ELECTRONIC PUBLISHING, VOL. 6(2), 67–91 (JUNE 1993)

Structure extraction and automatic hinting of Chinese outline characters

SEUNG WOON PARK AND SEUNG RYOUL MAENG

Department of Computer Science
Korea Advanced Institute of Science and Technology
373-1 Kusong-dong Yusong-gu Taejon 305-701
Korea

SUMMARY

In spite of a worldwide trend towards the use of outline fonts for displays and for printing devices, they are still not very common in the east Asian countries where Chinese characters are used. The reason for this is that the complex, structured shapes of Chinese characters take a long time to design and develop.

Several systems have proposed automatic generation of outline fonts from the original master fonts. These systems have the serious problem of quality degradation when rasterizing the font at small point sizes, because they do not incorporate a hinting mechanism to adjust the outlines under these circumstances.

In this paper, we present an experimental study on a hinting mechanism specially designed for Chinese-style characters. We propose a scheme which automatically generates the hinted outline data from the plain outline fonts.

We have implemented and experimented with four sets of Korean Myungjo (Ming) and Gothic style fonts, and have obtained good results with respect to font quality and development time.

KEY WORDS Automatic hinting Font rasterization Outline font Chinese character

1 INTRODUCTION

For character processing on screen displays and on mid-resolution laser printers, the worldwide trend is towards representing digital fonts as outline information for the characters. These outlines are then processed to get on-the-fly bitmap fonts, in various sizes, at run-time.

In spite of much interest in outline fonts within those east Asian countries that use Chinese characters, it is still the case that outlines are not commonly used, largely because of the different characteristics of Chinese characters as compared to Latin ones. These different characteristics can be summarized as follows.

First, there is a difference in the design process. Latin characters are designed by drawing the outline with pen-type media and then filling the inside of the character. Chinese characters use a different process whereby the trajectories of a character are made with a weighted brush. In a Latin font, there are the mathematical regularities and symmetries. Most of the components are composed with strict lines and strict arcs in the outline description. In a Chinese font, however, there are few regularities because most of the trajectories made by the brush consist of random curves. The main components of a Chinese character are lines and curves (Figure 1).

CCC 0894–3982/93/020067–25

Received 31 December 1991
Revised 22 June 1992

EOzhp 大韓民國

Figure 1. Example of Latin and Chinese fonts

There is another difference, which relates to typographical control. Latin text requires proportional spacing of characters with differing widths. Therefore the inter-character balance is essential to Latin fonts and several reference lines such as baseline, x-height and capsline are needed for the control of this balance. In Chinese characters, however, the space between the characters is fixed and the balance of the character's constituent components becomes more important because a Chinese character has a large number of strokes (the average number of strokes in a character is 15[1]). The quality of Chinese fonts is mainly dependent on this inter-stroke balance.

The number of characters in a font is yet another difference. While a Latin font set consists of no more than 300 characters, a Chinese font set has several thousand characters. The standard encoding systems for Korean, Japanese and Chinese contain, respectively, about 8000, 7000 and 13 000 characters[2].

Because of their structural characteristics, it is far more difficult to digitize and rasterize Chinese fonts than Latin ones. During digitization, Chinese fonts require more storage and longer development times than do the Latin fonts. Furthermore, the rasterization of the digitized Chinese fonts takes more time because of the complex strokes. In the case of small-size Chinese characters, the quality of the reproduced fonts can deteriorate remarkably.

In east Asian countries, there have been several attempts to overcome these problems. A reduction in storage can be effected by using outline descriptions and the METAFONT technique [3–7]. Automatic curve fitting techniques make it possible to reduce development times by generating outline data automatically from analog master fonts [8–11]. Hardware solutions such as font-accelerating ASIC chips can shorten the time required for font generation at run-time [12–15].

The most serious issue in representing Chinese characters with outline fonts is the quality degradation for small-size characters. For Latin characters, several researchers have succeeded in improving font quality for the middle- and low-resolution devices [16–22]. Some of these techniques are commercially available, but many of them are proprietary [23–27].

Ou and Ohno[28] proposed a hinting mechanism for Chinese fonts which generates a larger bitmap than the desired size and then scales it down. Before making the enlarged bitmap, the vertical and horizontal segments are extracted and strokes of a predefined width are added to the final-size bitmap to regularize the stem widths. This can be regarded as a basic hinting mechanism but it requires a lot of time to generate a bitmap of the desired size. Moreover, the quality of the bitmap is not very good because the method only adjusts to equal width those strokes that are strictly horizontal or vertical strokes and the detection mechanism for these horizontal and vertical strokes is rather crude.

Despite the fact that a hinting mechanism is more necessary for Chinese fonts than for Latin ones, only a few systems have succeeded in obtaining limited improvements for Chinese characters, and none of these systems is commercially available. The first reason for this is the lack of a satisfactory hinting solution for the structural characteristics of

Chinese fonts when rendered on middle- and low-resolution devices. The second reason is the large amount of labor-intensive and tedious manual operations required for generating the hinting information from the outline fonts. Therefore, we need to devise an automated methodology for generating hinted data from plain outline fonts.

Two papers have already been published about the automatic generation of hinting information for Latin fonts. One of these is by Andler[29] and the other by Hersch[30]. The main idea of Andler's work is to detect stems, curve extrema and serifs in terms of Latin typography. The generated character outline hinting information is then used for Hersch's rasterization scheme. The horizontal and vertical strokes can be detected by grouping a pair of different winding directions among the horizontal and vertical segments. But this method may fail to detect the stems in Chinese fonts because a Chinese character consists of complex strokes, and it is hard to detect the near-horizontal and near-vertical strokes.

Hersch's approach is more systematic. First, it defines a style-independent template (model) for the outline description of each character,[1] and then matches each of the points in the given outline to the ones in the template. If the match is successful, the predefined hinting information in the templates is then copied into the given real character outline. For matching the character outline to the model, the coordinate space describing the approximate location of the model character's contours is divided into 5-by-6 sub-areas. Furthermore, model outline segments lying between characteristic points (local extrema and junctions) are marked by their direction. Such a model description allows the matching program to match characteristic points of the model to points of the input character. Despite some good results for Latin fonts, this method is still far from being suitable for Chinese fonts owing to the following two problems. Firstly, it is hard to make templates for each one of several thousand Chinese characters. A template cannot cover a set of similar typefaces because the structure of Chinese characters is too complicated to be represented by a template. Secondly, it is hard to match correctly the points in the outline descriptions of Chinese fonts because the sub-areas with 5-by-6 divisions cannot handle complex structures such as Chinese fonts. As a consequence, it is not suitable, for Chinese characters, to match the points by using templates as there are many gaps between the modeled outline description and the real outline description for each of the typefaces.

In this paper, we propose an efficient rasterization scheme including a hinting mechanism suitable for Chinese fonts. Based on this scheme, we present an automated methodology for generating hinted outline fonts, which is the main theme of this paper.

The rest of the paper is made up as follows. In Section 2, we propose an intelligent rasterization scheme for Chinese characters. In Section 3, we explain the automatic procedure for generating hinted outline fonts. In Section 4, we analyze and discuss the experimental results. Finally, in Section 5, the conclusions are discussed and some suggestions are made for further work.

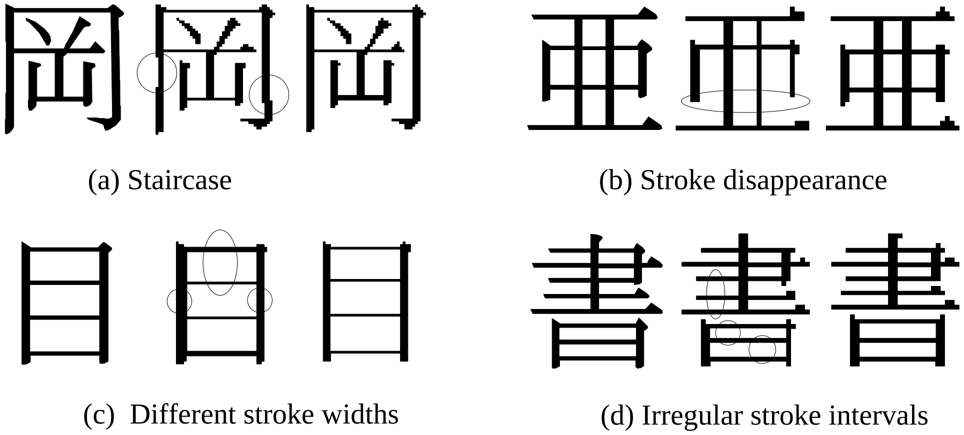
2 INTELLIGENT FONT RASTERIZATION SCHEME

2.1 Problems in primitive font rasterization

The plain outline font is defined by a set of closed paths which consists of line and curve segments. The primitive font rasterization algorithm simply scales the plain outline font at a given size, draws the outline and fills the interior area of the strokes.

[1] It can have several templates for one character.

Even if the primitive rasterization algorithm reproduces the character with complete mathematical precision, there are still difficulties in rasterizing small-size characters (i.e. smaller than 12 points at a resolution of 300 dpi). The algorithm regards the outline simply as a graphical description and does not take into account the innate characteristics of the font glyph. This leads to several anomalies and to a distortion of the original character shape, as shown in Figure 2. Within each part of that figure the characters on the left are the originals, the middle characters are drawn by the primitive font rasterizer and the ones on the right are drawn by an intelligent rasterizer in which the font characteristics are considered. In diagram (a), a staircase is generated whenever a stroke is nearly horizontal or vertical. Rounding errors have caused strokes of similar width within the character to look different. It is preferable to maintain uniform stroke widths, as in the right-hand example. Diagram (b) shows that strokes having a width smaller than one pixel may actually disappear. Since it is not desirable to exclude any stroke of a character, the width of a stroke must be at least 1 pixel. In diagram (c), strokes having similar or identical widths may, after rasterization, become unequal. Those strokes perceived as having equal widths in the original drawing must still have equal widths at small point sizes. In diagram (d), the stroke widths remain equal but the stroke intervals do not. We require that equal stroke intervals should remain equal after rasterization.



(a) Staircase

(b) Stroke disappearance

(c) Different stroke widths

(d) Irregular stroke intervals

Figure 2. Anomalies of primitive font rasterization

2.2 Proposed solutions

The above anomalies are mainly caused by rounding errors in converting real values into integer values, and also by the algorithm itself, which does not take into account the details of the character shape. We use the term *hint* to cover various stratagems for dealing with the problems mentioned in the previous section, with the aim of producing a character shape that is close to the original. An outline font which includes hinting information is called a *hinted outline font*, and a rasterizer which processes the hinting information in order to avoid or prevent the above problems is called an *intelligent rasterizer*.

Several hinting techniques for Latin fonts have already been introduced and most of these resolve into three possible methods[19]. One of these is to regularize the width of stems, i.e., vertical and horizontal strokes (Stem Hint), and another is to draw good-looking arcs by getting rid of flat arcs and any isolated pixels at the extreme point of an arc (Arc Hint). The third method involves adjustment of the various reference lines (Reference Line Hint). While the stem hint is applicable to Chinese fonts, the arc hint and the reference line hint are not suitable because Chinese fonts do not have strictly shaped arc strokes and reference lines. Chinese fonts require a more sophisticated hinting mechanism than the stem hint of Latin fonts if we are to solve the anomalies caused by the complex character shapes.

The hinting mechanism we focus on in this paper is stem regularization in order to solve the anomalies shown in (a) – (d). This mechanism maintains stroke widths and intervals close to those of the original shape by regularizing the horizontal and vertical strokes.

2.3 Stem regularization hinting

2.3.1 Basic concept

The main components of Chinese fonts are the horizontal and vertical strokes. About 37% of the components are horizontal strokes and 26% of them are vertical strokes. The average number of horizontal and vertical strokes in a character is 6 and 5 respectively[1]. In general, these horizontal and vertical strokes tend to have equal widths and intervals in a font.

Let us begin by supposing that X_1 and X_2 are two original coordinate values, and SF is the scaling factor. If the difference of these two values is less than half of $1/SF$ then the integer value of the difference of the scaled coordinate values will become zero, as follows.

$$abs(X_1 - X_2) < (1/SF) * 1/2, SF > 0$$

$$abs((X_1 - X_2) * SF) < 1/2$$

Therefore,

$$integer(abs(X_1 * SF - X_2 * SF)) = 0$$

Here, $integer(X)$ is the largest integer value less than or equal to $X + 0.5$ and $abs(X)$ is the absolute value of X.

Even if the integer difference becomes zero, the scaled integer coordinate values of X_1 and X_2 may still differ by one pixel, because of rounding error (and this error is sensitive to the choice of the coordinates).

However, if a point is calculated relative to other points we can eliminate the coordinate-sensitive round-off error, and the one-pixel difference can be avoided. A relative positioning of each point can also maintain equal strokes and intervals, as explained later.

It is important to select the proper reference points (in this paper, we call them *Base Points*) in order to control the coordinate-sensitive round-off error. Stem regularization hinting maintains equal stroke widths and intervals by selecting and controlling the Base Points properly and by using them to calculate other coordinate values in the form of a relative offset.

Figure 3 shows a simple example. Let us begin by focusing on the X coordinate values. If each point is calculated from the absolute values, then a difference of 1 pixel width can occur as a result of coordinate-sensitive round-off error, even though the real difference of

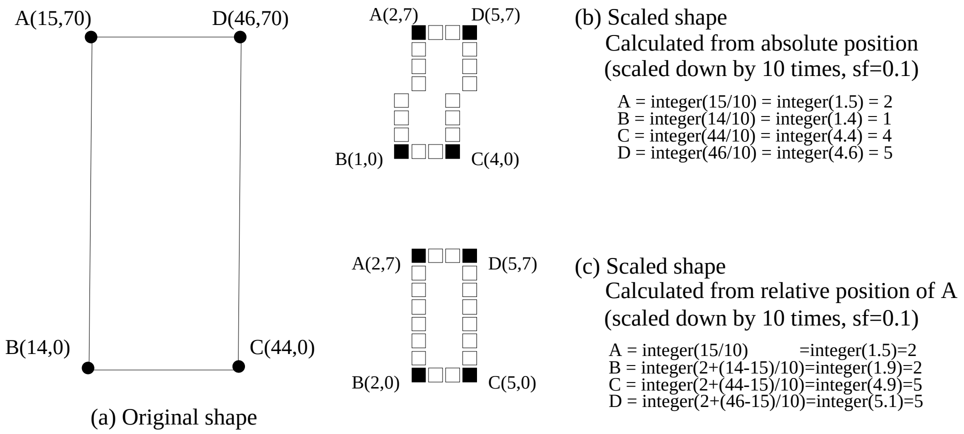

A(15,70)
D(46,70)
B(14,0)
C(44,0)
(a) Original shape
A(2,7)
D(5,7)
B(1,0)
C(4,0)
(b) Scaled shape
Calculated from absolute position
(scaled down by 10 times, sf=0.1)
A = integer(15/10) = integer(1.5) = 2
B = integer(14/10) = integer(1.4) = 1
C = integer(44/10) = integer(4.4) = 4
D = integer(46/10) = integer(4.6) = 5
A(2,7)
D(5,7)
B(2,0)
C(5,0)
(c) Scaled shape
Calculated from relative position of A
(scaled down by 10 times, sf=0.1)
A = integer(15/10) =integer(1.5)=2
B = integer(2+(14-15)/10)=integer(1.9)=2
C = integer(2+(44-15)/10)=integer(4.9)=5
D = integer(2+(46-15)/10)=integer(5.1)=5

Figure 3. Basic grid fitting mechanism

the two points is less than 0.5. To avoid this, we select a point such as A to be our Base Point, and calculate the coordinate values of the other points as relative offsets from point A.

2.3.2 Definition of points

To calculate the relative offset, all points in an outline font must be categorized by the following attributes for *X* and *Y* coordinates.

1. Primary Base Point (denoted PBP)
2. Secondary Base Point (denoted SBP)
3. Relative Point of PBP (denoted RelPBP)
4. Relative Point of SBP (denoted RelSBP)
5. Isolated Point (IP) (denoted IP)

PBPs are the basic reference points for calculating the other points, which are then worked out as follows (there must be one PBP per stroke).

If the attribute of i^{th} point is PBP, then

$$X'_i = ScaledCoord(X_i) = integer(X_i * XSF)$$

$$Y'_i = ScaledCoord(Y_i) = integer(Y_i * YSF)$$

In the above equations, XSF and YSF are the scale factors of the *X* and *Y* coordinates. X_n represents the original *X* coordinate value of the n^{th} point in the outline description. X'_n and $ScaledCoord(X_n)$ represent the scaled *X* coordinate values of the n^{th} point. Similarly, Y_n, Y'_n and $ScaledCoord(Y_n)$ are for *Y* coordinate values.

SBPs are the (optional) reference points for calculating stem widths and there must be zero or one SBP per stem. The coordinate value of an SBP is always calculated from an offset relative to some chosen PBP (which we shall call the ‘coupled’ PBP from now on).

The calculation goes as follows:

If the i^{th} point is chosen to be an SBP, then

$$\begin{aligned} X_i &= ScaledCoord(X_i) \\ &= ScaledCoord(PBP(X_i)) + integer((X_i - PBP(X_i)) * XSF) \end{aligned}$$

$$\begin{aligned} Y_i &= ScaledCoord(Y_i) \\ &= ScaledCoord(PBP(Y_i)) + integer((Y_i - PBP(Y_i)) * YSF) \end{aligned}$$

In the above equations $PBP(X_n)$ returns the X coordinate value of the coupled PBP corresponding to the SBP denoted as X_n. These are the basic equations for calculating scaled coordinate values for PBPs and SBPs but further details are given in the next section.

RelPBPs are those points which have not been chosen as SBPs but which are still calculated as an offset relative to some fixed PBP. For this reason the procedure for calculating the scaled coordinate value of a RelPBP is the same as above, but with the difference that multiple RelPBPs can exist for any chosen PBP.

If the i^{th} point is to be a RelPBP, then

$$\begin{aligned} X_i' &= ScaledCoord(X_i) \\ &= ScaledCoord(PBP(X_i)) + integer((X_i - PBP(X_i)) * XSF) \end{aligned}$$

$$\begin{aligned} Y_i' &= ScaledCoord(Y_i) \\ &= ScaledCoord(PBP(Y_i)) + integer((Y_i - PBP(Y_i)) * YSF) \end{aligned}$$

Continuing with our categorization of the various types of point we now come to RelSBPs which are those points calculated as an offset relative to some SBP. The procedure for calculating the scaled coordinate value of a RelSBP is as follows, and, again, multiple RelSBPs can be exist for any one SBP.

If the i^{th} point is to be a RelSBP, then

$$\begin{aligned} X_i' &= ScaledCoord(X_i) \\ &= ScaledCoord(SBP(X_i)) + integer((X_i - SBP(X_i)) * XSF) \end{aligned}$$

$$\begin{aligned} Y_i' &= ScaledCoord(Y_i) \\ &= ScaledCoord(SBP(Y_i)) + integer((Y_i - SBP(Y_i)) * YSF) \end{aligned}$$

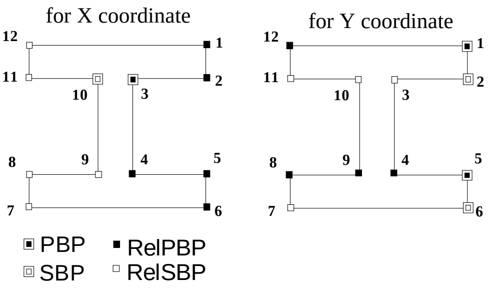
In the above $SBP(X_i)$ returns the X coordinate value of the SBP that is coupled to a particular RelSBP.

Finally, we come to points denoted as IP, i.e. independent points, which have relationship with any other point, and which are calculated as follows.

$$X'_i = ScaledCoord(X_i) = integer(X_i * XSF)$$

$$Y'_i = ScaledCoord(Y_i) = integer(Y_i * YSF)$$

All of the types of points that we have just described are gathered together in a basic data structure called the Base Point Table (BPT). This exists for each stem regularization of every character and it contains information about all the Base Points. In fact, there are two kinds of BPT for each of *X* and *Y* coordinates: the Primary BPT contains information for PBPs, while the Secondary BPT contains information for SBPs. Each row of the table corresponds to one point and each point is given a unique numeric identifier (see Figure 4).



BPT		
	X (PBP,SBP)	(3,10)
	Y (PBP,SBP)	(1,2),(5,6)

Point	X		Y	
Id	Attr	Index	Attr	Index
1	RelPBP	1	PBP	2
2	RelPBP	1	SBP	2
3	PBP	1	RelSBP	2
4	RelPBP	1	RelPBP	1
5	RelPBP	1	PBP	1
6	RelPBP	1	SBP	1
7	RelSBP	1	RelSBP	1
8	RelSBP	1	RelPBP	1
9	RelSBP	1	RelPBP	1
10	SBP	1	RelSBP	2
11	RelSBP	1	RelSBP	2
12	RelSBP	1	RelPBP	2

Figure 4. Examples of BPT and attribute of each point

2.3.3 Grid-fitting mechanism for base points

The scaled coordinate value for each type of point, calculated as shown in Section 2.3.2, can remove the staircase anomaly shown in Figure 2(a). However, the problems of stroke disappearance, unequal stroke width and irregular stroke intervals, shown in the remainder of that figure, still remain. These, in turn, can be removed by a method of repositioning the PBPs and SBPs, but a new data structure is required for this process.

The representative Stem Width Table (RSWT) is a table for calculating an SBP's offset value and it contains the representative width values of the vertical and horizontal strokes occurring in a font. There exist two RSWTs for the two kinds of strokes and the stroke width is calculated from the difference of the coordinate values of the SBP and its coupled PBP. We regard widths below a given threshold value as having the same value. For example, if the threshold value is 4 pixels and 8 horizontal stems exist in a font, and the stem widths are 22, 24, 23, 23, 21, 35, 37 and 37 pixels respectively, then the horizontal RSWT contains two elements of 23 and 37. Using this table an SBP is now calculated as follows.

Modified method for calculating an SBP

If the i^{th} point is chosen to be an SBP, then

```
$X'_i = ScaledCoord(X_i) = \{$
StemWidth $= X_i - PBP(X_i)$
    ;; Calculate the original vertical stem width containing $X_i$.
$ScaledStemWidth =$
    $integer(NearestValue(Vertical\ RSWT, StemWidth) * XSF)$
    ;; Calculate the scaled stem width by finding the nearest value in RSWT.
    ;; NearestValue(T,V) returns a value, T, from RSWT
    ;; which is nearest to the value V.
if $ScaledStemWidth = 0$, then $ScaledStemWidth = 1$
    ;; if the scaled stem width is zero, it is corrected to be 1.
returns $ScaledCoord(PBP\ (X_i)) + ScaledStemWidth$
    ;; SBP has the relative offset (ScaledStemWidth) of PBP
$\}$
```

The Y'_i values are calculated by the same method as the X'_is.

To avoid the total disappearance of the stroke, a stem width of 1 is returned whenever this method delivers a zero-pixel stem width. In this way the problem of stroke disappearance is eliminated. Furthermore, since all the stem widths are calculated from the RSWT, all the strokes within a given threshold value have the same stem width. This, in turn, means that the problem of differing stroke widths has also been solved (see Figure 5).

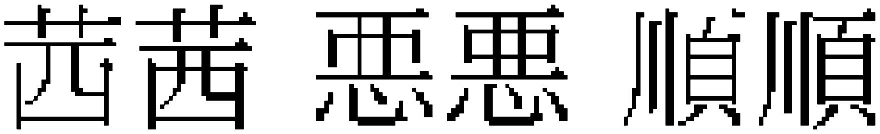


Figure 5. Examples of stem regularization

The final anomaly, that of irregular stroke intervals, can be removed by deliberately regularizing the intervals between strokes. To do this, each PBP has to be calculated relative to the *differences* between PBP coordinate values, rather than the absolute method shown in the previous section. The detailed procedure for relocating PBPs can be illustrated, for X coordinates, as follows.

Step 1. Suppose there are n PBPs. Sort them into ascending order of coordinate values.
Step 2. Calculate the scaled coordinate value of X_1, the first PBP.

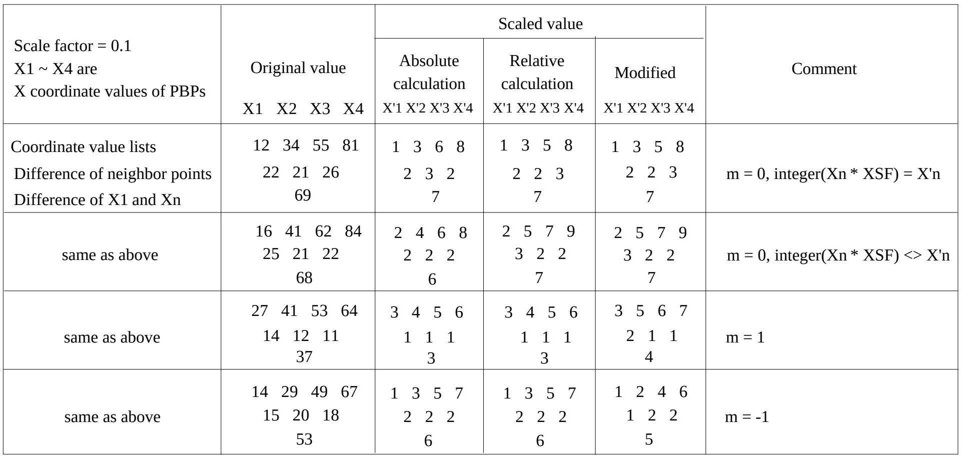
$$X'_1 = integer(X_1 * XSF)$$

Step 3. Calculate the scaled coordinate value of the final PBP, X_n as a relative offset from X_1.

$$X'_n = X'_1 + integer((X_n - X_1) * XSF)$$

Step 4. Calculate the relative differences and error terms between successive PBPs.
For each i from 1 to $n - 1$ perform

1. $DIFF_i = integer((X_{i+1} - X_i) * XSF)$



Scale factor = 0.1 X1 ~ X4 are X coordinate values of PBPs	Original value	Scaled value			Comment
		Absolute calculation	Relative calculation	Modified	
	X1 X2 X3 X4	X'1 X'2 X'3 X'4	X'1 X'2 X'3 X'4	X'1 X'2 X'3 X'4	
Coordinate value lists Difference of neighbor points Difference of X1 and Xn	12 34 55 81 22 21 26 69	1 3 6 8 2 3 2 7	1 3 5 8 2 2 3 7	1 3 5 8 2 2 3 7	m = 0, integer(Xn * XSF) = X'n
same as above	16 41 62 84 25 21 22 68	2 4 6 8 2 2 2 6	2 5 7 9 3 2 2 7	2 5 7 9 3 2 2 7	m = 0, integer(Xn * XSF) <> X'n
same as above	27 41 53 64 14 12 11 37	3 4 5 6 1 1 1 3	3 4 5 6 1 1 1 3	3 5 6 7 2 1 1 4	m = 1
same as above	14 29 49 67 15 20 18 53	1 3 5 7 2 2 2 6	1 3 5 7 2 2 2 6	1 2 4 6 1 2 2 5	m = -1

Figure 6. Examples of PBP relocation

2. $ERR_i = (X_{i+1} - X_i) * XSF - DIFF_i$

Step 5. Calculate the error term, *m*, to be compensated.

$$m = (X'_n - X'_1) - \sum_{i=1^n-1} DIFF_i = integer\Big(\sum_{i=1^n-1} ERR_i\Big)$$

where $-n/2 < m \leq n/2$
If *m* is non-zero, distribute the error term as follows.

if $m < 0$, select the *m* PBPs with the lowest ERR_i values,
and decrease the corresponding $DIFF_i$ values by one.
if $m > 0$, select the *m* PBPs with the largest ERR_i values,
and increase the corresponding $DIFF_i$ values by one.

Step 6. For each PBP, calculate the scaled coordinate value by accumulating the relative offset values as follows.

for *i* = 2 to $n - 1$

$$X'_i = X'_{i-1} + DIFF_i$$

The *Y* coordinates for the PBPs are calculated by the same method.

Another new data structure called the OLPBP (Ordered List of Primary Base Points) helps in implementing the above methods. There exist two OLPBPs, for *X* coordinates and *Y* coordinates respectively Each of these lists is sorted by coordinate value and the existence of the two lists allows us to skip step 1 in the algorithm given above. Several examples for relocating PBPs are shown in Figure 6.

We are now in a position to summarize the entire point relocation procedure as follows:

Stage 1. Relocate the PBPs using OLPBP.
This regularizes the intervals between strokes.

Stage 2. Relocate the SBPs using the RSWT.
This regularizes the stroke widths It also eliminates stroke disappearance

Stage 3. Calculate other types of points, such as RelPBPs and RelSBPs as relative offsets using BPT.
This last procedure eliminates abnormal 'staircase' effects

Each point on a character outline is recalculated as a grid-fitted point, in a postprocessing phase, using the methods we have described. The outline is then drawn with the grid-fitted points and the interior pixels are filled during the scan conversion phase. Since the stem regularization hinting is performed in a postprocessing phase, there are several efficient algorithms which can be adopted in earlier phases. In this paper, for example, we use the adaptive forward-difference algorithm of [21] for the Bézier curve drawing, together with an enhanced inside-fill algorithm which is based on the edge-flag algorithm of [31].

The hinting information explained previously is essential to our intelligent rasterizer, but it is hard to generate the data manually, and takes too much time, especially for Chinese fonts. In the next section we show how to automate the generation of hinting information.

3 AUTOMATIC GENERATION OF A HINTED OUTLINE FONT

3.1 Introduction

To construct hinted outline fonts with our intelligent rasterizer, the horizontal and vertical strokes must be extracted from the plain outline fonts. Chinese characters are generally composed with complex forms intersecting each other but, because the plain outline fonts do not contain any information about stroke structure, it is hard to detect near-horizontal and near-vertical components in these characters.

We shall now propose a scheme for detecting horizontal and vertical strokes by decomposing the outline font into several basic strokes. Once the horizontal and vertical strokes have been correctly identified the required information for our stem regularization hinting can be generated automatically.

The source material for our scheme is a set of plain outline fonts as explained in Section 2. The plain outline fonts must satisfy the following conditions.

1. There must be no intersection points within a single path, i.e., it must be a single contour.
2. Paths must not intersect each other.
3. The right part of the winding direction of a path must be the inside of the character.

The basic stroke extracted from a plain outline character can be regarded as the trajectory made by a single movement of a brush.

3.2 Automatic decomposition of a plain outline font

We shall use the term *visible segments* for all line segments and curve segments that are part of an outline. We assume also that there are *hidden segments* which are inside an outline. A hidden segment is generated by connecting the end points of two visible segments and there exist many such hidden segments in an outline. The particular hidden segment which is used to decompose an outline is called the *cut segment*.

The process of outline decomposition consists of finding the cut segments among the various candidate hidden segments and then generating the connected components from a subset of visible segments and cut segments. The scheme for decomposing a plain outline font into its basic strokes is organized as two procedures. A global decomposition procedure is carried out first, followed by a local decomposition procedure.

The global decomposition procedure detects the strictly decomposable strokes and proceeds to decompose them. A strictly decomposable stroke has at least one hidden segment but there will also be some visible segments connected to it which can be regarded as straight lines within some given threshold value for stroke linearity. Most of the strokes in a given outline are decomposed into basic strokes using this procedure; the intersecting and branching strokes of Figure 7 are examples of strictly decomposable strokes. Any strokes which are not decomposed in this procedure are called *compound strokes* and these can be analyzed using the local decomposition procedure.

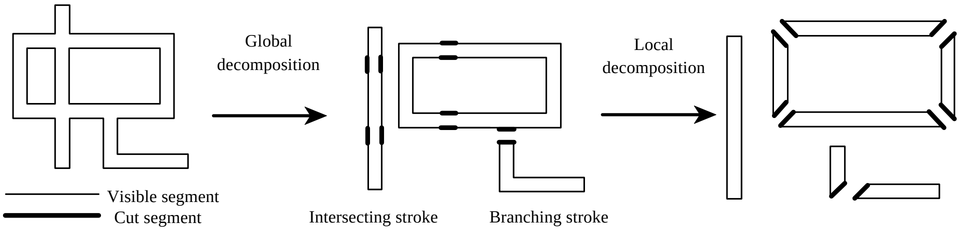


Figure 7. Examples of global and local decomposition

The idea of the local decomposition procedure is to detect those cut segments which are not part of a strictly decomposable stroke but which are nevertheless needed for decomposing a compound stroke into a basic stroke. The reason for separating the procedure into these two steps is that the approach for selecting the cut segment is different in the two cases. Figure 7 shows an example of global and local decomposition.

3.2.1 Global decomposition

The global decomposition procedure consists of five steps. Up to step 3, it tries to detect points of intersection and branching points. In the later steps the outline is decomposed into smaller outlines, each of which is a basic stroke or a compound stroke.

Step 1. Find the serif segments

A *serif*, as explained here, means the component located as the final part of a Chinese stroke with the following characteristics.

1. It generally consists of curve segments and the sum of the segment lengths does not exceed a given threshold value.[2]

[2] This threshold value is different for each typeface. For example, it is 80 units for the Myungjo typeface in a 512×512 mesh size.

2. In the case of curve segments, the angle of the tangent vector at both end points of a curve segment is less than 90°.
3. In the case of line segments, any two successive segments make an angle of less than 90°.

The segments satisfying the above conditions are joined as a path. An example of a serif is shown in Figure 8.

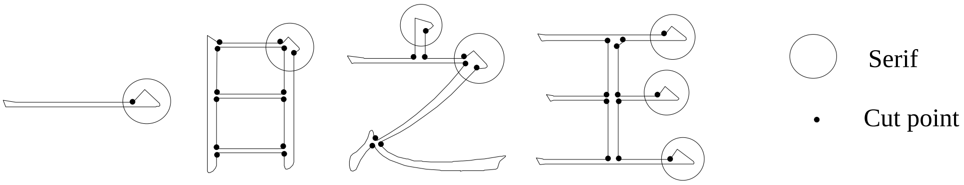


Figure 8. Examples of serif and cut point

Step 2. Find the cut points

A *cut point* as defined here is the joining point of two connected basic strokes or two intersecting basic strokes in a character (Figure 8). The candidates for cut points are any two successive segments for which the angle of intersection is greater than 180°. All points satisfying the above conditions are joined into a path.

Step 3. Select two cut points

In this step we identify a pair of cut points which meet all of the conditions show below. This pair of points will be used later for decomposing a character into its constituent strokes. (We assume that at least two such cut points will exist for each character, resulting from two connected strokes or two intersecting strokes.)

Let us define the two desired cut points as CP_i and CP_j. CP_i is called as *Start Cut Point* (SCP) and CP_j as *End Cut Point* (ECP). The pair of SCP and ECP is expressed as (SCP, ECP) and defines the *cut segment*.

Condition 1. CP_i and CP_j must not be points on the same segment.
Condition 2. CP_i and CP_j must not be points on the same serif stroke.
Condition 3. The distance between the CP_i and CP_j must be less than some given threshold value which is typeface-dependent.[3]
Condition 4. The tangent vectors of the two segments containing CP_i and CP_j must be parallel to one another. The angle between any two such tangent vectors should be less than some given threshold value.[4]
Condition 5. The line $\overline{CP_iCP_j}$ must not contain any other segment point on the contour of the character.

[3] 80 units for Myungjo, 120 units for Bold Myungjo, 90 units for Gothic, 130 units for Bold Gothic.
[4] 20° for the above typefaces.

Condition 6. The line $\overline{CP_iCP_j}$ must lie totally inside the boundary of the character. [5]

Condition 7. Suppose there is a second candidate, (CP_k,CP_j), for being the cut segment, where CP_j is already marked as ECP and the distance of $\overline{CP_iCP_j}$ is less than the distance of $\overline{CP_kCP_j}$ and the tangent vector angle of (CP_i,CP_j) is less than the angle of (CP_k,CP_j). Under these conditions (CP_i,CP_j) would be chosen as the cut segment.

Condition 8. If CP_i is already marked as SCP, then any candidate cut segment, (CP_i,CP_j), is to be compared with the previously chosen 'best' cut segment using the criteria laid down in condition 7.

The two cut points which satisfy the above conditions are entered into the cut segment list. The nature of each condition is illustrated in Figure 9.

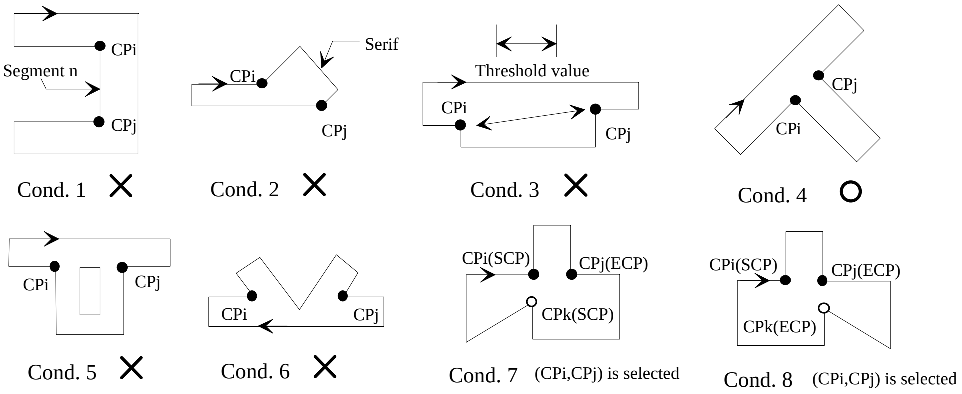


Figure 9. Example of conditions for identifying a cut segment during global decomposition. The symbol O *denotes a satisfied condition and* X *an unsatisfied condition*

Step 4. Duplicate the cut segment

If two cut points, CP_i and CP_j, appear only once in the cut segment list, then create a new cut segment whose SCP and ECP are CP_j, CP_i respectively. Cut points on intersecting strokes will already have been registered twice using the algorithm in step 3, so these must not be duplicated. Only the cut points on the branching strokes need to be duplicated.

Step 5. Decompose outline

In this step we decompose the plain outline character using the chosen cut segments. The decomposed outlines will also be plain outline characters. The overall procedure is as follows.

```
unmark all Segments.
while (each Segment is marked) {
  Choose an unmarked segment as Start Segment.
  Set Start Segment to Current Segment and mark it.
```

[5] This can be verified by scan converting a sufficiently big outline.

```
  do {
    if the last point of the Current Segment is the SCP of cut segment, CS, which is unmarked.
      Set cut segment CS as Current Segment.
    else
      Set next segment of Current Segment as Current Segment.
    Mark Current Segment.
  } while (Current Segment is not Start Segment)
  Note down the newly marked segments as new outline
}
```

In this procedure, the newly acquired outlines are either basic strokes or compound strokes (which consist of several basic strokes). The compound strokes can then be decomposed more precisely in the local decomposition step as described in the next subsection.

3.2.2 Local decomposition

The steps of the local decomposition procedure are similar to those for global decomposition. The procedure for decomposing compound strokes into basic strokes is to select *one* candidate for a cut point and to identify a hidden segment which includes this cut point and satisfies the conditions given below. (In global decomposition, *both* end points of the selected hidden segment are always cut points.)

To separate the connected strokes, repeat the following steps for each outline that has already been identified by the global decomposition procedure.

Step 1. Find cut point

Recalculate the cut points using the procedure described in step 2 of the global decomposition method.

Step 2. Select cut segment

This step is similar to step 3 of the global decomposition procedure, but the conditions are different. As only one cut point (CP_i) exists on the connected strokes for making a cut segment, it is necessary to select some other candidate point, P_j, arbitrarily from among the other points on the outline.

Condition 1. CP_i and P_j must not be points on the same segment.

Condition 2. The absolute value of the angle subtended by two segments including P_j should be greater than some given threshold value,[6] i.e., two segments must not form a single straight line.

Condition 3. The distance between CP_i and P_j must be less than a given threshold value.[7]

For every CP_i in each of the decomposed outlines, step 2 finds all points P_j which satisfy the above conditions. The pair (CP_i, P_j) whose length is the shortest is entered into the cut segment list as (SCP, ECP). If there are no more candidate pairs to be considered, the decomposition process for that outline is complete.

Each of the conditions set out above is illustrated in Figure 10.

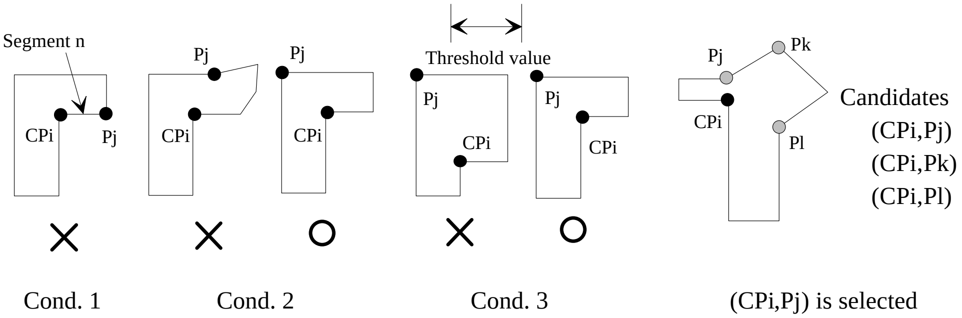


Figure 10. Example of conditions for cut segments in local decomposition

Step 3. Duplicate the cut segment

If the two points of a chosen cut segment are CP_i and CP_j, then enter an additional cut segment in the cut segment list whose SCP and ECP are CP_j and CP_i respectively.

Step 4. Decompose the outline

An outline is decomposed into two basic strokes using the same procedure employed in step 5 of the global decomposition. If the total length of the decomposed stroke is less than a given threshold value, that outline is not decomposed any further. Each decomposed stroke is regarded as a new outline and steps 1 to 4 of this section are repeated.

Figure 11 shows the decomposed basic strokes of a sample Chinese character. In this example, the sample outline is decomposed into 33 basic strokes.

3.3 Detection of vertical and horizontal strokes

The main operation for stem regularization hinting involves recognizing horizontal and vertical strokes (and those that are nearly horizontal or vertical) and locating a PBP and SBP for them (see Section 2.3). The horizontal and the vertical strokes are identified from

[6] 10° for Myungjo, Bold Myungjo, Gothic and Bold Gothic.

[7] 60 units for Myungjo, 120 units for Bold Myungjo, 90 units for Gothic, 130 units for Bold Gothic.

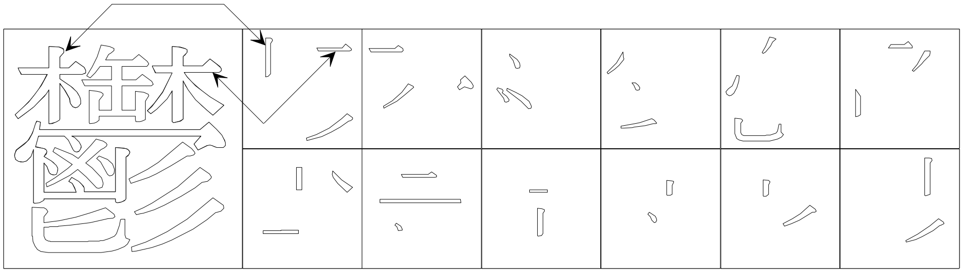


Figure 11. Example of outline decomposition

the decomposed basic strokes using methods described below. Since stem regularization involves the equalization of certain stroke widths it follows that we need at least two strokes, of a given type, before we can apply the method. We begin with the conditions for recognizing vertical strokes.

Condition 1. There character must have at least two segments each of which is parallel to the *y* axis and which lie in the opposite direction to each other as the character is traversed.

Condition 2. The horizontal distance[8] between the segments in a pair is less than a maximum threshold value and greater than some minimum threshold value.[9]

If a basic stroke is identified as a vertical stroke, a pair of points which has the smallest horizontal displacement from each other is selected as the PBP and SBP. For convenience we select the point with the larger *X* coordinate value as the PBP.

The method for identifying a horizontal stroke is similar to the above:

Condition 1. There must be at least two segments each of which is parallel to the *x* axis and which lie in opposite directions to each other as the character is traversed.

Condition 2. The vertical distance between the two segments in a pair must be less than a maximum threshold value and greater than some minimum threshold value.[10]

If a basic stroke is identified as a horizontal stroke, a pair of points which has the shortest vertical displacement from each other is selected as the PBP and SBP. For convenience, we select the point which has the larger *Y* coordinate value as the PBP.

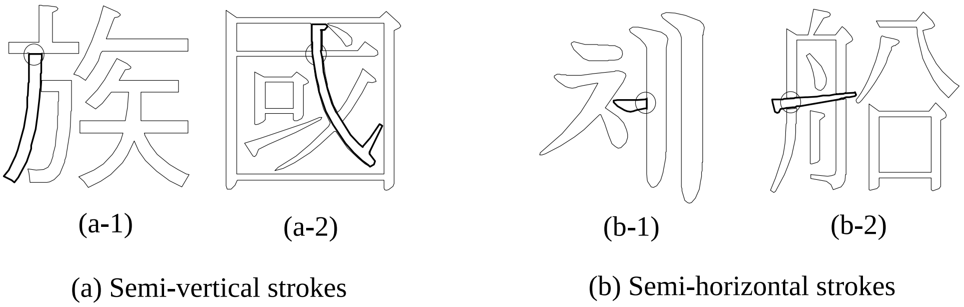
In addition to the strict vertical and horizontal strokes there are other strokes which feature in the stem regularization hinting as shown in Figure 12. These slightly slanted strokes are called *semi-vertical* and *semi-horizontal* respectively. They are classified as follows.

(a) Semi-vertical stroke

[8] The horizontal distance is the difference of *x* coordinate values.

[9] Maximum threshold values are 60 units for Myungjo, 120 units for Bold Myungjo, 80 units for Gothic and 120 units for Bold Gothic. The minimum threshold value is 20 units for each typeface.

[10] Maximum threshold values are 50 units for Myungjo, 80 units for Bold Myungjo, Gothic and Bold Gothic. Minimum threshold values are 10 units for Myungjo and Bold Myungjo, 20 units for Gothic and Bold Gothic.



(a) Semi-vertical strokes (b) Semi-horizontal strokes

Figure 12. Examples of semi-vertical strokes and semi-horizontal strokes

1. The stroke branching from a strict horizontal stroke (a-1 in Figure 12)
2. The stroke intersecting with a strict horizontal stroke (a-2 in Figure 12)

(b) Semi-horizontal stroke

1. The stroke branching from a strict vertical stroke (b-1 in Figure 12)
2. The stroke intersecting with the strict vertical stroke (b-2 in Figure 12)

The PBP and SBP for these strokes are the points which branch from, or intersect with, a strict horizontal or vertical stroke (see Figure 12).

3.4 Attribute assignment to points

Once the horizontal and vertical strokes have been identified, the decomposed strokes need to be classified and an attribute assigned to each point. The *X* and *Y* coordinates of each point are tagged with an attribute to denote whether it is a PBP, SBP etc. (see Section 2.3). The attribute assignment for the *X* coordinate of each point is performed as follows.

Case 1. Vertical stroke containing PBP and SBP.

```
For each X coordinate on this stroke, an attribute of RelPBP or RelSBP is assigned.
if the X coordinate value of a point is close to the PBP X value,
    the point is assigned an attribute of RelPBP in the X-coordinate part of the Primary
    Base Point table.
if the X-value of a point is close to that of an SBP,
    it is assigned an attribute of RelSBP in the table.
```

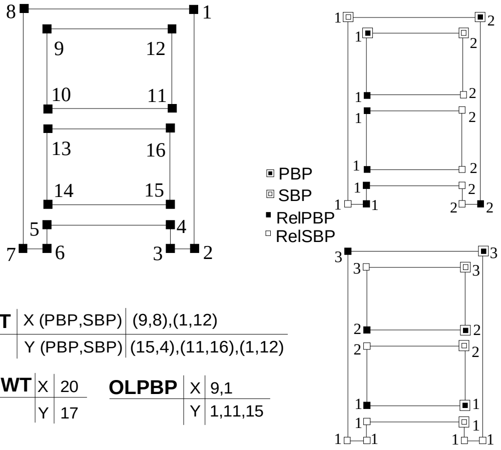
Case 2. Non-vertical stroke with no PBP.

```
(a) if the stroke does not contain any cut segment,
    an arbitrary point is selected as PBP for the X coordinate
    and the other points are assigned as RelPBPs with respect to that point.

(b) if the stroke contains a cut segment,
    (a-1) if there is a corresponding stroke
            which has the same hidden segment and also has a PBP and SBP,
            assign attributes as in case 1.
    (a-2) else
            assign attributes as in case 2(a).
```

The attributes for *Y* coordinates are assigned using similar methods. The procedure is repeated until the attributes of all points are determined.

As a final step, all the generated information is stored in the data structures of BPT, RSWT and OLPBP (defined in Sections 2.3.2 and 2.3.3) Examples of these tables are shown in Figure 13.



BPT	X (PBP,SBP)	(9,8),(1,12)
	Y (PBP,SBP)	(15,4),(11,16),(1,12)

RSWT	X	20
	Y	17

OLPBP	X	9,1
	Y	1,11,15

Point Id	X Pos	Attr	Index	Y Pos	Attr	Index
1	202	PBP	2	220	PBP	3
2	202	RelPBP	2	14	RelSBP	1
3	182	RelSBP	2	14	RelSBP	1
4	182	RelSBP	2	34	SBP	1
5	75	RelPBP	1	34	RelSBP	1
6	75	RelPBP	1	14	RelSBP	1
7	55	RelSBP	1	14	RelSBP	1
8	55	SBP	1	220	RelPBP	3
9	75	PBP	1	204	RelSBP	3
10	75	RelPBP	1	136	RelPBP	2
11	182	RelSBP	2	136	PBP	2
12	182	SBP	2	204	SBP	3
13	75	RelPBP	1	119	RelSBP	2
14	75	RelPBP	1	51	RelPBP	1
15	182	RelSBP	2	51	PBP	1
16	182	RelSBP	2	119	SBP	2

Figure 13. Example of attribute assignment

We have now completed the description of our methods for detecting horizontal and vertical strokes within complex structured outline fonts of Chinese characters. We have introduced the idea of decomposing each character into its basic strokes. Most of the vertical and horizontal strokes (including semi-horizontal and semi-vertical strokes) are correctly detected by this scheme. The next section presents some further results and analyses.

4 IMPLEMENTATION AND EXPERIMENTAL RESULTS

4.1 Overview of implementation

Our automatic hinting system is the fourth module of an Outline Font Production System (OFPS), which has been developed by Human Computers, Inc. The OFPS consists of a Scanning Station, an Automatic Curve Fitting Station, an Outline Inspection Station, an Automatic Hinting Station and a Hinting Inspection Station.

The Scanning Station scans in the original fonts, drawn on paper, and stores them as bitmaps after filtering out any noise. The Automatic Curve Fitting Station is a batch-processing module which extracts the outlines as lines and 3-D Bézier segments from the bitmap characters. The Outline Inspection Station is an interactive outline font editor with which a font designer inspects and modifies the outline fonts generated by the Automatic Curve Fitting Station. The Automatic Hinting Station is implemented according to the scheme set out in this paper. Finally, the Hinting Inspection Station is also an interactive editor with which a font designer inspects and modifies the hinted outline fonts generated by the Automatic Hinting Station. The fonts produced by OFPS are used in commercially available Korean laser beam printers.

The simulator for our intelligent rasterizer is implemented in the C language on a 486 PC, and the results from performance analysis are shown in Sections 4.3 and 4.4. The kernel of our rasterizer is embedded in the laser beam printer described above.

4.2 Results of hinting

We show some of our rasterizing results in the Appendix. The upper row in each pair of rows is generated by the primitive rasterizer and the lower row is generated by our intelligent rasterizer using the hinting information. The sample fonts are selected from representative Chinese characters, and the hinted outline font is directly generated by the Automatic Hinting Station with no further retouching at the Hinting Inspection Station.

4.3 Storage overhead analysis

In this section, we analyze the additional storage overhead for the hinting information needed for stem regularization hinting. The results of analyzing four sets of Chinese characters are shown in Table 1. Each font set contains 4888 Chinese characters.

The average size of the plain outline fonts is 393.0 bytes and that of the hinted outline font is 504.1 bytes. The extra 28.3% space required for storing the hinting information is not excessive compared to the total size of an outline font.

Table 1. Additional storage overhead for hinted outline fonts

	Myungjo	Gothic	Bold Myungjo	Bold Gothic
Number of				
line segments	49.3	42.1	43.0	40.8
curve segments	30.7	20.1	34.2	20.0
Number of				
vertical strokes	5.1	5.5	5.5	5.6
horizontal strokes	4.4	4.8	4.6	4.9
Non hinted data size	450.8 bytes	337.1 bytes	453.5 bytes	330.6 bytes
Hinted data size	569.8 bytes	440.8 bytes	571.3 bytes	434.3 bytes
Additional storage overhead	26.4 %	30.8 %	26.0 %	31.3 %

4.4 Run-time overhead analysis

The additional time for generating the bitmap from a hinted outline font, using our intelligent rasterizer, is shown in Table 2. The generation time was measured on a 486 PC system and is the average elapsed time for generating one Chinese character of the Myungjo typeface.

Table 2. Additional run-time overhead for hinted outline fonts

Units : milliseconds.
Figures in parentheses are percentages of the total time.

		16x16	24x24	32x32	40x40	48x48	128x128	256x256
Non hinted fonts	Scaling	1.38 (8.8)	1.38 (7.8)	1.38 (6.8)	1.38 (6.0)	1.38 (5.4)	1.38 (2.6)	1.37 (1.3)
	Outline Drawing	14.16 (90.6)	16.16 (91.2)	18.64 (91.7)	20.90 (91.9)	23.24 (91.9)	47.70 (88.9)	88.12 (82.0)
	Filling	0.09 (0.6)	0.18 (1.0)	0.31 (1.5)	0.47 (2.1)	0.67 (2.7)	4.57 (8.5)	17.97 (16.7)
	Total	15.63 (100)	17.72 (100)	20.33 (100)	22.75 (100)	25.29 (100)	53.65 (100)	107.46 (100)
Hinted fonts	Scaling	2.43 (14.3)	2.43 (12.8)	2.44 (11.4)	2.43 (10.3)	2.43 (9.2)	2.45 (4.5)	2.43 (2.3)
	Outline Drawing	14.46 (85.2)	16.38 (86.2)	18.64 (87.2)	20.80 (87.7)	23.24 (88.2)	47.64 (87.2)	87.88 (81.2)
	Filling	0.09 (0.5)	0.18 (1.0)	0.31 (1.4)	0.47 (2.0)	0.67 (2.5)	4.54 (8.3)	17.90 (16.5)
	Total	16.98 (100)	18.99 (100)	21.39 (100)	23.71 (100)	26.34 (100)	54.63 (100)	108.21 (100)
Additional Overhead		1.35 (8.6)	1.27 (7.2)	1.06 (5.2)	1.14 (4.2)	1.05 (4.1)	0.98 (0.8)	0.75 (0.7)

As shown in the table, stem regularization hinting is an efficient scheme with low overheads. For the example of a 32×32 bitmap, the additional overhead for stem regularization hinting is 5.2% of total time, which is minor compared to the total processing time.

5 CONCLUSIONS AND FURTHER STUDY

The quality of a font generated mathematically, from an outline description, cannot initially be expected to be better than bitmap fonts carefully designed for a specific size. Various researchers have been trying to enhance the quality of automatically generated characters to match that of hand-edited bitmaps. Good results have already been obtained for Latin fonts but Chinese fonts require a different hinting mechanism and a different automated methodology owing to the complexity of the fonts and the large number of characters for each typeface.

In this paper we have proposed and implemented an intelligent rasterization scheme, with an efficient hinting mechanism, and automation scheme for generating hinted outline characters for the rasterizer. Our rasterization scheme can be summarized as ‘width and interval control for stems’, and this seriously influences the quality of Chinese characters at small sizes. To solve this, we devised an automatic detection mechanism for horizontal and vertical strokes by decomposing the plain outline characters into several basic strokes. The quality of the fonts generated by our scheme is satisfactory; the additional overheads for storage and time are reasonable and entirely satisfactory for use in a practical system.

Research on enhancing the quality of Chinese fonts is only just beginning and we can expect further improvements. For example, we are attempting to minimize the storage requirements for a set of characters by sharing any duplicated strokes in a typeface. Our existing results for automatic decomposition of outline fonts is a starting point for this new work and the enhanced automatic decomposition will be based on a structural glyph analysis of Chinese characters.

REFERENCES

1. Tetsuzou Uehara *et al.*, ‘Shape characteristics of kanji font represented by skeleton vector method’, *IEICE Journal*, **J72-D-II**(11), 1807–1815, (1989).
2. Y. S. Moon and T. Y. Shin, ‘Chinese fonts and their digitization’, in *Proceedings of EP90*, pp. 235–248, (1990).
3. Y. M. Tung, ‘Lccd, a language for chinese character design’, *Software Practice and Experience*, **11**, 1273–1292, (1981).
4. John Hobby and Gu Guoan, ‘Using metafont to design chinese characters’, *Computer Processing of Chinese and Oriental Languages*, **1**(1), 4–23, (1983).
5. Kai. M. Chan *et al.*, ‘Jacm - just another chinese metafont’, in *Proceedings of the 1988 International Conference on Computer Processing of Chinese and Oriental Languages*, pp. 311–315, (1988).
6. H. Jurgensen and H. Y. Wong, ‘Chinese character generator’, in *Proceedings of the 1988 International Conference on Computer Processing of Chinese and Oriental Languages*, pp. 298–302, (1988).
7. Heming Chen and Sinji Ozawa, ‘A methodical generation of various kinds of mincho character’, *IEICE Journal*, **J72-D-II**(9), 1423–1431, (1989).
8. Iwao Sekita Kazuo Toraichi and Ryoichi Mori, ‘On automatic compressing of fonts of high quality characters’, *IEICE Journal*, **J70-D**(6), 1164–1172, (1987).
9. Kuang-Yao Chang *et al.*, ‘A fast automatic chinese multi-style characters zooming and generating system : A novel vector approach’, in *Proceedings of the 1988 International Conference on Computer Processing of Chinese and Oriental Languages*, pp. 258–261, (1988).
10. Y. S. Moon and Y. P. Szeto, ‘Efficient construction of high quality chinese font libraries’, in *Proceedings of the 1988 International Conference on Computer Processing of Chinese and Oriental Languages*, pp. 262–265, (1988).
11. Y. S. Moon and W. K. Hui, ‘High quality chinese fonts generation for desktop publishing — a computer vision approach’, *Pattern Recognition Letters*, **9**, 147–151, (1989).
12. Yamaha Corp. of America, *Super Font Generator LSI GC-1001*, 1990.
13. Seiko Epson Corp., *Epson Outline Font Processor*, 1990.
14. Toshiba Corp., *Font Graphics Accelerator-2 TC85011F*, 1990.
15. Marc Corthout and Evert J. Pol, ‘Supporting outline font rendering in dedicated silicon: the pharos chip’, in *Proceedings of the International Conference on Raster Imaging and Digital Typography II*, pp. 177–189, (1991).
16. Donald E. Knuth, *The METAFONT Book*, Addison-Wesley, Reading, MA, 1986.
17. Roger D. Hersch, ‘Descriptive contour fill of partly degenerated shapes’, *IEEE Computer Graphics and Applications*, **16**(7), 61–70, (1986).
18. Roger D. Hersch, ‘Character generation under grid constraints’, *ACM Computer Graphics*, **21**(4), 243–251, (1987).
19. Roger D. Hersch, ‘Introduction to font rasterization’, in *Proceedings of the International Conference on Raster Imaging and Digital Typography*, pp. 1–13, (1989).
20. Claude Betrisey and Roger D. Hersch, ‘Flexible application of outline grid constraints’, in *Proceedings of the International Conference on Raster Imaging and Digital Typography*, pp. 242–250, (1989).
21. Jakob Gonczarowski, ‘Fast generation of unfilled and filled outline characters’, in *Proceedings of the International Conference on Raster Imaging and Digital Typography*, pp. 97–110, (1989).
22. Roger D. Hersch, ‘Font rasterization: The state of the art’, *Tutorial Paper Supported by EEC DIDOT-COMETT project*, (1991).
23. Apple Computer Inc., *Macintosh System Software Release 7.0 Outline Fonts — Preliminary Developer Note*, 1989.
24. Hewlett Packard Inc., *Creating Intellifont-compatible Fonts using the AFGA Compugraphic Standard*, 1989.
25. Sun Microsystems Inc., *TypeMaker Reference Manual*, 1989.
26. Adobe Systems Inc., *Adobe Type 1 Format*, 1990.
27. Microsoft Corp., *True Type Font Files*, 1990.

28. Chialing Ou and Yoshio Ohno, ‘Font generation algorithms for kanji characters’, in *Proceedings of the International Conference on Raster Imaging and Digital Typography*, pp. 123–133, (1989).
29. Sten F. Andler, ‘Automatic generation of gridfitting hints for rasterization of outline fonts or graphics’, in *Proceedings of EP90*, pp. 221–234, (1990).
30. Roger D. Hersch and Claude Betrisey, ‘Model-based matching and hinting of fonts’, *ACM Computer Graphics*, **25**(4), 71–80, (1991).
31. Bryan D. Ackland and Neil H. Weste, ‘The edge flag algorithm — a fill method for raster scan displays’, *IEEE Transactions on Computers*, **C-30**(1), 41–47, (1981).

APPENDIX

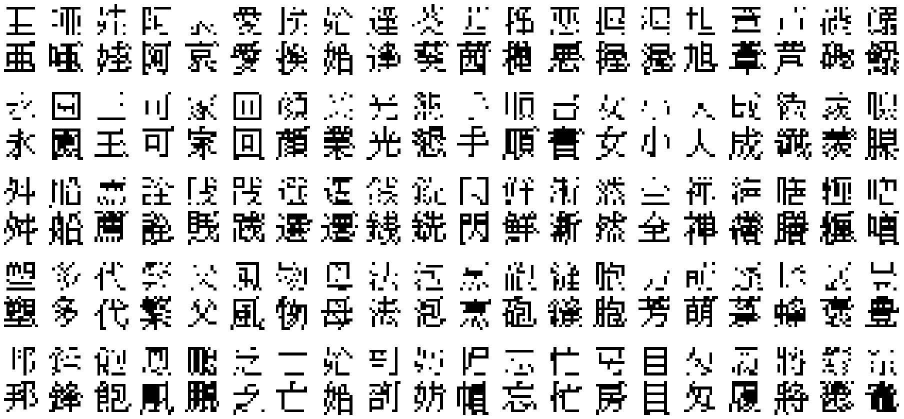


Figure 14. 12 × 12 bitmaps enlarged

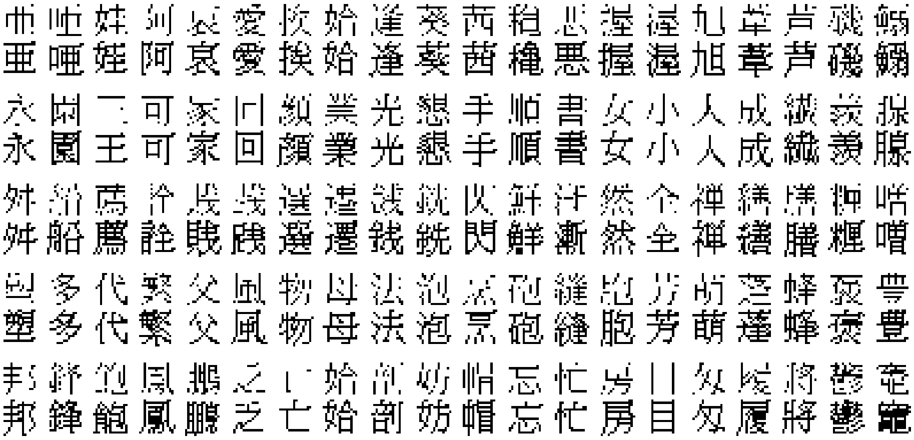


Figure 15. 16 ×16 bitmaps enlarged

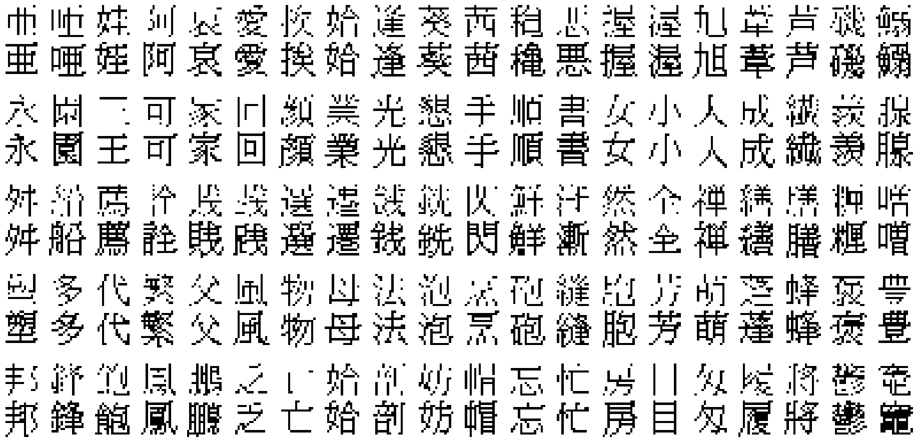


Figure 16. 20 × 20 bitmaps enlarged

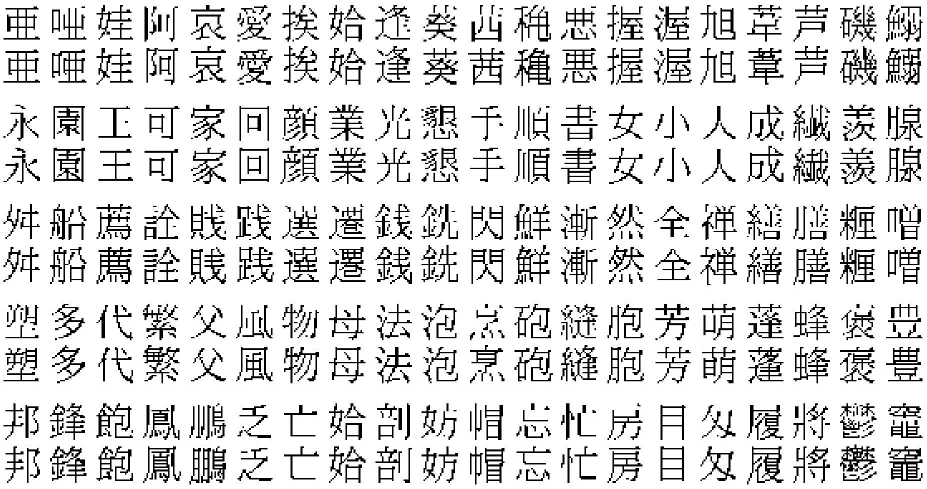


Figure 17. 24 × 24 bitmaps enlarged

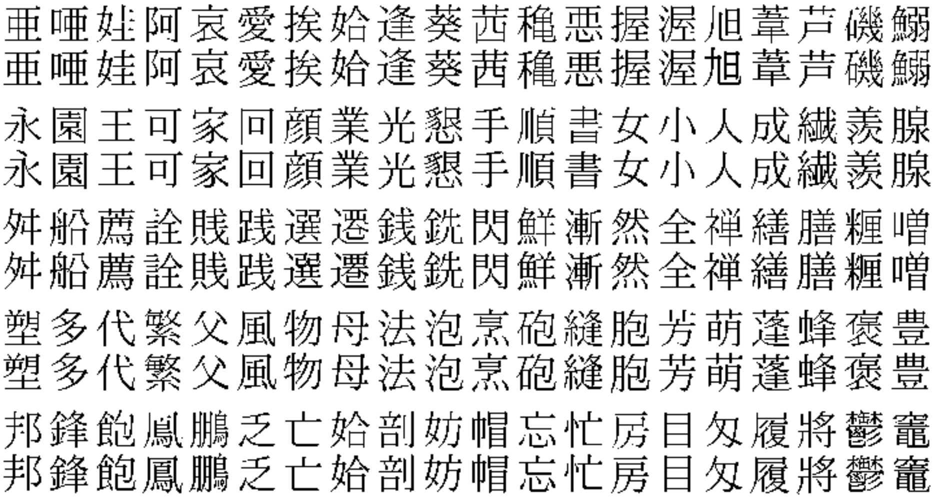

亜唖娃阿哀愛挨姶逢葵茜穐悪握渥旭葦芦磯鰯
亜唖娃阿哀愛挨姶逢葵茜穐悪握渥旭葦芦磯鰯
永園王可家回顔業光懇手順書女小人成繊羨腺
永園王可家回顔業光懇手順書女小人成繊羨腺
舛船薦詮賎践選遷銭銑閃鮮漸然全禅繕膳糎噌
舛船薦詮賎践選遷銭銑閃鮮漸然全禅繕膳糎噌
塑多代繁父風物母法泡烹砲縫胞芳萌蓬蜂褒豊
塑多代繁父風物母法泡烹砲縫胞芳萌蓬蜂褒豊
邦鋒飽鳳鵬乏亡姶剖妨帽忘忙房目匁履將鬱竈
邦鋒飽鳳鵬乏亡姶剖妨帽忘忙房目匁履將鬱竈

Figure 18. 32 × 32 bitmaps enlarged

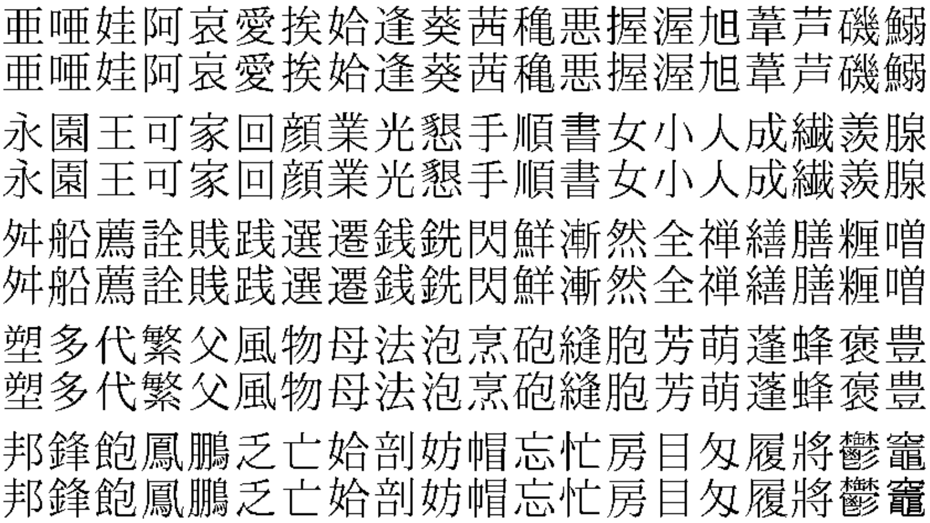

亜唖娃阿哀愛挨姶逢葵茜穐悪握渥旭葦芦磯鰯
亜唖娃阿哀愛挨姶逢葵茜穐悪握渥旭葦芦磯鰯
永園王可家回顔業光懇手順書女小人成繊羨腺
永園王可家回顔業光懇手順書女小人成繊羨腺
舛船薦詮賎践選遷銭銑閃鮮漸然全禅繕膳糎噌
舛船薦詮賎践選遷銭銑閃鮮漸然全禅繕膳糎噌
塑多代繁父風物母法泡烹砲縫胞芳萌蓬蜂褒豊
塑多代繁父風物母法泡烹砲縫胞芳萌蓬蜂褒豊
邦鋒飽鳳鵬乏亡姶剖妨帽忘忙房目匁履將鬱竈
邦鋒飽鳳鵬乏亡姶剖妨帽忘忙房目匁履將鬱竈

Figure 19. 40 × 40 bitmaps enlarged