetting,* Digital Press and the American Mathematical Society, 1979.
2. Knuth, Donald E., *The Computer Modern Family of Typefaces,* Stanford Computer Science Report STAN-CS-80-780 (January 1980).
3. Tung Yun Mei, *LCCD, A Language for Chinese Character Design,* Stanford Computer Science Report STAN-CS-80-824 (October 1980).
4. Tung Yun Mei, LCCD, A Language for Chinese Character Design, *Software Practice and Experience* **11** (December 1981), 1273-1292.
5. Unpublished Chinese character designs, *Shanghai Printing Technology Institute.*
6. Gu Guoan and Hobby, John D., Using METAFONT to Design Chinese Characters, *Journal of the Chinese-Language Computer Society,* to appear.
7. Knuth, Donald E., The Letter S, *The Mathematical Intelligencer* 2 (1980).