Oriental Fonts Auto Boldness

By
Lo I Fan

thesis
QA
76.9
I55L6
1994

26 MAY 1995

By
Lo I Fan

A DISSERTATION
SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF PHILOSOPHY
DIVISION OF COMPUTER SCIENCE
THE CHINESE UNIVERSITY OF HONG KONG
SEPT., 1994

ABSTRACT

Bitmap fonts are notorious for inflexibility and memory consumption. In the recent years, outline fonts have gradually replaced bitmap fonts in many word processing environment because the scalability of outline fonts fulfills the principle of "What You See Is What You Get", and it is compact in storage. In addition, expanded version and italic version can be generated in an ad hoc manner from a master outline by geometrical transformations. However, to generate a bold version from a stored unbold version in an ad hoc manner, which is termed as "Auto Boldness", is a deep problem because the outline of a character must be modified intelligently to attain bold effect. Bolding Chinese characters is especially challenging. It is due to the fact that outlines of Chinese characters are very delicate and complex. The existing methods of auto boldness for Chinese outline fonts are undesirable, owning to poor quality of bold version and memory consuming. Consequently, we invented an efficient and fast algorithm to bold Chinese outline fonts automatically with a great flexibility of multi-level boldness. That is thousands of bold versions with different blackness can be generated in an ad hoc manner. Moreover, the quality of bold is good, and memory requirement is reasonable. In Chapter one, the concepts of outline fonts, and the existing methods of outline fonts auto boldness will be introduced. Then, the main ideas of our solution to Chinese outline fonts auto boldness will be presented step by step in Chapter two. The detailed algorithms of auto boldness will be presented in Chapter three and four. Chapter three talks about the functions and the algorithms of a set of auto boldness primitives from which the auto boldness programs for outline fonts can be built. Chapter four talks about a rule based system to analyze the Chinese character outline in order to generate the auto boldness program on character basis, which executes the auto boldness primitives. Finally, Chapter five will contain with a performance assessment and a discussion of the pros and cons of our solution.

ACKNOWLEDGMENTS

This project would not have been possible without the expertise and cooperation of many people. Our biggest debt of gratitude goes to our project supervisor Dr.Y.S.Moon, whose ideas and support for the project are ever so appreciated.

Notations Convention

[N] denotes N th temporary equation during a derivation process, where N is an integer. The equations of a derivation process are numbered consecutively. The derived formula is denoted by (C.N), that means the N th formula of chapter C, where C and N are integers.

Table of Contents

Chapter 1: Introduction

1.1 The Evolution of Fonts 1
1.2 Bitmap Fonts 2
1.3 Outline Fonts
    1.3.1 Arc and Vector Form 4
    1.3.2 Spline Form 4
    1.3.3 Pros and Cons of Outline Fonts 8
1.4 Examples of Outline Fonts
    1.4.1 Adobe's PostScript 9
    1.4.2 Apple's and Microsoft TrueType
        1.4.2.1 Outline Representation 10
        1.4.2.2 Rasterisation 12
        1.4.2.3 Hinting 13
1.5 Bold Fonts
    1.5.1 Definition of Bold 15
    1.5.2 Definition of Auto Boldness 16
    1.5.3 Auto Boldness by Double Printing 17
    1.5.4 Auto Boldness by Multi-Master Technique 18
1.6 Chinese Fonts
    1.6.1 Chinese Character Sets 19
    1.6.2 The Subtleties of Chinese Fonts Auto Boldness 21
1.7 Project Objective 23
1.8 Goals 23

Chapter 2: Main Ideas of Chinese Font Auto Boldness

2.1 Prototype of Auto Boldness Driver 24
2.2 Design Features of the Prototype Auto Boldness Driver 25
2.3 Data Structure and Algorithm of Auto Boldness
    2.3.1 Data Structure of TrueType Character Outline 27
    2.3.2 Algorithm of Auto Boldness 28
    2.3.3 Algorithm Description 29
2.4 Component Font Auto Boldness 35

Chapter 3: Language of Auto Boldness

3.1 Enhancements of TrueType Engine to support Auto Boldness 36
3.2 Symmetric Bold Instruction 38
3.3 Rotate Bold Instruction 47
3.4 Asymmetric Bold Instruction 50
3.5 Comparison of Bold Instructions 54
3.6 Serif Accommodation Instruction 55

Chapter 4: Shape Parsing and Auto Bold Code Generation

4.1 Compilation Process and Auto Boldness 62
4.2 Shape Lexical Analyzer 64
4.3 Shape Token Attributes Evaluation
- 4.3.1 **line** Token 66
- 4.3.2 **bezier2** Token 67
- 4.3.3 **sharp** Token 70
- 4.3.4 **concave** Token 75
- 4.3.5 **convex** Token 75

4.4 Scope of Shape Parsing 76
4.5 Shape Parsing Mechanism 77
4.6 Model Grammar Rules
- 4.6.1 Grammar Rule Format 81
- 4.6.2 Grammar Rule Item 82
- 4.6.3 Grammar Rule Assignment 83
- 4.6.4 Grammar Rule Condition 83

4.7 Auto Boldness Code Generation 84
4.8 Program Methodology of Prototype Auto Boldness Driver 86

Chapter 5: Conclusions

5.1 Work Achieved 87
5.2 The Pros and Cons of Auto Boldness Algorithm 88
5.3 Bold Quality Assessments 91
5.3 Future Directions 93

References

Appendix One

Appendix Two

Chapter One: Introduction

1.1 The Evolution of Fonts

The invention of printing technology marked a new era in human history. Ever since, ideas and knowledge could be printed as documents, and accumulated for generation after generation. The most essential element of printing is letterform. Many technologies for the production of letterforms have preceded our modern methods, including stone carving, pen and ink, wood-block printing, movable type. Both the speed and cost of printing were not desired at that time. Moreover, only a very experienced artist was capable of producing legible letterforms. The difficulties in designing letterforms is that printed characters must appear visually related to one another and stylistically consistent.

Then, the typewriter emerged, and the process of printing was mechanized by setting pieces of lead into a press. Hence, the speed of printing was improved, and people could learn how to use this machine very well in a short time period. The pieces of lead were called metal type which lasted for over 500 years. Metal type of different sizes and styles were available for typesetting. Metal type has shortcomings such as being unable to print large size letters, and limited styles.

In the twentieth century, the printing technology was beginning to change quite rapidly. Photo composition had been in general use for less than 20 years. Thus, a great deal of efforts was spent on converting the typefaces designed for metal into something appropriate for photo composition. At the same time, computer printing technology emerged, and digital technology adopted in output devices. Hence, the same type of reworking would again be necessary to generate the fonts conforming to digital technology, that is bitmap fonts.

Bitmap fonts can be immediately output to a printing device, but it requires a large amount of storage space. Thus, outline fonts emerged, and replaced bitmap fonts. Outline font has many excellent advantages. The storage requirement of a outline font is much less than that of a bitmap font, and different sizes and styles can be generated "on the fly". But, it must go through a rasterisation process to generate raster image, before outputting to a

printing device. Moreover, in order to guarantee the quality of the raster image, hinting must be applied before rasterisation, resulting in relatively slow speed.

1.2 Bitmap Fonts

Until the recent years, computer printing has become the most popular printing method, and gradually replaced the old ones. There are several methods of storing, representing, and reproducing fonts inside a computer. They differ in their economy, efficiency, and typographic utility.

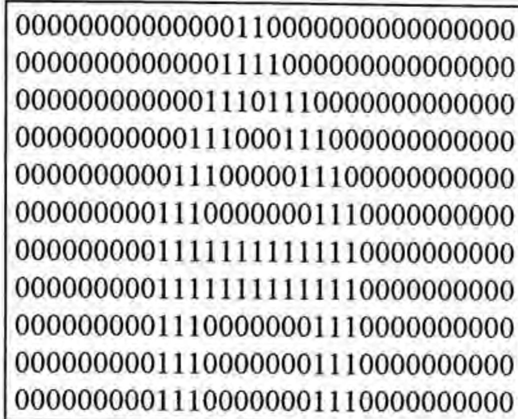


Figure 1.1: Letter A in Bitmap Form

The simplest method is to store the glyphs as arrays of bits - that is , as bitmaps, as shown in fig 1.1. The printing speed of bitmap font is the fastest, compared with other computer fonts, because bitmap is the identical way that fonts are used in most output devices. However, bitmap font requires a large amount of computer storage.

For example, a 12-point font, stored as a bitmap at 75 dpi, might occupy about 1.5K bytes. The storage required is proportional to the area of the bitmap, and proportional to the square of the change in size, and also as the square of the resolution.

Suppose, storage required = S (in bytes), point size = P and resolution = R (in dpi)

$$S = \frac{P^2}{12^2}\frac{R^2}{75^2}1.5 \quad = \frac{P^2 R^2}{540000} \qquad (1.1)$$

Therefore, twenty-four-point type consumes four times the space at the same resolution, that is 6K bytes in this example. 1200-dpi 12-point type would require

$$1.5 * \frac{1200^2}{75^2} = 384K \text{ bytes - 256 times the storage of the original font.}$$

The storage required will be tremendous, if fonts of many different styles and sizes must be available.

Bitmap font can be gained by converting the metal type or phototypesetting master to raster image. But, if the dimensions of the original design are not multiples of the pixel spacing at the required resolution, some features in the original design, such as symmetry, serif and slender join, will not be retained in the raster image. Thus, as a matter of fact, all digitized outlines must be edited and adjusted, even though the scanning process is perfect.

In order to generate a high quality bitmap font at a particular resolution, there are three methods of bitmap font design: top town, bottom up and collateral. If an idealized design of high resolution is used as the basis for a type family, with lower-resolution fonts derived from it, it is called a top-down design. The lower the resolution is, the worse of the quality of the derived fonts. Conversely, if an individual font or two of low resolution is used as the basis of the family, with high-resolution fonts derived from it, it is called a bottom-up design. The higher the resolution is, the worse of the quality of the derived fonts. Finally, collateral design seeks to optimize the overall quality throughout all resolutions, but sacrifices the quality at the extremes, lowest resolution and highest resolution.

1.3 Outline Fonts

The storage needed for a bitmap font increases rapidly as the point size or resolution increases. Moreover, the quality of an algorithmically generated bitmap font at a particular resolution from the basis of the family is also not desired. So, can a font be represented in a way that is resolution independent, and compact in storage ? The idea of outline font is to store only points on the periphery of letterforms. Figure 1.2 shows how an outline font might be described.

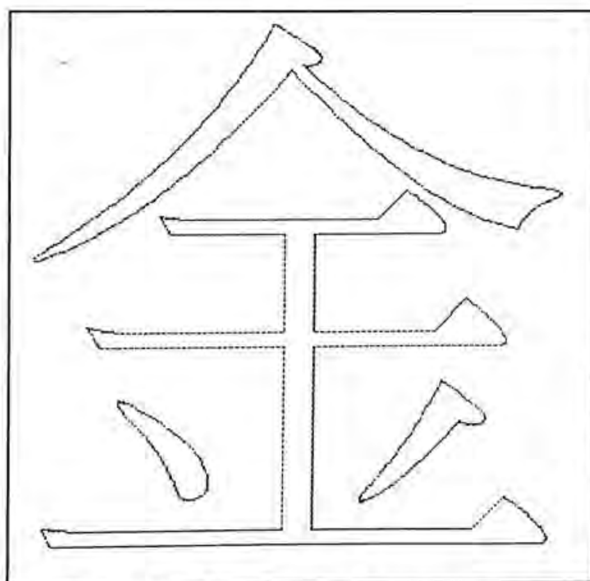


Figure 1.2: Character Shape Stored as Outline

1.3.1 Arc and Vector Form

If the character shapes are encoded as polygons, a great many data points is necessary to approximate curves accurately. A method that embodies curves intrinsically gives better results. One of the encoding methods is circular arcs and straight lines. As shown in figure 1.3, a dot stroke of Chinese font is represented by three joined circular arcs. The center of arc P1,P2, arc P2,P3,P4, and arc P4,P1 are C1,C3, and C2 respectively. Circular arcs and straight lines representation is compact in storage, because it is only needed to store the centers of arcs, and end points of arcs. However, a circular arc can only represent a smooth curve segment, and fails to represent a delicate outline; otherwise enough segments must be used to represent a delicate outline, but it defeats the purpose of compact storage. For this reason, various alternatives to represent outline are needed.

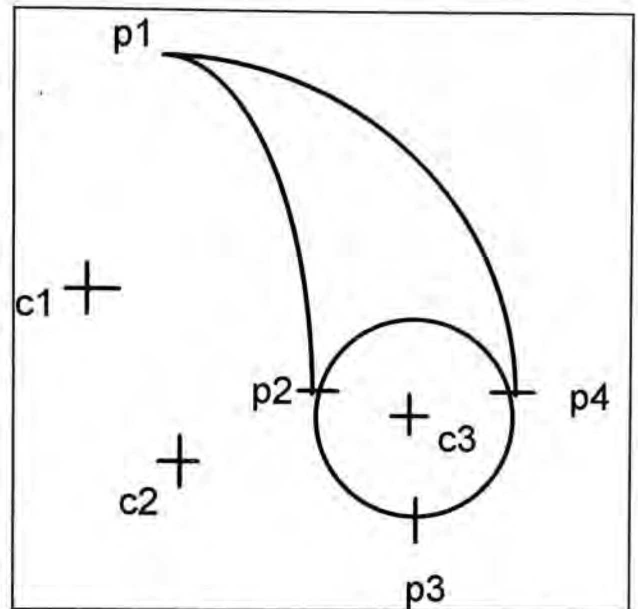


Figure 1.3: Represent a Stroke of Chinese Font by Circular Arcs

1.3.2 Splines Form

Splines are curves that are controlled by a small set of given points, tangents, or other data. To approximate a particular curve, the curve must first be broken into segments that meet at their endpoints. The meeting points are called joints. Then, each segment can be specified by a parametric formula. The parametric formula supplies points of the segment, generated by varying a parameter over a specified range. For instance, any point (x,y) along the curve segment can be described parametrically as two equations:

1. $x = x(t)$
2. $y = y(t)$ such that $t \in [0,1]$

where t is a parameter. The value t = 0 corresponds to the starting point of the segment. While t = 1.0 gives the end point of the segment.

The parametric equations of a spline segment have a general form:

$$x(t) = a_0 + a_1 t + a_2 t^2 + ... + a_{n-1} t^{n-1} + a_n t^n$$
$$y(t) = b_0 + b_1 t + b_2 t^2 + ... + b_{n-1} t^{n-1} + b_n t^n \qquad (1.2)$$

where a_i and b_i are coefficients and n is the order of the polynomials. When n is 2, the polynomials are called conic. When n is 3, they are cubic. The constants are carefully chosen in order to attain smoothness at the joints. Generally speaking, splines of order n have continuity in the (n-1) derivative at each joint. Thus, a conic spline has a continuous first derivative, and a cubic spline has continuous first and second derivatives at the joints. However, in order to simulate the sharp points of a character outline, a discontinuity in the slope at the joints must be created.

The many kinds of parametric curves that can be used to simulate a character outline fall into two categories: those for which the curve is constrained to pass through the control points, called interpolating curves, and those without this constraint. Non interpolating curves have control points both on and off the curves. The on-curve points at the joints specify the curve's location, while the off-curve points determine the slope and shape of the curve as it passes through and between these points. Thus, during the character outline design process, the designer can control the curve's location and slope delicately by moving the on-curve and off-curve control points with a graphic mouse.

Bezier splines are one important class of cubic splines, because only a few number of segments is required to describe a complex shape of a curve. Moreover, it is relatively fast computationally, and offer both kinds of user interaction - on-curve and off-curve control points.

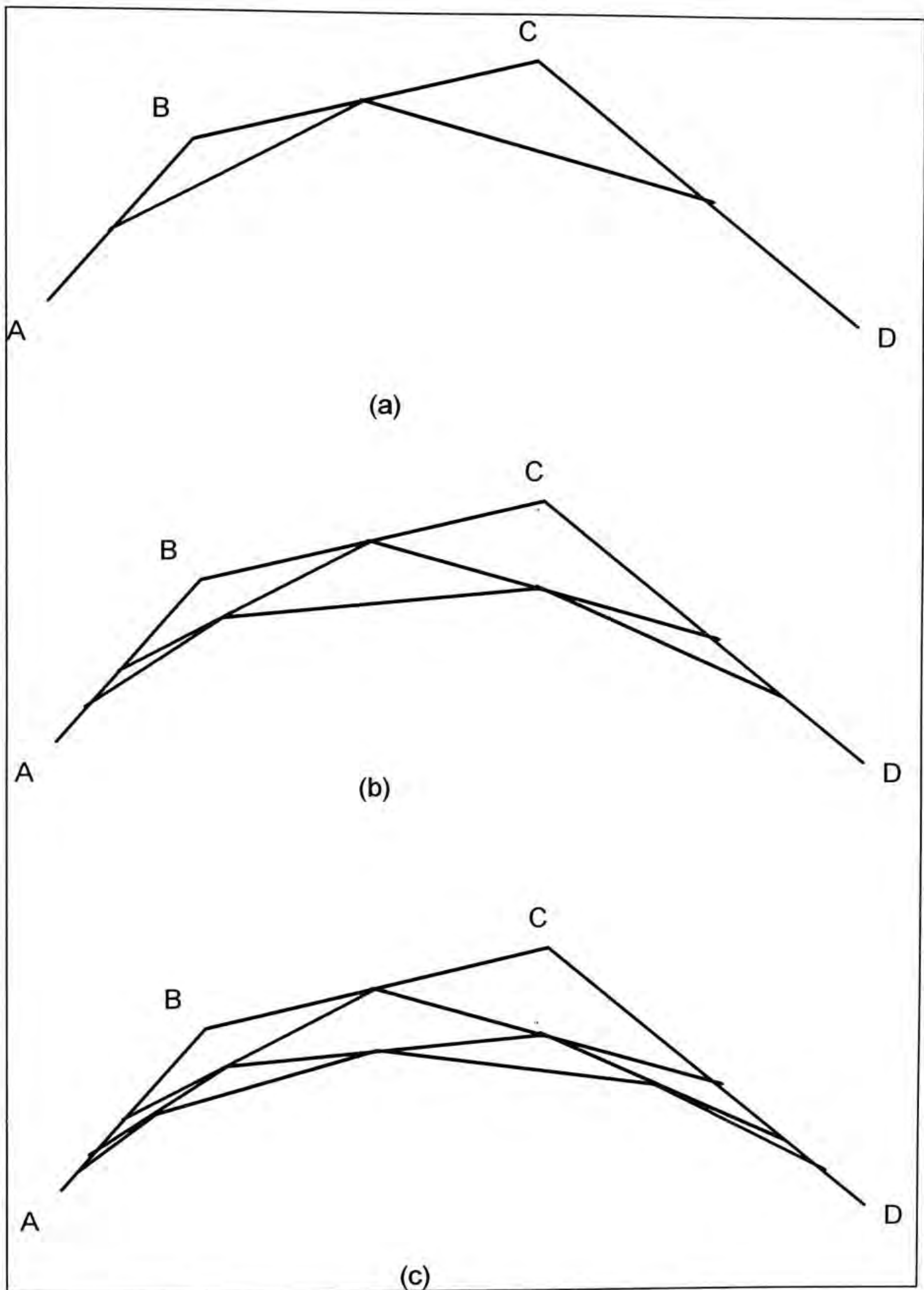


Figure 1.4: Recursive Construction of Bezier Spline Curve

A Bezier curve has four control points, two on-curve, and two off-curve. The behavior of the curve can be described recursively[Knuth 86]. As shown in fig 1.4(a), the four points (A,B,C,D) are the four control points of a Bezier curve. A,B are on-curve point, and C,D are off-curve points. At the first iteration, the four control points are connected by straight lines, and a polygon is formed. At the second iteration (fig 1.4(b)), the midpoints of the lines are connected in sequence. This process continues, connecting adjacent midpoints, until the polygon converges to a curve. This curve is the spline curve defined by the four control points (A,B,C,D).

Bezier spline has the following properties:

1. It goes through the end points (A,D)
2. The tangent at A is in the direction of the line joining A and B.
3. The tangent at D is in the direction of the line joining C and D.
4. The curve is staying within the convex hull of the four points.

Knuth gives a simple formula, in complex-variable notation, to describe a Bezier spline[Knuth 86].

$$Z(t) = (1-t)^3 Z_1 + 3(1-t)^2 tZ_2 + 3(1-t)t^2 Z_3 + t^3 Z_4 \qquad (1.3)$$

where Z_i are the control points (X_i, Y_i) and t is the parameter ranging between 0 and 1.

Four control points give rise to a cubic Bezier curve, whilst three control points give rise to a quadratic Bezier curve. A quadratic Bezier curve has two on-curve control points, and one off-curve control point. The same recursive algorithm can be applied to the three control points until at the limit a curve is formed. Thus, a quadratic Bezier has the same properties of cubic Bezier. If the quadratic bezier is defined by control points A (on-curve), B (off-curve) and C (on-curve), the curve will go through A and C, and off-curve control point B will control the tangents at A and C. Moreover, the curve is staying within the convex hull of the three points.

A simple formula, in complex-variable notation, to describe a quadratic bezier is:

$$Z(t) = (1-t)^2 Z_1 + 2t(1-t)Z_2 + t^2 Z_3 \qquad (1.4)$$

where (Z_1, Z_2, Z_3) are the three control points. (Z_1, Z_3) are on-curve control points. Z_2 is the off-curve control point. t is in [0.0,1.0].

1.3.3 Pros and Cons of Outline Fonts

The pros and cons of outline representation can be summarized as follows:

Pros:

1. **Compact Storage:** only store the control points of outline, instead of storing pixel by pixel in bitmap representation.

2. **Easily Scaled:** Letters of varying size with the same design can be created from the same data. Mathematically, the outline version of varying size can be generated by multiplying the control point coordinates of the original outline by a 2x2 scaling matrix.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s & 0 \\ 0 & s \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad (1.5)$$

where (x,y) is a control point in original outline, and (x',y') is a control point of the scaled outline. s is the scale. If $s > 1$, the outline will be scaled up. If $s < 1$, the outline will be scaled down.

3. **Expanded Version:** expanded versions can be calculated by compression or expansion in the x-dimension. Mathematically, the outline version of varying width can be generated by multiplying the control point coordinate of original outline with a 2x2 matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} S_x & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad (1.6)$$

where S_x is the scale in the x-dimension.

4. **Italic Version:** italic versions can also be generated on the fly.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & S \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \qquad (1.7)$$

where $S > 0$, the greater the value of S is, the more slant the character outline has. Normally, take $S = 0.2$

Cons:

1. **Expensive Rasterisation Process:** the outline form can not directly be sent to an output device. It must go through a rasterisation process to generate a character bitmap, before being sent to output device. The rasterisation process is time-consuming, especially in the case of oriental fonts. Thus, it is worthy of developing an efficient algorithm to do rasterisation.

2. **Undesired Output Quality:** If the dimensions of the character outline, such as serif height, stroke width, etc., are not a multiple of the intended resolution, the bitmap generated in rasterisation process will lose the features of the original outline. Thus, a process called hinting must be applied to the character outline, before rasterisation. In a hinting process, the character outline control points are migrated slightly so that dimensions of the character outline can be adjusted to be multiple of the intent resolution. Then, the generated bitmap can retain the features of the original outline.

1.4 Examples of Outline Fonts

1.4.1 Adobe's PostScript

The Adobe's PostScript language is a simple interpretive programming language with powerful graphics capabilities. Its primary application is to describe the appearance of text, graphical shapes, and sampled images on printed pages. PostScript treats any output as graphical shapes. It provides many commands for drawing graphics, such as straight lines, arcs and cubic Bezier curves. In addition, it provides some control commands, such as for-loop and if-then-else.

There are three groups of operators in PostScript language: Level 1, Level 2, and Display PostScript operators. Level 1 operators are basic operators for simple graphics, text and font. Level 2 operators include all basic operators. The functions of Level 2 operators include dictionaries operations, memory management (virtual memory), resource management, device setup, and the operations for composite fonts.

The PostScript interpreter uses a character's code to select the definition from the dictionary to generate a characters shape. As a matter of fact, a character's definition is a procedure that executes graphics operations, such as straight line and cubic Bezier to produce the character's shape, instead of storing the character outline's control points. There are several kinds of fonts, each distinguished by the FontType entry in the font dictionary. Each type of fonts has its own conventions for organizing and representing the information within it. The font types defined are:

(a) Type 0 is a composite font composed of other fonts called base fonts (Type 1 or Type 3 font), organized hierarchically. Composite fonts are a Level 2 feature.

(b) Type 1 is a base font that defines character shapes by using a specially encoded procedure. It is faster and more powerful than Type 3 font.

(c) Type 3 is a user-defined base font that defines character shapes as ordinary PostScript language procedure.

Both Type 1 and Type 3 are base fonts which are restricted to a maximum of 256 characters per font. The major differences is that Type 1 font support hinting. Type 0 font is a composite font to support a very large character sets. It is suitable for Oriental fonts, such as Japanese and Chinese characters.

1.4.2 Apple's and Microsoft TrueType

1.4.2.1 Outline Representation

Both PostScript and TrueType can be regarded as outline fonts. PostScript generates a character shape by executing a corresponding procedure. In contrast, TrueType generate a character shape by scan converting the character outline which is stored in a font file. Moreover, PostScript can handle computer graphics and image, but TrueType has no such a capacity.

In a TrueType font, glyph shapes are described by their outlines. A glyph outline consists of a series of contours. A simple glyph may have only one contour. More complex glyphs can have two or more contours. Composite glyphs can be constructed by combining two or more simpler glyphs. Certain control characters that have no visible manifestation will map to the glyph with no contours.

Contours are composed of straight lines and quadratic Bezier curves. In the font file, the control points information can be found in "glyf" table of font file. Each contour is stored as an order list of numbered control points. There are two types of control points: on-curve control points and off-curve control points. Three scenarios will happen in the list.

Case 1: Two consecutive on-curve control points represents a straight line primitive.

Case 2: Three consecutive control points (A,B,C) such that A is on-curve, B is off-curve and C is on-curve represent a quadratic Bezier.

Case 3: Four consecutive control points (A,B,C,D) such that A is on-curve, B is off-curve, C is off-curve and D is on-curve represents two joined quadratic Beziers. This method can save the storage space of one on-curve control point, because there is a hidden on-curve control point between B and C, and can be calculated by finding the mid-point (M) of B and C. Then, (A,B,M) and (M,C,D) represent two joined quadratic Beziers.

As shown in fig 1.5, the on-curve points are shown as black circles, and off-curve points are shown as open circles. The points are numbered consecutively along the contours. The contour A of points [0,1,2,....,48] is in anti-clockwise direction, and the contour B of points [49,50,51,52] is in clockwise direction, and the contour C of points [53,54,55,...,60] is in clockwise direction. It is obvious that contour B and contour C is enclosed by contour A. It is generally true that the direction of the enclosed contour is the reverse of the direction of the enclosing contour.

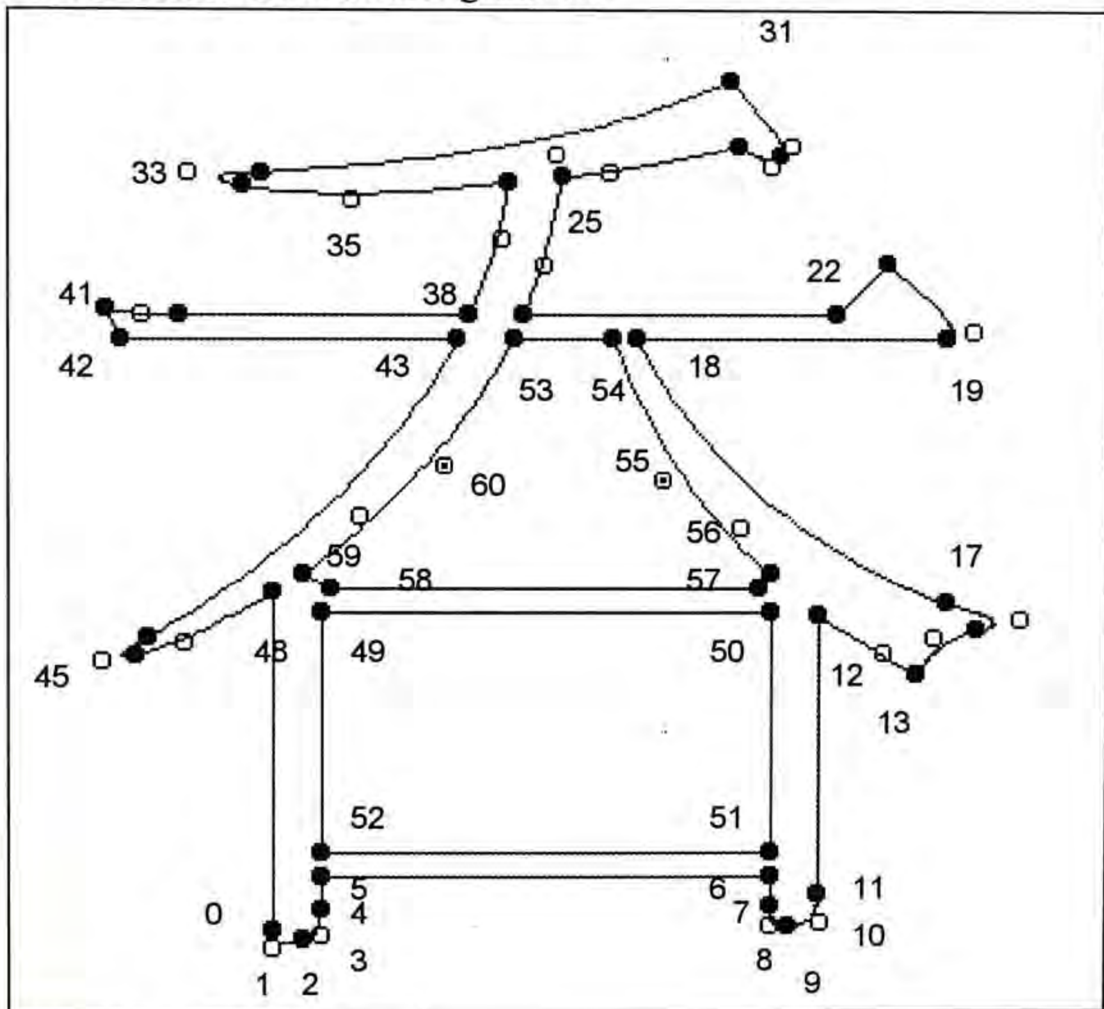


Figure 1.5: A Glyph Outline

1.4.2.2 Rasterisation

Once the master outline has been scaled and grid-fitted, it is ready to be rasterized by the scan converter [True 91]. The scan converter takes the grid-fitted outline and applies a set of rules to determine which pixels will be part of the glyph image when printed or displayed on the screen.

The first of these rules is as follows:

Rule 1: If a pixel's center falls within or on the glyph outline, that pixel is turned on and becomes part of the bitmap image of the glyph.

The TrueType scan converter uses the non-zero winding number rule to distinguish the interior from the exterior of a glyph. This rule is as follows:

Points that have a non-zero winding number are inside the glyph. All other points are outside the glyph.

The winding number of a point can be found by following the steps:

1. Draw a ray from the point in question toward infinity.
2. Starting with a count of zero.
3. Add one to the count each time a glyph contour crosses the ray from right to left or bottom to top. (Such a crossing is termed as on-transition because the TrueType scan converter scans from left to right and bottom to top.)
4. Subtract one from the count each time a contour of the glyph crosses the ray from left to right or top to bottom. (Such a crossing is termed an off-transition.)
5. If the final count is non-zero, the point is an interior point. Otherwise, it is an exterior point.

An on-transition is shown in fig 1.6. Here the contour crosses the ray from bottom to top. An off-transition is shown in fig 1.7. Here the contour crosses the ray from left to right.

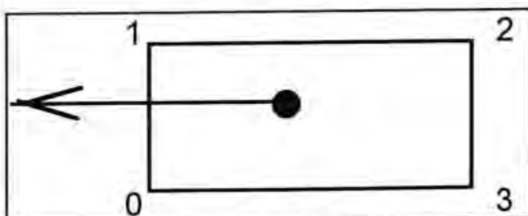


Figure 1.6: An On-Transition

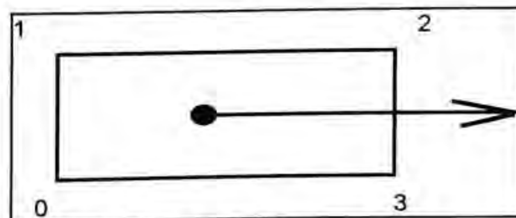


Figure 1.7: An Off-Transition

1.4.2.3 Hinting

Outlines can be rasterized to generate bitmap on the fly, but unfortunately such bitmaps show quantizing errors (quantum errors) leading to unacceptable representation at low resolution. The quantum errors are escalated in Chinese font, because Chinese characters are mainly composed of horizontal and vertical strokes. If the stroke width of the original outline is not a multiple of the device resolution, the bitmap generated will have unequal stroke width, resulting in uneven distribution of black and white in the bitmap. Moreover, owning to discrete effect, the serif in the original outline might not be kept in the raster image. It is quite unacceptable.

In order to render fonts with high typographic quality on middle and high resolution devices, outline grid adaptation techniques have been developed which generate raster characters of improved quality. Hints are applied to the character outline to migrate control points, before rasterization takes place. Hints provide a precise phase control in order to generate raster characters while keeping essential typographical and geometrical features.

In English font, the features modifications include [Final 92]:

a. Grid fitting adjustment:
- Adjustments to character's height, width, or inter-character spacing
- Migration of stroke positions to prevent dropouts
- Fattening of strokes caused by other dropout compensations
- changes to maintain stroke or curve symmetry and smoothing

b. Regularization:
- Equalization of stem weights
- Loss of contrast between stems and hairlines caused by minimum pixel size
- Additional shape distortions to maintain similar weights of curved and diagonal strokes, when compared to vertical and horizontal strokes

c. Readability corrections:
- Increasing the x-height relative to the cap-height
- Increasing the bowl sizes
- Squaring of the character shapes to preserve readability

TrueType hinting is quite different from PostScript Type 1 Font Hinting. In PostScript Type 1, the hinting system depends on an intelligent rasterization algorithm to render character outline correctly. The algorithm is built into the PostScript interpreter and hidden from the users. To hint a font, only constraints on the size and positioning of character features, such as width of a stem or x-height, need to be stated [Adobe 90].

In contrast to PostScript, TrueType hinting-programming language provides a large instructions set for hinting, including treatment of freedom vector and projection vector, nine explicit rounding options, cut-in services and delta hints, [True91]. Instructions can be associated with particular glyphs or can be associated with a font as a whole. Instructions associated with a particular glyph are termed a glyph program. Instructions associated with a font as a whole are termed a font program.

The font program (found in the 'fpgm' table in the font file) is a set of instructions executed once, the first time a font is accessed by an application. The font program is used to create function definitions and instruction definitions. Functions and instructions defined in the font program can be accessed in the individual glyph programs.

The control value program (found in the 'prep' table in the font file) is a sequence of instructions executed every time the point size or transformation changes, The control value program is used to make font wide changes rather than to manage individual glyphs.

The 'glyf' table in the font file stores both the outline control points and the individual glyph programs. Instructions associated with a glyph are executed every time that glyph is requested.

1.5 Bold Fonts

1.5.1 Definition of Bold

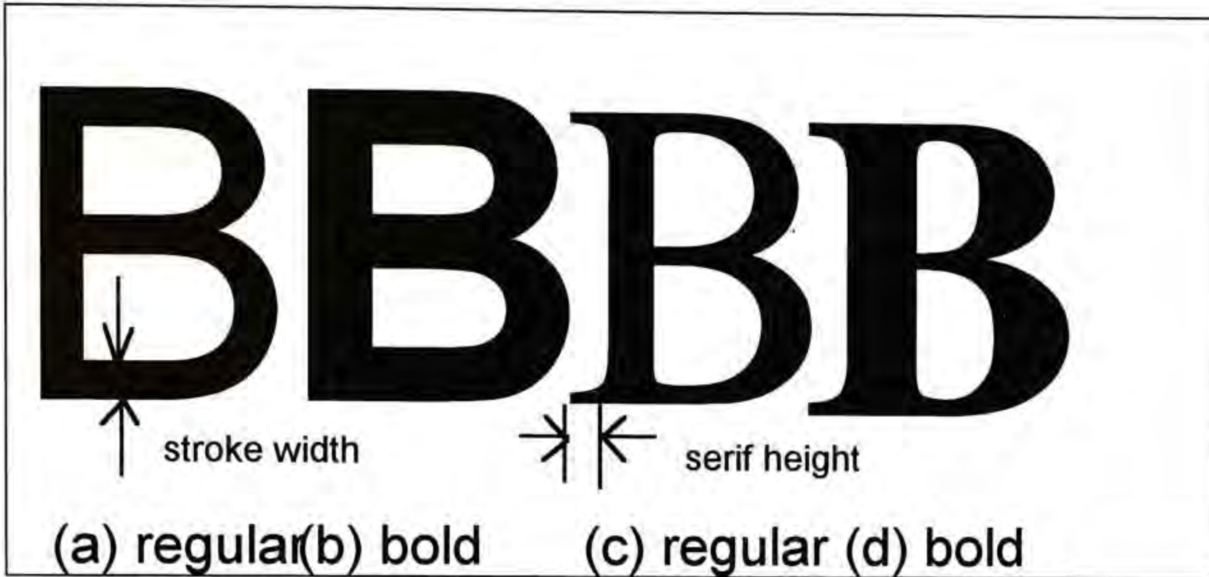


Figure 1.8: Regular and Bold

The purpose of bold is to darken a selected area of text. The raster image of a character looks darker, because the dark area is expanded while the white area is diminished. At the same time, the original features, including symmetry, serif height, outline continuity, etc., are kept in the bold version. There is two approaches of bold. The first approach is to increase all strokes width of a character evenly. As shown in fig 1.8b, the bold character's strokes width are increased evenly. This approach is suitable for sans serif fonts. The second approach is to increase the contrast of a character by only increasing the vertical stroke width, as shown in fig 1.8d. This approach is suitable for serif fonts.

The boldness of a character can be expressed quantitatively by its weight

$$W = \frac{T}{x} \qquad (1.8)$$

where T is the stem width and x is the x-height.

The larger the value of W, the darker each letter will appear. However, this formula can only be applied for a regular typeface, because even though the weights of two typefaces, narrow face and wide face, are equal, narrow face will appear darker than wide face. Moreover, it is very hard to define the stroke width of a typeface exactly, because the stem width of some typefaces varies from letter to letter. The waist measurement of a letter 'I' can be used as an approximation, or an average value can be used.

Typographic contrast can be defined as the ratio of the weights of vertical stems to horizontal stems.

$$C_T = \frac{T_v}{T_h} \qquad (1.9)$$

where T_v is the vertical stem width and T_h is the horizontal stem width.

Many traditional typefaces have high contrast, in excess of 3.0, giving them a look that typographers term brilliant or glittery. Extremely high contrast results in striking, but less legible typefaces. Low contrast ratios, those close to unity, describe faces that appear monotonous and flat overall. A contrast of less than unity would result in a typeface distinctly unnatural.

1.5.2 Definition of Auto Boldness

There are two methods to bold a selected area of text in a word processing environment. The first one is to select a corresponding bold font. Nevertheless, font makers do not always produce a font, and a corresponding bold version, because the production of bold version is very difficult and labor intensive. Moreover, the installation of a bold font does consume much memory space. Thus, it is better to generate a bold version from a stored unbold font on the fly, and this technique is termed as 'Auto Boldness'. The current techniques of auto boldness include Double Printing and Multi-Master Technique.

1.5.3 Auto Boldness by Double Printing

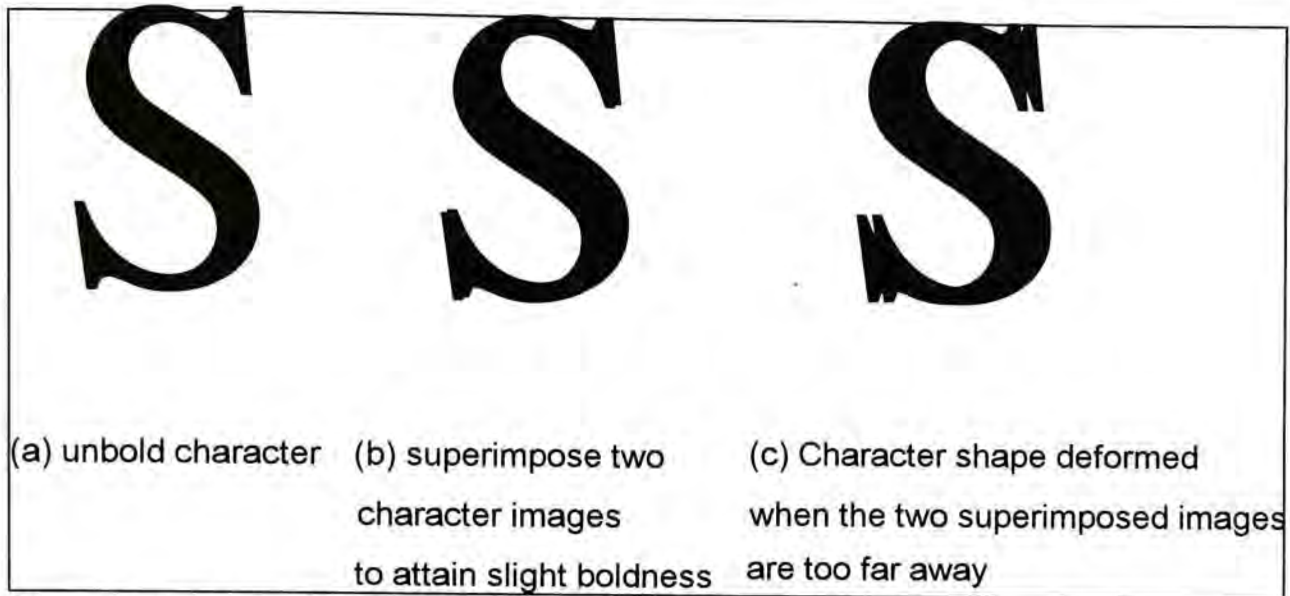
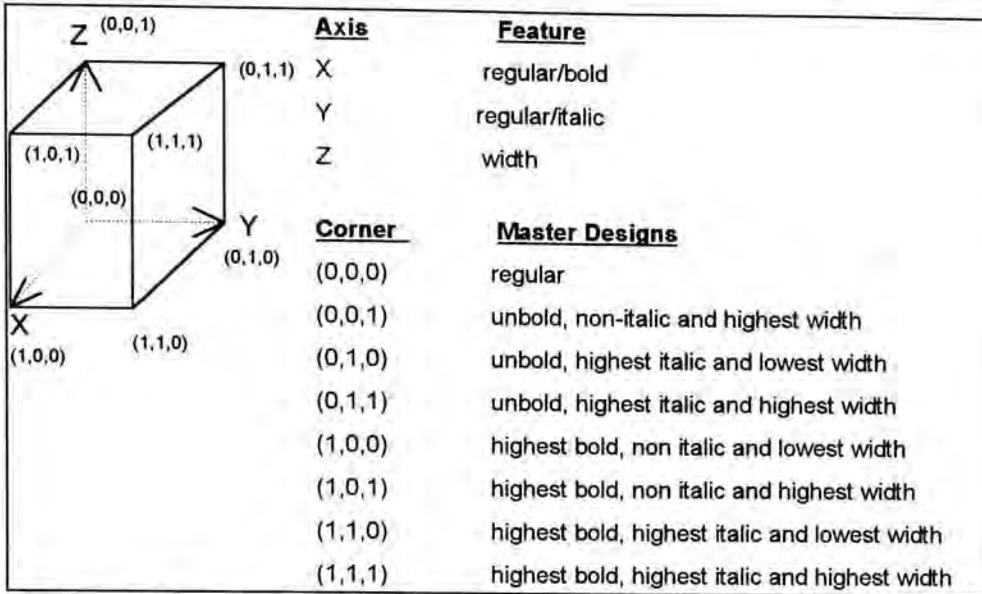


Figure 1.9: Auto Boldness by Double Printing

It is simple and general enough to work on all fonts. The idea is to double print a character with a slight horizontal displacement between the two printing positions, as in fig 1.9b. The effect is to increase the contrast space (the ratio of thick to thin stem width). Chinese font and English font have a common property that horizontal stems are thin, while vertical and inclined stems are thick. Double Printing can increase the thick stem width, but retains the thin stem width, leading to an increase of contrast. Many word processor using this technique have a dialog box called font style with a set of buttons. One of the button is called "Bold". If you press it, the box, which is called sample, will display a bold character. Thus, the option is only Bold/Non-Bold without intermediate bold level. Moreover, only a slight increase of contrast ratio can be attained, because the integrity of character shape will be affected if the distance between the two printing positions is too large, as shown in fig 1.9c. As a result, the usefulness of Double Printing is very limited.

1.5.4 Auto Boldness by Multi-Master Technique



Axis	Feature
X	regular/bold
Y	regular/italic
Z	width

Corner	Master Designs
(0,0,0)	regular
(0,0,1)	unbold, non-italic and highest width
(0,1,0)	unbold, highest italic and lowest width
(0,1,1)	unbold, highest italic and highest width
(1,0,0)	highest bold, non italic and lowest width
(1,0,1)	highest bold, non italic and highest width
(1,1,0)	highest bold, highest italic and lowest width
(1,1,1)	highest bold, highest italic and highest width

Figure 1.10: Design Axis

This technique is developed by Adobe [Adobe 92], supporting multi-level of auto boldness. Having the master design of the regular font and the master design of the corresponding highest level bold font, the intermediate level of bold font can be obtained by interpolating the two given master designs. The interpolation is actually the weighted sum of the two designs control point coordinate. Each master design has a corresponding weight which is user specified, and the weights of master designs constitute a Weight Vector, of which the dimension is equal to the number of master designs to do interpolation (2 or more), and the sum of the elements of Weight Vector equals to 1.0. As shown in fig 1.10, the case of more than two master designs is possible. Because, a font can be characterized by a number of design features, such as regular/bold, regular/italic, width, etc. Each design feature can be described quantitatively by a design axis. For example, if the design feature is regular/bold, the two extremities of the design axis are the regular master design and the highest bold level master design. In fig 1.10, the eight corners of the cube represent the eight master designs needed, and the point within the cube represents a font instance with a particular weight vector. The closer the point is to the corner, the more similar of the font instance to the master design.
Suppose the number of design axis = N. Then, number of master designs needed = 2^N.

Nevertheless, Multi-Master Technique is not applicable to Chinese Fonts. Chinese character has a very large character set. Thus, each master design, which is actually a font file stored in computer memory, will consume much memory space, about one to two

mega bytes. The memory required to store master designs is 2 mega bytes x 2^N. It is very tremendous, and unrealistic for personal computing environment. The advantages of Multi-Master Technique are that the speed to generate the font instance is very fast, and the quality of font instance can be guaranteed by making the production quality master designs.

1.6 Chinese Fonts

1.6.1 Chinese Character Sets

a. Large Character Set

In English, words can be formed by alphabets concatenation. But, this is not the case of Chinese words, because, a Chinese character is ideaographical. In other words, the shape of a Chinese character is similar to the thing denoted by the character, and a word is an indivisible unit, instead of a concatenation of alphabets. Therefore, a tremendous amount of Chinese characters have been invented to meet the daily needs of Chinese peoples over the past 5000 years history of China.

There are two main coding schemes to encode Chinese characters as double bytes coding schemes. Firstly, Big-5 Chinese Coding Scheme used in Taiwan encodes 13,053 characters including 5,401 commonly used and 7,652 less commonly used ones. Secondly, GB Coding Scheme used in Mainland China encodes 7,000 commonly used Simplified Chinese characters. The set of Chinese characters is simplified due to combining some similar characters (similar in shape or meaning) into a new simplified one.

b. Low Repetition Rate

For English, font caching can effectively reduce the time spent on rasterisation in typesetting and laser printing. However, in Chinese, the font caching strategy is not very straightforward because of the low character repetition rate. In contrast, English has a small set of commonly used words. Caching the commonly used words can reduce the need to perform the time-consuming rasterisation process.

But, it is still possible to make use of cache strategy for Chinese font, because many Chinese characters can be decomposed into a left component and a right component. The repetition rate of the components is very high. Thus, caching the components is an intelligent approach. However, it involves the automatic extraction of components.

c. Condensed Strokes

Chinese character can be disassembled as strokes. High stroke count is another feature of Chinese characters. Characters with as many as 15 strokes are not uncommon. In order to squeeze out the last drop of memory, stroke based TrueType font has come out in the recent years. The idea of stroke based TrueType font is to store a number of strokes in the font file, then character can be assembled by geometrically transforming the strokes fetched from the font file. Thus, the storage requirement of character is only the index of strokes, and the parameters to do transformations. In TrueType terminology, it is called component fonts.

1.6.2 The Subtleties of Chinese Fonts Auto Boldness

a. **hard to extract strokes**

In order to achieve a high quality of bold font, a Chinese character must firstly be disassembled as strokes. Then, the extracted strokes are identified and bolded individually. Nevertheless, strokes extraction is a subtle problem for Chinese fonts. Firstly, a same class of strokes can manifest in a Chinese character with different orientations, sizes. and locations. As shown in fig 1.11, Dot strokes in different orientations and locations can appear in a Chinese character. Thus, the algorithm to extract strokes must be independent of size, orientation and location. However, such an algorithm is very hard to develop.

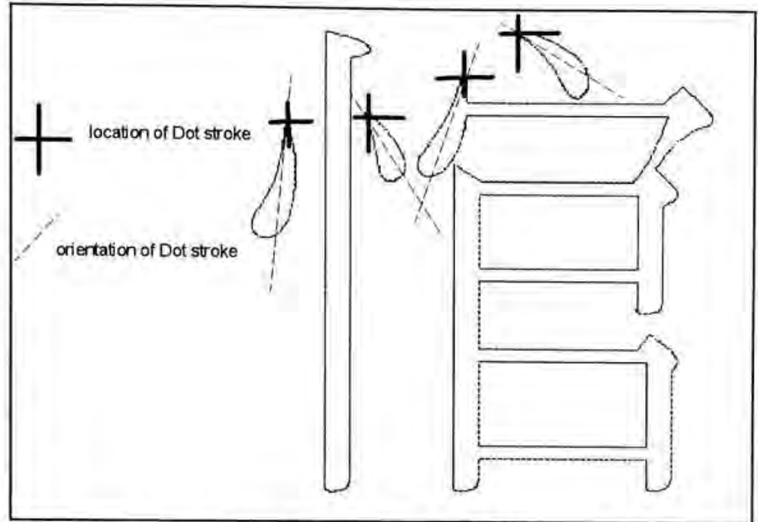


Figure 1.11: Dot Strokes of a Chinese Character

b. **hard to determine the new position of strokes interception after bold**

Strokes of a character can be intercepted with one another. In outline font, if two strokes outline are intercepted, the two strokes outline are clipped to form a single outline. This makes it more difficult for stroke extraction. Moreover, even though the intercepted strokes are all extracted successfully, it is hard to determine the new position of the interceptions after bold. As shown in fig 1.12, p is the interception of left incline stroke and right incline stroke. Viewing from left incline stroke, point p must be migrated in the direction of vector V2, but viewing from right incline stroke, point p must be migrated in the direction of vector V1. The direction of V1 and V2 contradicts to each other. Thus, the interception p must be carefully placed. If not, the curve segment near point p will become discontinuous after auto boldness.

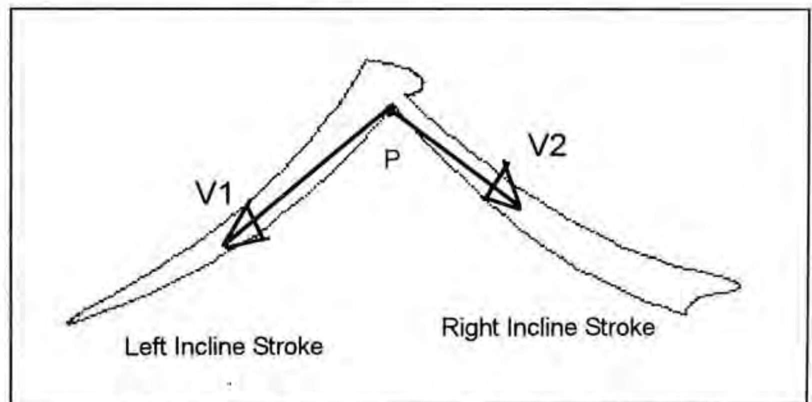


Figure 1.12: The Interception of Left Incline Stroke and Right Incline Stroke

c. **hard to bold the delicate outline of Chinese fonts**

The shape of a Chinese character is far more complex than English character. The latter is only composed of simple curves and straight line segments, but the former one is composed of serifs and curves with inflexion points and sharp edges, as shown in fig 1.13. If the serifs of a character are not carefully treated, the serifs will be deformed after auto boldness. Thus, a Chinese character can be considered as a very delicate object, and it is extremely challenging to modify a delicate outline to attain bold effect.

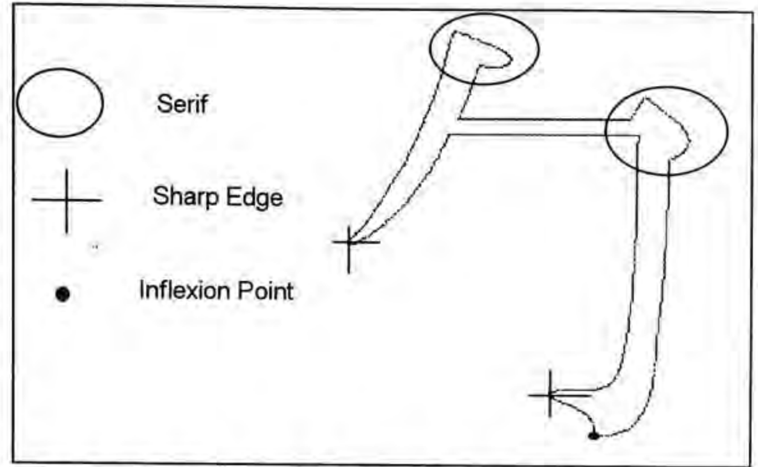


Figure 1.13: The Delicate Outline of Chinese Character

d. **hard to keep the balancing nature of Chinese character**

The design of a Chinese character is very balanced to such an extent that the center of gravity (CG) is near the center of the bounding box of the character outline. Thus, during auto boldness, the increment of weight must be equally applied throughout the whole character outline so that the CG can be kept at the center of bounding box.

e. **hard to keep the white spaces of Chinese character**

The white spaces of a Chinese character must be maintained, and if two white spaces, which are equal in their dimensions, must remain equal after auto boldness. However, some Chinese characters have so many strokes that their white spaces are very slim, and it can hardly be retained in auto boldness.

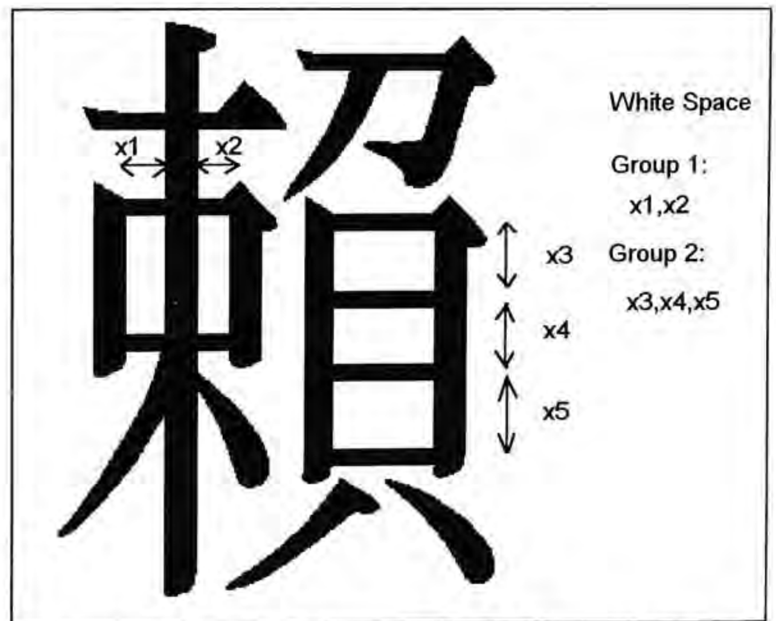


Figure 1.14: White Spaces of a Chinese Character

1.7 Project Objective

we aim at developing a new algorithm for Chinese outline fonts auto boldness which can control the stroke width of characters automatically and the stroke width can be fine tuned by the user at will. The bold version is generated "on the fly" from the stored unbold fonts, and the average time to bold a character must be short enough for real time purposes. Furthermore, good quality of bold must be ensured. Continuity, symmetries, white spaces and serifs must be kept in the bold version. There is no need to install an additional bold version to do interpolation as in Multi-Masters Technique. Thousands of bold versions with different level of boldness can be generated from a single unbold font automatically by the algorithm.

1.8 Goals

1. The input to the algorithm is only the unbold font.
2. The algorithm must be fast enough for real time environment.
3. A significant bold effect must be attained.
4. Intermediate level of boldness can be attained.
5. The integrity and continuity of the character outline must be kept.
6. The original characteristics must be preserved.
7. Serifs must be carefully handled.
8. White spaces must be kept.
9. The algorithm must be general enough to work on all fonts.
10. It is totally a software solution, and no additional hardware is required.

Chapter Two: Main Ideas of Chinese Font Auto Boldness

2.1 Prototype of Auto Boldness Driver

The generation of a good quality bold font involves much effort. As a result, it is more economical to develop a driver which is intelligent enough to generate bold version from an unbold master on the fly. In our project, a prototype driver has been built .

The outline of a Chinese character varies considerably among different fonts. Thus, we have tried our best to devise an auto bold algorithm for as many Chinese fonts as possible. It can be shown that our algorithm can be worked on Sung, Ming fonts, which have common characteristics of regularity in shape and distinct sharp points (will be described later). However, the prototype driver is domain specific to Microsoft Chinese Windows 3.0 CFSUNG font. CFSUNG is a TrueType font. Each character is described by a set of contours, and the contour is in fact a set of joined primitives, which are either straight line (two control points) or a quadratic Bezier (Three control points). Moreover, this font is non-component based. Thus, if the two stroke contours are intercepted, the two contours will be clipped to form a single contour. So, our auto boldness algorithm involves the following steps:

1. ***Locate the strokes***
2. ***Classify and fit a model to the strokes.***
   The model of a stroke has two components. The first one is the pure stroke part, and the second one is the serif part.
3. ***Apply some suitable autobold methods to modify the pure stroke part outline.***
4. ***Adjust the serif part to accommodate the modified pure stroke part.***
5. ***Smooth the whole character outline.***

2.2 Design Features of the Prototype Auto Boldness Driver

Storage Requirement:
There is no need to store additional bold information in the font data file. All the bold information can be generated in an ad hoc manner. The input of the Prototype Autoboldness Driver is only the raw data of CFSUNG.FON font data file.

Speed:
In a 486 PC, the average speed to bold a Chinese Character is less than a half of second, suitable for real time environment. The high speed process is due to an efficient heuristic search technique **(Sharp Point Classification)** to extract and classify the strokes.

Flexibility:
The bold level of the generated bold font can be tuned by user. The tuning involves adjusting the parameter **(BoldLevel)**. BoldLevel is a rational number ranging from 0.0 to 1.0, and the definition of it is:
BoldLevel = Increase in Stroke Width / Original Stroke Width (2.1)

Quality:
The outline of the resulting auto bold character is smooth and perfect, retaining its original characteristics, such as serif width and height. Furthermore, the balancing nature of the original outline is preserved.

吹 ====> 吹 吹

BoldLevel=0.0

BoldLevel=0.4

BoldLevel=0.8

BoldLevel = 1.0

見 ==========> 見 見

BoldLevel = 0.0

BoldLevel=0.4

BoldLevel=0.6

見

BoldLevel = 1.0

2.3 Data Structure and Algorithm of Auto Boldness

2.3.1 Data Structure of TrueType Character Outline

Coding	Description
Char.ContourNum	number of contours
Char.Contour[]	array of contours, size of ContourNum
Char.Contour[i].PrimitiveNum	number of primitive of ith contour
Char.Contour[i].Primitive[]	array of primitives for ith contour, size of PrimitiveNum
Char.Contour[i].Primitive[j].Typ	Type of the j th primitive of i th contour Line : Straight Line QBezier : Quadratic Bezier
Char.Contour[i].Primitive[j].Cx[k]	x coordinate of k th control point of j th Primitive of i th Contour if Typ==Line then k = 0,1 On-Curve control points if Typ==QBezier then k = 0 On-Curve control point k = 1 Off-Curve control point k = 2 On-Curve control point
Char.Contour[i].Primitive[j].Cy[k]	y coordinate of k th control point of j th Primitive of i th Contour

The outline of a character is described by a set of contours, and the contour is in fact a list of joined primitives. The primitive is either Straight Line or Quadratic Bezier. Because the primitives in the contour list are joined, the following conditions must hold:

[1. **if**(Char.Contour[i].Primitive[j].Typ==Line) **then**
Char.Contour[i].Primitive[j].Cx[1] == Char.Contour[i].Primitive[k].Cx[0]
2. **if**(Char.Contour[i].Primitive[j].Typ==QBezier) **then**
Char.Contour[i].Primitive[j].Cx[2] == Char.Contour[i].Primitive[k].Cx[0]
where k = (j + 1) **mod** Char.Contour[i].PrimitiveNum *(2.2)*]

The character outline extracted from Windows 3.0 CFSUNG.Fon font data file is in TrueType format. Thus, the outermost contour of a character is in an anti-clockwise direction, and the inner contour direction is the reverse of the outer contour direction[True 91]

2.3.2 Algorithm of Auto Boldness

Autoboldness(In_Out Char, In BoldLevel)
Char - *the input unbold character outline in TrueType format, and the output bold character outline*
BoldLevel - *a rational number to specify the bold level ranges from 0.0 to 1.0*

```
{
  Preprocessor(Char);
  for(i=0; i < Char.ContourNum; i++)
    for(j=0; j < Char.Contour[i].PrimitiveNum; j++)
      if(Char.Contour[i].Primitive[j].Typ==QBezier)&&
                IsSharpPoint(Char.Contour[i].Primitive[j]))
        if(FitModel(LeftInclineStroke,Char,i,j)==True)
            Model = LeftInclineStroke;
          else if(FitModel(RightInclineStroke,Char,i,j)==True)
            Model = RightInclineStroke;
          ....... for model fitting of other type of strokes
          if(any stroke model fits)
            PureStroke = ExtractPureStrokeOutline(Model,Char,i,j);
            Serif = ExtractSerifOutline(Model,Char,i,j);
            BoldPureStroke = BoldExtractedPureStrokeOutline(Model,PureStroke,BoldLevel);
            BoldStroke = SerifAccomondation(BoldPureStroke,Serif);
  BoldVerticalStrokes(Char,BoldLevel);
  Smoothing(Char);
}
```

2.3.3 Algorithm Description

Step 1: The first step of the autoboldness algorithm is to preprocess the character outline to find the attributes of the primitives. The attributes include concavity(either concave or convex) for quadratic Bezier, the inclined angle C (values from $0^{0} to 360^{o}$), and the curve length L. If the primitive is straight line, C is the inclined angle of the line joining the first on-curve control point to the second on-curve control point. If the primitive is Quadratic Bezier, C is the inclined angle of the line joining the first on-curve control point to the third on-curve control point.

Step 2: Then, the for loop with i,j integer counters is to locate, classify, extract and bold the strokes of a character. To locate a stroke, a simple heuristic called "**Sharp Point Searching**" is used.

Def 2.3: Sharp point is defined as a point on the contour with a steep change in first derivative, or a point on the contour with high curvature.

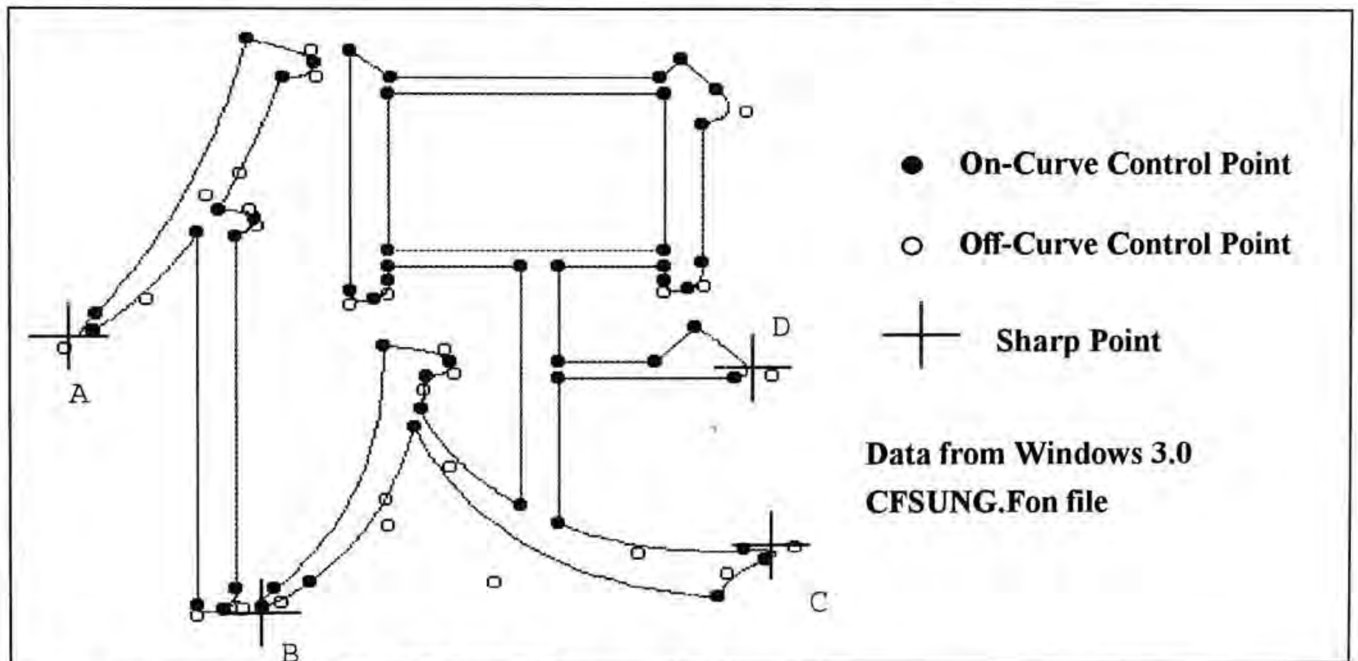


Figure 2.1: Sharp Points of a Chinese Character

It is found that each curve stroke must contain one or more (mostly one) sharp point(s). Thus, a stroke can be located by searching the locations of sharp points. In figure 2.1, Sharp points A,B belong to a left incline stroke. Sharp point C belongs to a right incline stroke. Sharp point D does not belong to any curve stroke, but a point on the serif of a horizontal stroke. In figure 2.1, a sharp point can be found easily without using the complicated calculus formula, because a sharp point is always on a quadratic Bezier which has a small curve length(attribute L) and three control points can form an acute triangle.

Step 3. The existence of a sharp point implies a high probability of the existence of a curve stroke. Thus, the sharp point is further tried to be fitted to some curve stroke models, the nested if(FitModel(...)) else ... statement. The function FitModel(LeftInclineStroke,Char,i,j) will return true, only if the sharp point on primitive j of the i th contour fits the model of Left Incline Stroke. In figure 2.1, Sharp points A,B fit the model of Left Incline Stroke, and sharp point C fits the model of RightIncline Stroke.

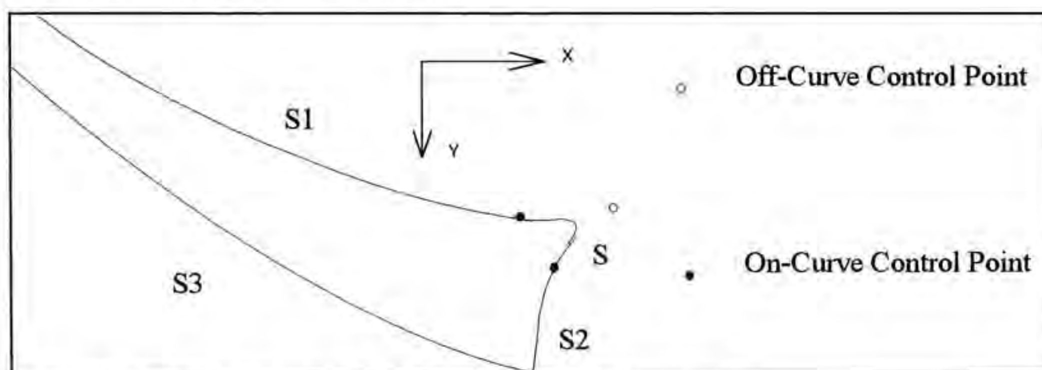


Figure 2.2: Model of a RightIncline Stroke

The example of Right Incline Stroke Model is shown in figure 2.2. S1,S2,S3 form the continuous curve segment on the contour of a right incline stroke. S is a quadratic Bezier of which a sharp point exists. So, the model of Right Incline Stroke can be defined as a set of heuristic rules.

1. S is quadratic bezier **AND** a sharp point exists.
2. S1 is the next continuous curve segment of S in anti-clockwise direction.
3. S2 is the next continuous curve segment of S in clockwise direction.
4. S3 is the next continuous curve segment of S2 in clockwise direction.
5. S1 is concave **AND** S2 is concave **AND** S3 is convex
6. (Length(S1) > Length(S2)) **AND** (Length(S3) > Length(S2))
7. S1 is decreasing in y-coordinate in anti-clockwise direction **AND**
   S2 is decreasing in y-coordinate in anti-clockwise direction **AND**
   S3 is increasing in y-coordinate in anti-clockwise direction

The details of Stroke Model Matching is presented in chapter 4, and stroke classification is presented in Appendix One.

Step 4. A stroke outline is composed of two parts: One is pure stroke outline and other is the serif outline. Therefore, after a stroke has been classified, the stroke will be segmented as pure stroke outline (function **ExtractPureStrokeOutline**) and serif outline (function **ExtractSerifOutline**). Then, the pure stroke outline will be bolded automatically(function **BoldExtractedPureStrokeOutline**).

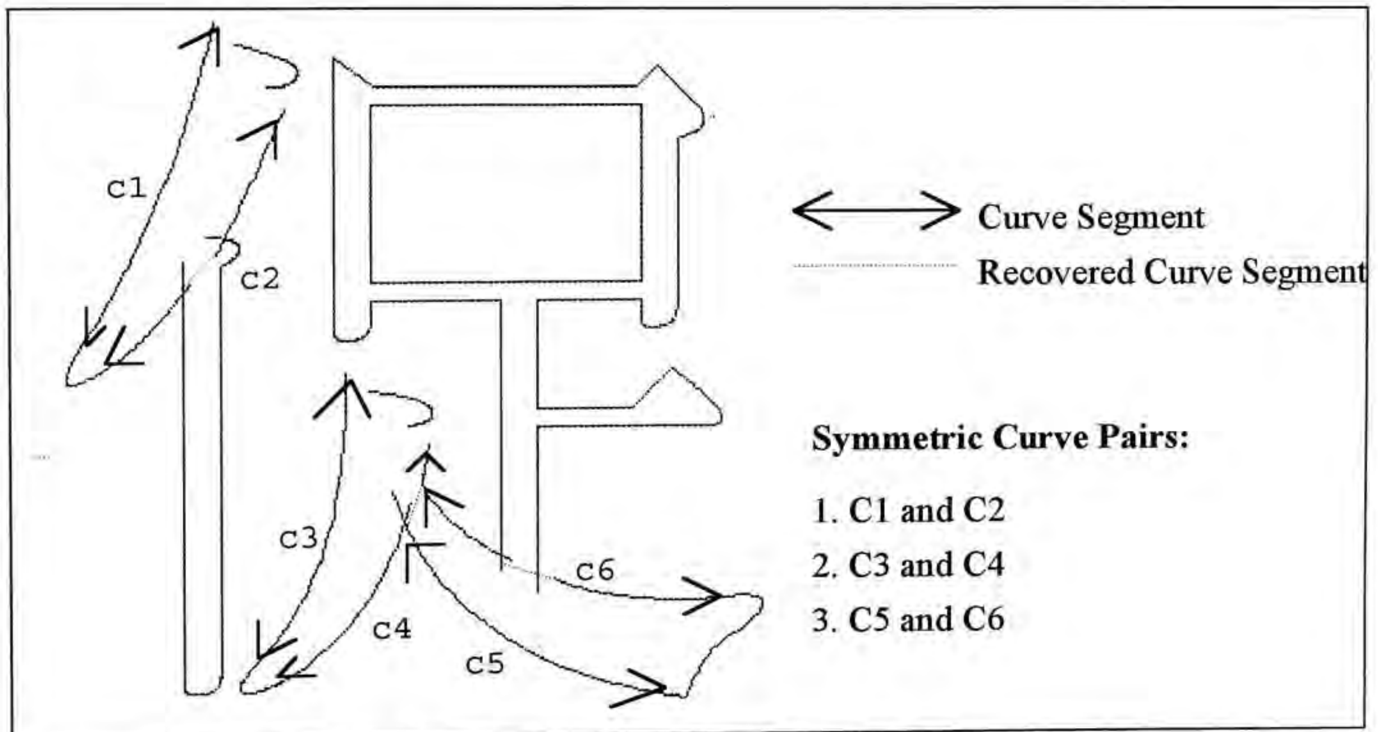


Figure 2.3: Strokes Outline Bold without Serif Handling

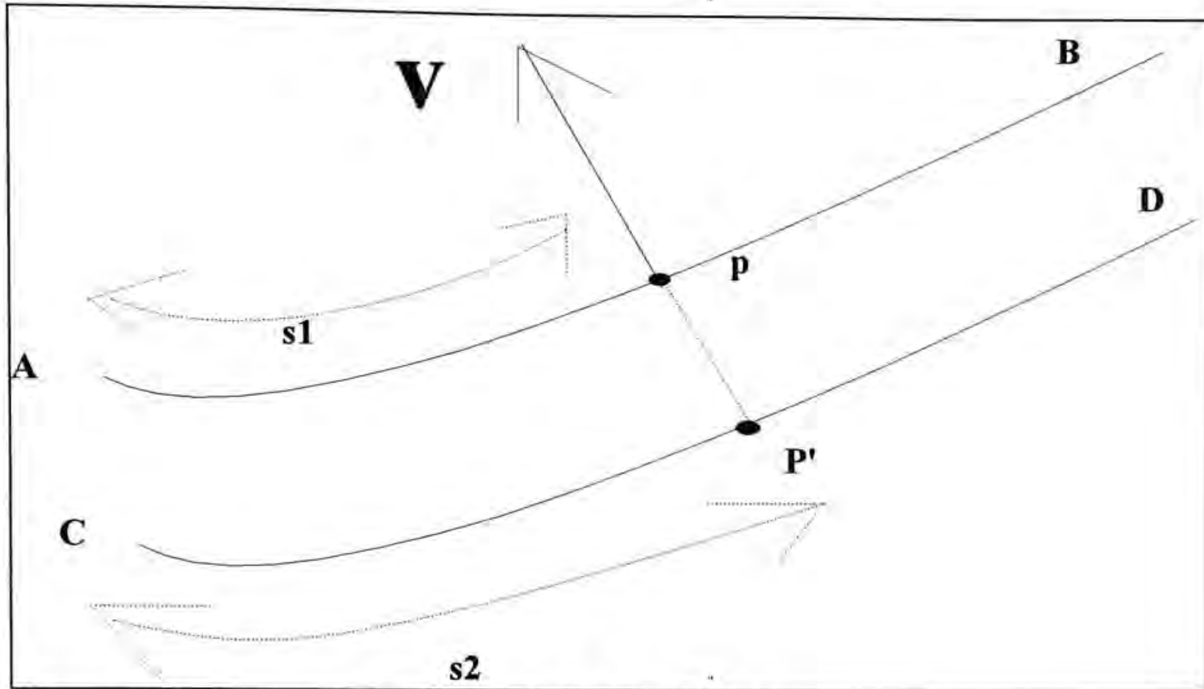


Figure 2.4: Symmetric Curves Pair Modification

Refers to figure 2.3, the pure stroke outlines have been modified, but the serif outlines have yet to be modified, resulting in the discrepancy between serif outline and pure stroke outline. The pure stroke is modified by applying the method of Symmetric Bold, to be discussed later. But, before applying this method, the missing curve segment of pure stroke outline, due to stroke interception, must be recovered, the dotted line in figure 2.3. The algorithm to recover the missing curve segment will be presented in chapter 3. The idea of Symmetric Bold is that the pure stroke outline is always composed of a pair of symmetric curves [Percept 92], and there is a simple formula to modify a symmetric curves pair to make bold effect. Refers to fig 2.4, curve AB and curve CD are symmetric. P is a on-curve control point at curve AB. In order to attain bold effect, the on-curve control point at P must be migrated. A mapping from points on the curve AB to points on the curve CD can be established. P is mapped to P' on the curve CD.

Thus, $\frac{S1}{C1} = \frac{S2}{C2}$. where S1 and S2 are the length of curves AP and CP' respectively.
C1, C2 are the length of curves AB and CD respectively.

The migration of point P can be characterised by a displacement vector.

The displacement vector of P is $V = \frac{BoldLevel}{2} * \begin{pmatrix} x-h \\ y-k \end{pmatrix}$ (2.4)

where P = (x,y) and P' = (h,k)

Formula 2.4 can only handle on-curve control points. Thus, it can not be applied directly to quadratic Bezier primitives because of the off-curve control point of quadratic bezier. However, in chapter 3, we show that formula 2.4 can be modified to deal with Quadratic Bezier nicely.

Step 5. Serif outline must be modified to accommodate the modified pure stroke outline, the function **SerifAccomondation**.The idea is to find the model of serif firstly, and then some suitable geometrical transformations are used to modify the serif outline. Three objectives must be attained: 1. The serif outline must be joined to the pure stroke outline. 2. The shape of serif must be unchanged. 3. The serif height, width must be preserved. The detailed algorithm will be shown in chapter 3.

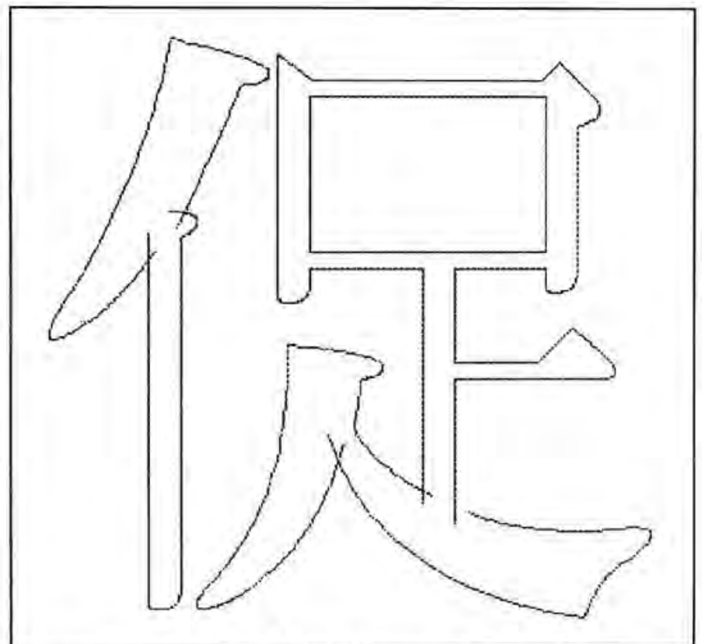


Figure 2.5: Bold Outline with Serif Handling

Step 6. Until now, only the curve strokes have been bolded. In order to increase the contrast ratio (thick stem width/thin stem width), only vertical strokes are needed to bold. The procedure **BoldVerticalStrokes** is used to bold vertical strokes, and the idea is similar to curve strokes autoboldness. Vertical strokes can also be divided into pure stroke outline and serif outline.

Step 7. Because strokes are bolded individually, the curve segment at the interception of strokes will have a gap. Thus, a procedure **Smoothing** is used to eliminate the gaps. Now, the outline of character can be input to rasterizer. However, the strokes, which are not intercepted originally, will be intercepted after autoboldness. The rasterizer, using alternate counting algorithm, will fail to rasterise the strokes interception part, but the rasteriser, using wind counting algorithm can rasterise the bolded character successfully as shown in figure 2.6.

Figure 2.6: Character Bold with BoldLevel=1.0

Definition 2.5: Alternate counting algorithm

The system fills the area between odd-numbered and even-numbered curve segments on each scan line. That is, the system fills the area between the first and second side, between the third and fourth side, and so on.

Definition 2.6: Wind counting algorithm

The system uses the direction in which a figure was drawn to determine whether to fill an area. Each contour is drawn in either a clockwise or anti-clockwise direction. Whenever an imaginary line drawn from an enclosed area to the outside of a figure passes through a clockwise contour, a count is incremented; when the line passes through an anti-clockwise contour, the count is decremented. The area is filled if the count is non zero when the line reaches the outside of the figure.

2.4 Component Font Autoboldness

The algorithm will fail, if the sharp point of a curve stroke is intercepted by other strokes, resulting in sharp point missing. The algorithm can't locate a stroke with a missing sharp point, and such a stroke will be left unbold.

Autoboldness to full set of Chinese characters is still feasible in stroke based font of Windows 3.1, refers to fig 2.7. The contours of a stroke based character will be intercepted, because each stroke is represented as a closed contour, and the interception area of strokes is left unclipped. The wind counting algorithm is used to rasterize a stroke base character [True 91]. A non-stroke based character(CFSUNG Chinese font) is totally different, the contours of a character do not intercept each other, because the contour of strokes are clipped against each other. The alternate counting algorithm is used to rasterize a non-stroke based Character. As a result, some sharp points of curve strokes will be missed in the case of non-stroke based character, but this will not be the case for a stroke based character.

Thus, it is certainly possible to locate all strokes of a character in the case of stroke based font. Furthermore, there is no need to identify stroke, and the speed of autoboldness can be improved.

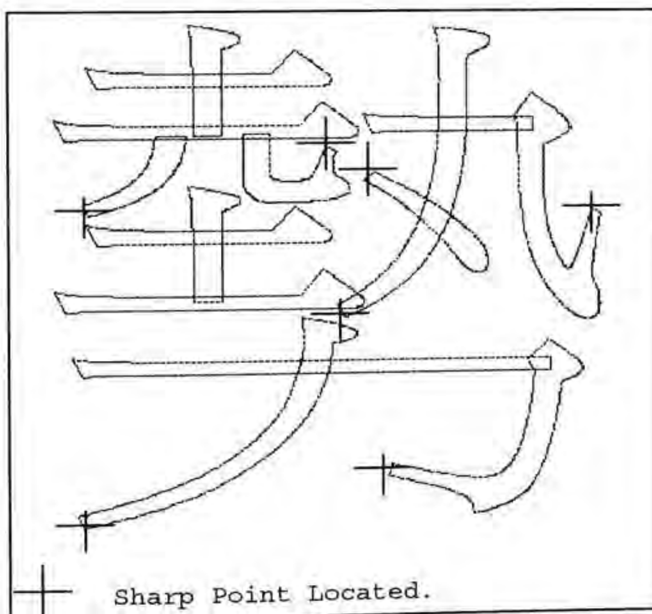


Figure 2.7: Sample of a Stroke Based Character

Chapter Three: Language of Auto Boldness

3.1 Enhancements of TrueType Engine to support Auto boldness

It is totally impossible to generalize the auto bold algorithm for all Chinese fonts. To tailor the auto bold algorithm for each font is very labor intensive. Moreover, this approach contradicts principles in Software Engineering, such as maintainability, reusability and flexibility. Pragmatically speaking, we must make a trade off between generalization and tailoring for each fonts. The idea is to extract the commonalities to build a common module (a library) so that this module is reusable in the development of the auto bold driver for any font. Thus, the later effort is only spent on the development of the font domain dependent module (a program to call the library).

It is not a new idea because it is already a practice in TrueType font hinting [True91]. The common module of hinting is a set of TrueType hinting instructions and the TrueType engine to interpret the instructions. The hinting instruction is an assembly like programming language to hint characters by control points migration. During the interpretation of the font program, the TrueType engine will interact with a byte stack by pushing and popping the bytes from the stack. The byte stack acts as a temporary storage for the TrueType engine to store the subroutines calling parameters and the local variables.

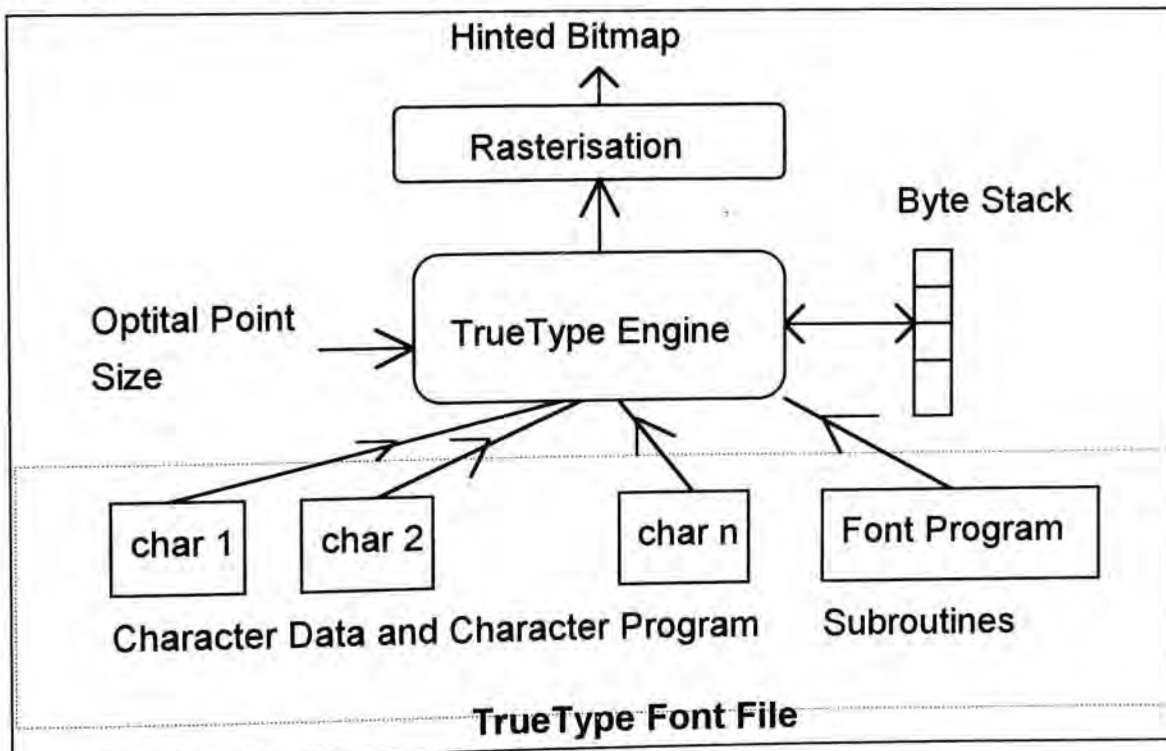
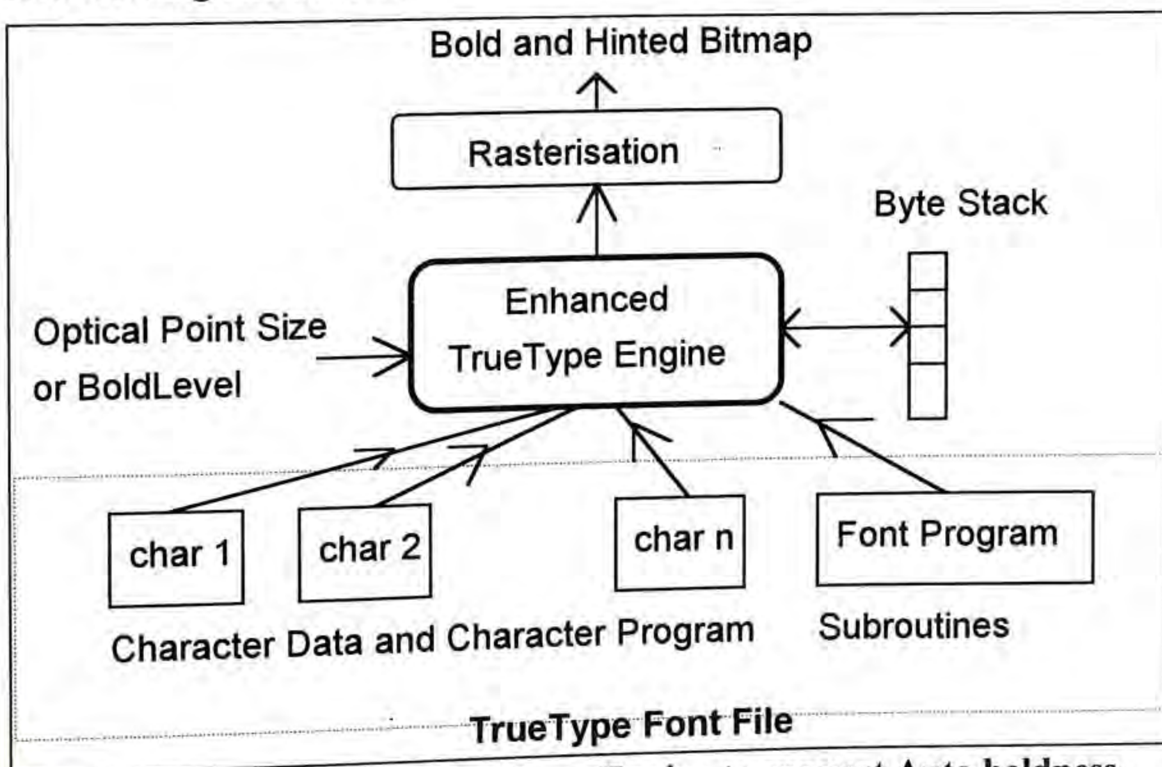


Figure 3.1: TrueType Hinting

Moreover, each time the optical point size is changed by user, the TrueType engine will be invoked to hint the character outline again, and the updated character outline will be subsequently re-rasterised. The TrueType font file can be segmented as a number of tables. The font program table (fpgm) defines the interfaces and implementations of the hinting subroutines. The glyph table (glyf) stores the outline control points of the characters, and the hinting programs for each character. The hinting program for a character is a segment of codes to call the subroutines in font program table. Executing it, the character outline will be hinted. It is a common practice to define the subroutines to hint the horizontal strokes, vertical strokes and curve strokes in the fpgm table, and the mission of the character programs in glyf table is to call the appropriate stroke hinting subroutines in fpgm table. The fpgm table is generated manually, but the character programs in glyf table can be generated manually or automatically. The auto-generation of hints can be implemented by general purpose programming language, such as C, Pascal, and it includes the phases: 1. read the character outline (control points coordinates) from the font file. 2. do shape analysis to extract the strokes. 3. generate a segment of codes to call the appropriate subroutines in fpgm table to hint the strokes. 4. merge the segment of codes with the font file.

In order to realize auto boldness for TrueType font, the TrueType engine must be enhanced. Firstly, the TrueType instructions set must be expanded to allow auto boldness. Secondly, the TrueType Engine must be invoked when the optical point size or the parameter BoldLevel is changed by user, as shown in fig 3.2.

The additional TrueType instructions include the auto bold instructions and the serif handling instructions.



3.2 Symmetric Bold Instruction

Purpose:

To bold a symmetric curves pair according to the parameter BoldLevel. When BoldLevel=0.0, the outline is unbold; When BoldLevel=1.0 the outline has the highest level of boldness; When BoldLevel is between 0.0 to 1.0, the outline has an intermediate level of boldness.

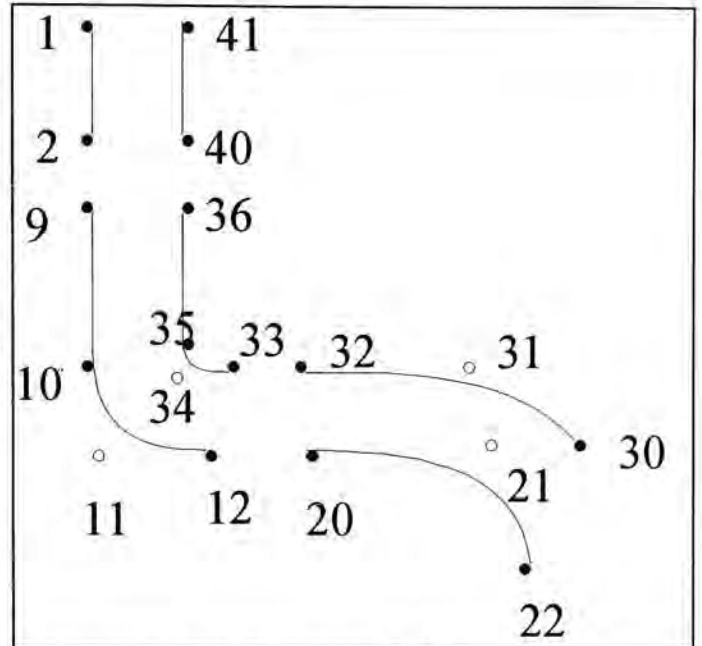


Figure 3.3: the extracted Pure Stroke Outline

Input Parameters:

1. Symmetric Curves Pair

a. an ordered list of point number for the first curve segment.
In fig 3.3, curve1 = [30,31,32,33,34,35,36,40,41]

b. an ordered list of point numbers for the second curve segment.
In fig 3.3, curve2 = [22,21,20,12,11,10,9,2,1]

c. the order of point numbers for curve1 is the reverse of the order of point numbers for curve2

2. BoldLevel: level of boldness, a rational number from 0.0 to 1.0

Algorithm Description:

1. Missing Part Recovery

Before applying symmetric bold, the missing curve parts, resulted from strokes interception, must be recovered. In fig 3.3, the curve segment from point 2 to 9, point 12 to 20, point 32 to 33, and point 36 to 40 are missed. The idea is to use a quadratic bezier to fill the gap between the existing curve segments. The on-curve control points of the quadratic bezier can be set to the end

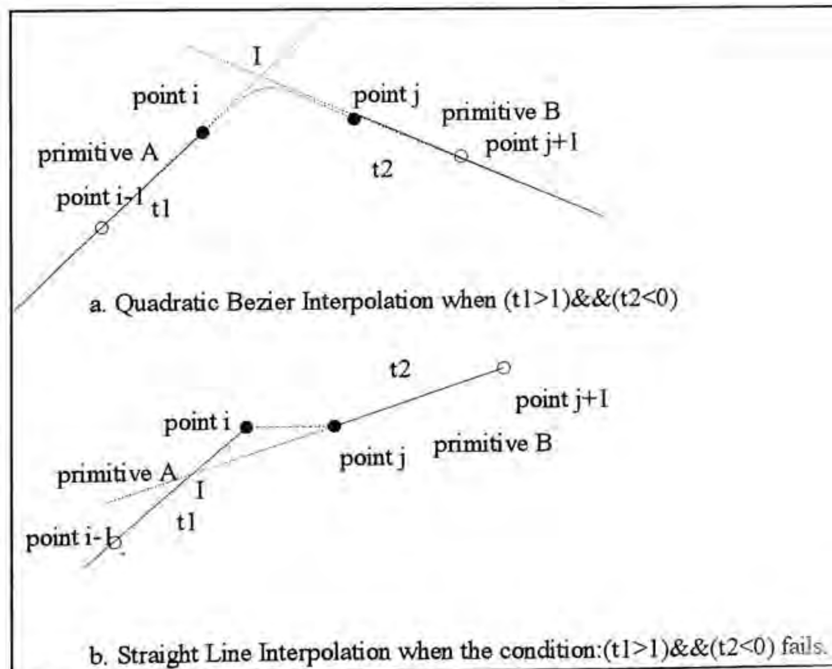


Figure 3.4: Recovery of Missing Curve Segment

point of the existing curve segment, and the off curve control point can be set to the interception point of the two tangent lines of the end points.

Ref. to fig 3.4a, the curve segment between primitive A and B is missed. Point i is the on-curve point of primitive A. If primitive A is a straight line primitive, point i-1 is also on-curve. If primitive A is a quadratic bezier, point i-1 is off-curve. Line A is the line joining the point i-1 and point i, the equation of it can be

expressed parametrically as $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix} + t1 \begin{pmatrix} x_i - x_{i-1} \\ y_i - y_{i-1} \end{pmatrix}$ [1]

Line B is the line joining the point j and point j+1, the equation of it can be

expressed parametrically as $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_j \\ y_j \end{pmatrix} + t2 \begin{pmatrix} x_{j+1} - x_j \\ y_{j+1} - y_j \end{pmatrix}$ [2]

point I is the interception point of Line A and B, and can be calculated by solving t1 and t2 in the simultaneous equation

$$\begin{pmatrix} x_{i-1} \\ y_{i-1} \end{pmatrix} + t1 \begin{pmatrix} x_i - x_{i-1} \\ y_i - y_{i-1} \end{pmatrix} = \begin{pmatrix} x_j \\ y_j \end{pmatrix} + t2 \begin{pmatrix} x_{j+1} - x_j \\ y_{j+1} - y_j \end{pmatrix} \quad [3]$$

obtaining by eliminating the couple $\begin{pmatrix} x \\ y \end{pmatrix}$ from equations 1 and 2

Suppose the solution is t1 = t1' and t2 = t2'

Then, the algorithm to recover the missing part between the primitive A and the primitive B in fig 3.4 is:

1. Solve for t1 and t2 from the simultaneous equation 3 such that t1 = t1' and t2 = t2'

2. if (t1>1)&&(t2<0) then

> ***calculate the interception point I of Line A and Line B by substituting t1' into equation 1;***
> ***use a quadratic bezier to fill the gap between primitive A and B such that first on-curve point is point i, second off-curve point is I, and . third on-curve point is point j (as shown in fig 3.4a)***

else

> ***use a quadratic bezier to fill the gap between primitive A and B such that first on-curve point is point i, second off-curve point is the mid-point of point i and point j, and third on-curve point is point j. (as shown in fig 3.4b).***

2. Apply Symmetric Bold Method

The stroke of a Chinese character can be decomposed as two entities - pure stroke outline and serif, and the pure stroke outline can be further decomposed as sharp point, symmetric curves pair and remaining segments. Viewing the appearance of a stroke, symmetric curves pair is more important than the serif and remaining part.

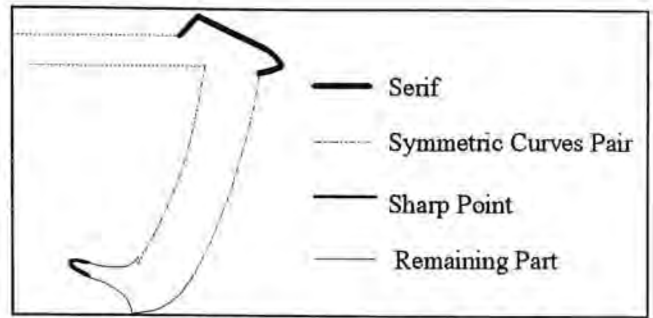


Figure 3.5: Segmentation of a Left Hoke

Thus, in auto boldness, the symmetric curves pair is processed firstly. Then, the serif and remaining part must be modified to accommodate the modified symmetric curves pair. If the symmetric characteristic of a stroke can be preserved in auto boldness, many distinct advantages will follow. Firstly, the CG. of a stroke can be preserved. Secondly, a uniform bold effect can be achieved in the resulting character outline. Thirdly and most importantly, if the original character outline is hinted, the resulting character outline will also be hinted. The Symmetric Bold Method is the most satisfactory method to bold a symmetric curves pair.

As shown in fig 3.6, the curve AB and curve CD are symmetric to each other. Intuitively, to bold the curves pair, curve AB control points must be migrated upward, and curve CD control points must be migrated downwards. Mathematically, the migration of control points can be characterized by displacement vectors. Displacement vector for a control point defines the direction and the distance of migration.

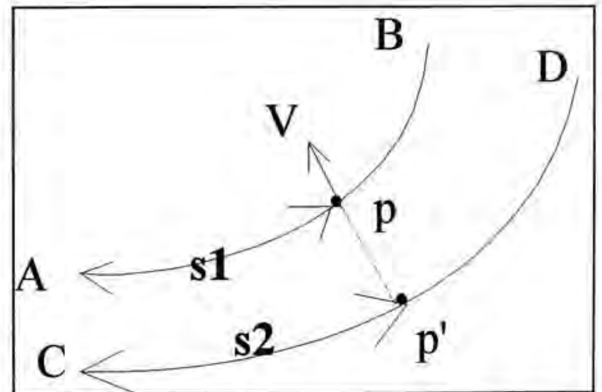


Figure 3.6: Symmetric Bold Method

P is any on-curve control point of curve AB. A one-one mapping (Fab) from the points of curve AB to the points of curve CD can be constructed.

Thus, P' = Fab(P) where P' is a point of curve CD.

such that, $\frac{s1}{CurveAB} = \frac{s2}{CurveCD}$ where s1,s2 are the length of curves AP,CP'.

CurveAB,CurveCD are the length of curves AB,CD.

Then, the displacement vector of point p is

$$v = \frac{BoldLevel}{2} * \begin{pmatrix} x-h \\ y-k \end{pmatrix} \quad \textbf{(3.1)}$$

where p = (x,y) and p' = (h,k).

Formula 3.1 can only be applied to an on-curve point. Therefore, formula 3.1 can be applied directly to a straight line primitives. For a quadratic Bezier, some trick must be used. The idea is that suppose a quadratic bezier primitive has control points

(x0,y0) : first on-curve point
(x1,y1) : second off-curve point
(x2,y2) : third on-curve point

Quadratic bezier can be expressed parametrically as:

$x(t) = (1-t)^2 x0 + 2t(1-t)x1 + t^2 x2$ [1]

$y(t) = (1-t)^2 y0 + 2t(1-t)y1 + t^2 y2$ [2]

where $t \in [0,1]$

(h,k) is the point on the primitive such that t = 0.5

so that:

$h = 0.5^2 x0 + 2 * 0.5^2 x1 + 0.5^2 x2$ [3]

$k = 0.5^2 y0 + 2 * 0.5^2 y1 + 0.5^2 y2$ [4]

By formula 3.1, calculate the displacement vectors v1,v2,v3 for the on-curve point (x0,y0), (h,k) and (x2,y2) respectively. Then, update the points (x0,y0), (h,k) and (x2,y2) by v1,v2,v3 respectively.

$$\begin{pmatrix} x0' \\ y0' \end{pmatrix} = \begin{pmatrix} x0 \\ y0 \end{pmatrix} + \begin{pmatrix} v1_x \\ v1_y \end{pmatrix}, \quad \begin{pmatrix} h' \\ k' \end{pmatrix} = \begin{pmatrix} h \\ k \end{pmatrix} + \begin{pmatrix} v2_x \\ v2_y \end{pmatrix}, \quad \begin{pmatrix} x2' \\ y2' \end{pmatrix} = \begin{pmatrix} x2 \\ y2 \end{pmatrix} + \begin{pmatrix} v3_x \\ v3_y \end{pmatrix}$$

Thus, the new quadratic bezier primitive satisfy the following conditions:

1] first on-curve point (x0',y0')
2] third on-curve point (x2',y2')
3] on-curve point (h',k') with t = 0.5

Then, the new quadratic bezier off-curve point (x1',y1') can be found by using equation 1 & equation 2.

$h' = 0.5^2 x0' + 2 * 0.5^2 x1' + 0.5^2 x2'$

$k' = 0.5^2 y0' + 2 * 0.5^2 y1' + 0.5^2 y2'$

Simplifying, we get

$$x1' = \frac{4h' - x0' - x2'}{2}, \quad y1' = \frac{4k' - y0' - y2'}{2} \quad \textbf{(3.2)}$$

By using formula 3.2, the off-curve point of a quadratic bezier can be handled. However, there is still a limitation. If the parameter BoldLevel is large, the above method cannot be applied. Thus, we can say mathematically that the above method is valid when $BoldLevel \longrightarrow 0$

3. Retaining the Geometrical Properties of Primitives

As straight line primitive can be classified as horizontal, vertical or slant. If a straight line primitive in the original character outline is horizontal(or vertical), it must remain horizontal (or vertical) in the bold character outline. Thus, if a straight line primitive has control points (x0,y0) and (x1,y1), the pseudo code to modify a straight line primitive to attain a bold effect is:

1] By formula 3.1, calculate the displacement vectors v1,v2 for (x0,y0) and (x1,y1) respectively.
2] update (x0,y0) and (x1,y1) by v1 and v2 respectively

$$\begin{pmatrix} x0' \\ y0' \end{pmatrix} = \begin{pmatrix} x0 \\ y0 \end{pmatrix} + \begin{pmatrix} v1_x \\ v1_y \end{pmatrix}, \quad \begin{pmatrix} x1' \\ y1' \end{pmatrix} = \begin{pmatrix} x1 \\ y1 \end{pmatrix} + \begin{pmatrix} v2_x \\ v2_y \end{pmatrix}$$

3] if ((x0,y0) and (x1,y1) is horizontal) {

$$\textit{set } y = \frac{y0' + y1'}{2}$$

$$\begin{pmatrix} x0' \\ y0' \end{pmatrix} = \begin{pmatrix} x0' \\ y \end{pmatrix} \quad \& \quad \begin{pmatrix} x1' \\ y1' \end{pmatrix} = \begin{pmatrix} x1' \\ y \end{pmatrix}$$

4] if((x0,y0) and (x1,y1) is vertical) {

$$\textit{set } x = \frac{x0' + x1'}{2}$$

$$\begin{pmatrix} x0' \\ y0' \end{pmatrix} = \begin{pmatrix} x \\ y0' \end{pmatrix} \quad \& \quad \begin{pmatrix} x1' \\ y1' \end{pmatrix} = \begin{pmatrix} x \\ y1' \end{pmatrix}$$

4. Pseudo Code of Symmetric Bold with small BoldLevel

```
SymmetricSmallBold(List1,List2,BoldLevel)
input : List1 is an ordered list of point numbers for the first curve segment.
        List2 is an ordered list of point numbers for the second curve segment.
        curve segments List1 and List2 are symmetric, and
        order of List1 is the reverse of the order of List2.
        BoldLevel is a small level of boldness, such that it is a rational number between 0 to 0.2
{
  /* Recover the missing curve segments of List1 */
  p = null point;
  for(i=0; i < |List1|; i++) {  /* |List1| means the cardinality of the order list List1 */
    q = point i of List1;
    if((both p and q are on-curve points) {
      if(||p,q|| != 0) {   /* ||p,q|| is the distance between p and q */
        p1 = the previous point of p in List1;
        q1 = the next point of q in List1;
        Solve the simultaneous equations
```

$$\begin{pmatrix} p1.x \\ p1.y \end{pmatrix} + t1\begin{pmatrix} p.x - p1.x \\ p.y - p1.y \end{pmatrix} = \begin{pmatrix} q.x \\ q.y \end{pmatrix} + t2\begin{pmatrix} q1.x - q.x \\ q1.y - q.y \end{pmatrix}$$

```
        so that t1 = t1' and t2 = t2'
        if((t1>1)&&(t2<0)) {
          I = Interception of Line p1,p and Line q,q1;
          add I as a virtual off-curve point between p and q;
        } else {
          M = mid-point of p and q;
          add M as a virtual off-curve point between p and q;
        };
      };
    };
    p = q;
  };
  /* idea to recover missing curve segments for List2 is the same for List1 */
  ..................
```

Length1 = curve length of List1; Length2 = curve length of List2;
/* apply symmetric bold method to the curve segment of List1 */
for(i=0; i < |List1|; i++) {
p = i th point of List1;
if(p is on-curve) {
s1 = curve length from 0 th point to i th point of List1;
ratio1 = s1/Length1;
s2 = ratio1*Length2;
q is a point on the curve of List2, such that curve length from 0 th point of List2 to q = s2;

find the displacement vector for p, $v = \frac{\text{BoldLevel}}{2}\left\langle \begin{matrix} p.x - q.x \\ p.y - q.y \end{matrix} \right\rangle$;

update p by v, p = p + v;
} else if(p is off-curve) {
p0 = (i-1) th on-curve point of List1; p1 = (i+1) th on-curve point of List1;
z is a point on the quadratic bezier with control points (p0,p,p1) such that

$$z.x = 0.5^2(p0.x + 2p.x + p1.x);$$

$$z.y = 0.5^2(p0.y + 2p.y + p1.y);$$

s1 = curve length from 0 th point to z of List1;
s2 = curve length from 0 th point to p1 of List1;
ratio1 = s1/Length1; ratio2 = s2/Length1;
t1 = ratio1*Length2; t2 = ratio2*Length2;
m is a point on the curve of List2,
such that curve length from 0 th point of List2 to m = t1;
n is a point on the curve of List2,
such that curve length from 0 th point of List2 to n = t2;

find the displacement vector for z, $v1 = \frac{BoldLevel}{2}\left\langle \begin{matrix} z.x - m.x \\ z.y - m.y \end{matrix} \right\rangle$;

find the displacement vector for p1, $v2 = \frac{BoldLevel}{2}\left\langle \begin{matrix} p1.x - n.x \\ p1.y - n.y \end{matrix} \right\rangle$;

update z by v1 so that z = z + v; update p1 by v2 so that p1=p1+v2;

$$p.x = \frac{4z.x - p0.x - p1.x}{2}; \quad p.y = \frac{4z.y - p0.y - p1.y}{2};$$

i++;
};
};
};
/* symmetric bold method for curve segment of List2 is the same as List1 */
....

```
/* keep the geometrical properties of primitives in List1*/
for(i=0; i < |List1|; i++) {
  p0 = ((i-2)>=0)?(i-2) th point of List1: Null point;
  p1 = ((i-1)>=0)?(i-1) th point of List1: Null point;
  p2 =  i th point of List1;
  if(both p1 and p2 are on-curve point) {
    if(Line p1 and p2 is a horizontal line originally) {
      if((p0 is on-curve)&&(Line p0 and p1 is a horizontal line originally) {
        y = p0.y;
        p1.y = y; p2.y = y;
      } else {
        y = (p1.y + p2.y) / 2;
        p1.y = y; p2.y = y;
      };
    } else if(Line p1 and p2 is a vertical line originally) {
      if((p0 is on-curve)&&(Line p0 and p1 is a vertical line originally) {
        x = p0.x;
        p1.x = x; p2.x = x;
      } else {
        x = (p1.x + p2.x) / 2;
        p1.x = x; p2.x = x;
      };
    };
  };
};
/* algorithm to keep the geometrical properties of primitives in List2 is the same as List1 */
.............
}
```

5. Pseudo Code of Symmetric Bold with High Bold Level

The algorithm SymmetricSmallBold can only be applied when the level of boldness is small ($BoldLevel \in [0, 0.2]$). For $BoldLevel \in [0, 1.0]$, SymmetricSmallBold with BoldLevel=0.1 must be used iteratively to the symmetric curves pair until a high level of boldness is attained. Let the number of iterations be m. Then, $(1.0 + 0.1)^x = (1.0 + BoldLevel)$,

$$\Longrightarrow x = \frac{\log(1.0 + BoldLevel)}{\log(1.1)} \qquad (3.3)$$

$$\Longrightarrow m = \text{trunc}(x) \qquad (3.4)$$

Owning to the truncation error, one more step of SymmetricSmallBold with BoldLevel=b is necessary after the iteration steps.

Thus, $(1.0 + 0.1)^m (1.0 + b) = 1.0 + BoldLevel$

$$\Longrightarrow b = \frac{1.0 + BoldLevel}{1.1^m} - 1.0 \qquad (3.5)$$

```
SymmetricBold(List1,List2,BoldLevel)
input : List1 is an ordered list of point numbers for the first curve segment.
         List2 is an ordered list of point numbers for the second curve segment.
        curve segments List1 and List2 are symmetric, and
        order of List1 is the reverse of the order of List2.
        BoldLevel is a level of boldness, such that it is a rational number between 0 to 1.0
{
  x = log(BoldLevel+1.0)/log(1.1);
  m = trunc(x);
  b = (BoldLevel+1.0)/(1.1**m) - 1.0;
  for(i=0; i < m; i++)
      SymmetricSmallBold(List1,List2,0.1);
  SymmetricSmallBold(List1,List2,b);

}
```

3.3 Rotate Bold Instruction

Purpose:

The Symmetric bold method is quite expensive to implement. However, in some cases, some other methods, which are less expensive, can be used to bold the outline.

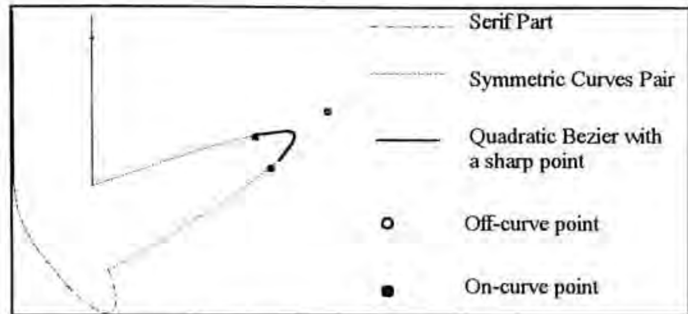


Figure 3.7: Segmentation of an Upper Incline Stroke

The Rotate Bold method is a special case of the Symmetric Bold method where the symmetric curves pair approache to a pair of straight lines. As shown in fig 3.7, the pair of symmetric curves is very similar to a pair of straight lines. The algorithm of rotate bold method is much less complex than that of symmetric bold method because it is not necessary to do missing part recovery, and it is a non-iterative algorithm.

Input Parameters:

1. Symmetric Curves Pair
   Similar to symmetric bold method
2. BoldLevel
   Similar to symmetric bold method

Algorithm Description:

As shown in Fig 3.8, curve1 and curve2 are a pair of symmetric curves. The curve curve1 has a list of points, List1 = [5,6,7,8,9]. The curve curve2 has a list of points, List2 = [20,19,18,17,16,15]. Moreover, curve curve1 and curve2 approach to a straight line. Thus, rotate bold method can be applied to bold the outline.

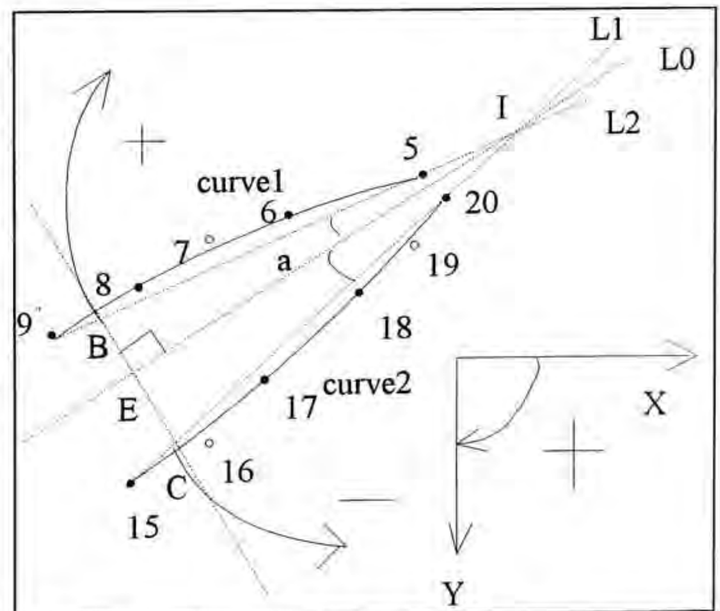


Figure 3.8: Rotate Bold Method

L1 is a line joining point 15 and point 20, and L2 is a line joining point 9 and point 5. I is the interception of L1 and L2. L0 is a bisector of the angle between L1 and L2. BC is a line perpendicular to line L0. Line BC meets line L0, L1, L2 at point

E, C and B respectively. The angle a is the angle BIE, and angle BIE is equals to angle CIE.

The bold effect can simply be attained by rotating the points of curve1 in a positive direction about I, such that the length of BE is increased by BoldLevel*BE, and the length EI is unchanged. Therefore, tan(a) is increased by BoldLevel*tan(a). Furthermore, the points of curve2 are rotated about I in a negative direction. If the angle of rotation for points in curve1 is da, the angle of rotation for points in curve2 is -da.

Thus, the angle of rotation = da = $\tan^{-1}((BoldLevel + 1.0)\tan(a)) - a$
The points in curve1 are rotated by da about I, such that

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(da) & -\sin(da) \\ \sin(da) & \cos(da) \end{pmatrix} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} Ix \\ Iy \end{pmatrix} \right) + \begin{pmatrix} Ix \\ Iy \end{pmatrix} \qquad (3.6)$$

where $\begin{pmatrix} x \\ y \end{pmatrix}$ is a point on curve1, and $\begin{pmatrix} Ix \\ Iy \end{pmatrix}$ is the point I.

The points on curve2 are rotated by -da about I, such that

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos(da) & \sin(da) \\ -\sin(da) & \cos(da) \end{pmatrix} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} Ix \\ Iy \end{pmatrix} \right) + \begin{pmatrix} Ix \\ Iy \end{pmatrix} \qquad (3.7)$$

where $\begin{pmatrix} x \\ y \end{pmatrix}$ is a point on curve2, and $\begin{pmatrix} Ix \\ Iy \end{pmatrix}$ is the point I.

To find tan(a), if
slope1 = slope of L1, slope2 = slope of L2 then

$$\tan(\angle BIC) = \frac{slope1 - slope2}{1 + slope1 * slope2} = \tan(2a)$$

Since, $\tan(2A) = \dfrac{2\tan(A)}{1 - \tan^2 A}$

$$\text{Thus, } \tan(a) = \frac{\sqrt{1 + \tan^2(2a)} - 1}{\tan(2a)} \qquad (3.8)$$

Pseudo Code of Rotate Bold

```
RotateBold(List1,List2,BoldLevel)
input : List1 is an ordered list of point numbers for the first curve segment.
        List2 is an ordered list of point numbers for the second curve segment.
        curve segments List1 and List2 are symmetric, and
        order of List1 is the reverse of the order of List2.
        BoldLevel is a level of boldness, such that it is a rational number between 0 and 1.0
{
  n1 = number of points in List1;
  n2 = number of points in List2;
  p1 = 0 th point of List1; p2 = (n1-1) th point of List1;
  q1 = 0 th point of List2; q2 = (n2-1) th point of List2;
  slope1 = (p1.y-p2.y)/(p1.x-p2.x+0.0000001); /* 0.0000001 copes with division by zero */
  slope2 = (q1.y-q2.y)/(q1.x-q2.x+0.0000001);
  c = (slope1-slope2)/(1+slope1*slope2+0.00000001);
  b = (sqrt(1+c**2)-1)/c;
  da = atan((BoldLevel+1.0)*b) - atan(b);
  (Ix,Iy) = the interception point of Line p1,p2 and Line q1,q2;
  for(i=0; i < n1; i++) {
     (x,y) is the i th point in List1;
     x = cos(da)*(x-Ix)-sin(da)*(y-Iy)+Ix;
     y = sin(da)*(x-Ix)+cos(da)*(y-Iy)+Iy;
  };
  for(i=0; i < n2; i++) {
     (x,y) is the i th point in List2;
     x = cos(da)*(x-Ix)+sin(da)*(y-Iy)+Ix;
     y = -sin(da)*(x-Ix)+cos(da)*(y-Iy)+Iy;
  };
}
```

3.4 Asymmetric Bold Instruction

Purpose:

Both Symmetric Bold and Rotate Bold instructions can only deal with symmetric curves pair, but can not handle a stroke mainly composed of asymmetric curves pair. As shown in fig 3.9, a dot stroke is composed of a pair of asymmetric curves pair, that is, curve AB and curve CD. J is a point on curve AB, at which the curve concavity is changed. Curve AJ is concave, and curve JB is convex. However, curve CD is convex without change of concavity. Thus, curve AB and curve CD are not symmetric to each other. If Symmetric Bold method is applied to curve AB and curve CD, the concavity feature cannot be preserved. Thus, Asymmetric Bold method is designed to solve this problem.

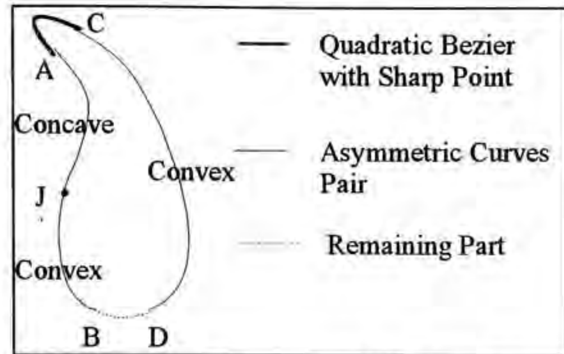


Figure 3.9: Asymmetric Curves Pair of a Dot Stroke

Input Parameters:

1. Asymmetric Curves Pair

a. an ordered list of point numbers for the first curve segment.

b. an ordered list of point numbers for the second curve segment.

c. The order of point number for curve1 is the reverse of the order of point number for curve2

2. Skeleton

Skeleton is a lines polygon to describe the shape represented by asymmetric curves pair.

a. an order list of on-curve points for the skeleton

3. BoldLevel: level of boldness, a rational number from 0.0 to 1.0

Algorithm Description:

Curve AB and curve CD are asymmetric to each other. The polygon of lines (P0,P1,P2,P3) is the skeleton of curve pair AB and CD. A mapping (Fab) of points from curve AB to the skeleton which is surjective can only be defined. Z is a point on curve AB, and Z' (Fab(Z)) is a point on the skeleton.

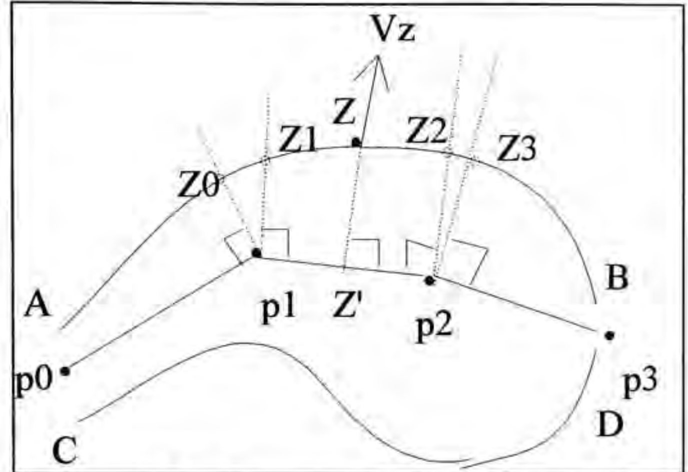


Figure 3.10: Asymmetric Bold Method.

Then, the displacement vector for Z is:

$$Vz = BoldLevel\left\langle \begin{matrix} Z.x - Z'.x \\ Z.y - Z'.y \end{matrix} \right\rangle \qquad (3.9)$$

and Z is updated by Vz, such that Z = Z + Vz

The mapping Fab is defined as

Fab: $P_{curveAB} \xrightarrow{surjective_only} P_{skeleton}$

where $P_{curveAB}$ is the set of points of curve AB, and $P_{skeleton}$ is the set of points of the skeleton (polygon of lines P0,P1,P2,Pm).

For any Z $\in P_{curveAB}$, then Z' $\in P_{skeleton}$ such that $Z \xrightarrow{Fab} Z'$ (Fab(Z) = Z')

The following rules must be satisfied:

1. If $\exists$ i $\in$ [0,1,2,...,m-1], such that Z can be vertically projected to the line segment Pi,Pi+1 , Then, Z' = the image of Z on the line segment Pi,Pi+1 is Z'
2. If rule 1 can not be satisfied and

   $\exists j \in [0,1,2,...m]$ such that $\forall i \in [0,1,2,....,m]$, ||Pj,Z|| ≤ ||Pi,Z||

   ||a,b|| is the Euclidean distance between point a and point b.

   Then, Z' = Pj

As shown in fig 3.10, rule 1 can be applied to the points on curve segments AZ0,Z1Z2, Z3B,

1. points on curve AZ0 are vertically projected to the line segment P0P1.
2. points on curve Z1Z2 are vertically projected to the line segment P1P2.
3. points on curve Z3B are vertically projected to the line segment P2P3.

and rule 2 can be applied to the points in curve segments Z0Z1 and Z2Z3.

1. points in curve Z0Z1 is mapped to point P1.
2. points in curve Z2Z3 is mapped to point P2.

To determine if a point (h,k) can be vertically projected to a line segment (x1,y1) and (x2,y2):

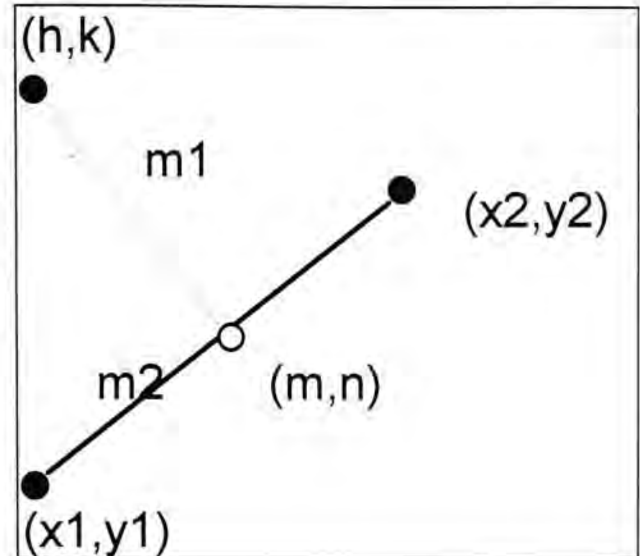


Figure 3.11: Vertical Projection of a Point (h,k) to a Line Segment (x1,y1), (x2,y2)

Parametric equation of line (x1,y1) and (x2,y2) is:
x = x1 + t(x2-x1) [1]
y = y1 + t(y2-y1) [2]
where $t \in [0,1]$

slope of line (x1,y1), (x2,y2) = m2 = (y2-y1)/(x2-x1) [3]
slope of line (h,k),(m,n) = m1 = (n-k)/(m-h) [4]
where (m,n) is the image of (h,k)
Since, Line (h,k).(m,n) is perpendicular to line (x1,y1), (x2,y2).
Then m1*m2 = -1 [5]

By equation1, 2,
m = x1 + t'(x2-x1) [6]
n = y1 + t'(y2-y1) [7]

By substituting equation 6,7 into equation 4, we get
m1 = (y1+t'(y2-y1)-k)/(x1+t'(x2-x1)-h) -------equation 8
substituting equation 8, 3 into equation 5, and simplify it, we get

$$t' = \frac{(x2 - x1)(h - x1) + (y2 - y1)(k - y1)}{(x2 - x1)^2 + (y2 - y1)^2} \quad (3.10)$$

if t' < 0 or t' > 1 then
the point (h,k) can not be vertically projected to the line (x1,y1),(x2,y2)

Pseudo Code of Asymmetric Bold Instruction:

```
AsymmetricBold(List1,List2,Skeleton,BoldLevel)
input : List1 is an ordered list of point numbers for the first curve segment.
        List2 is an ordered list of point numbers for the second curve segment.
        curve segments List1 and List2 are asymmetric, and
        order of List1 is the reverse of the order of List2.
        Skeleton is an order list of points for the skeleton of List1 and List2.
        BoldLevel is a level of boldness, such that it is a rational number between 0 to 1.0
{
  /* modify the curve of List1 to attain bold effect */
  i = 0; j = 0;
  While(i < |List1|) { /* |List1| is the cardinality of List1 */
      Z = i th point of List1;
      p1 = j th point of Skeleton; p2 = (j+1) th point of Skeleton;
      t' = ((p2.x-p1.x)*(Z.x-p1.x)+(p2.y-p1.y)*(Z.y-p1.y)) /
           ((p2.x-p1.x)**2+(p2.y-p1.y)**2);
      if((t'<0) || (t'>1)) {
         j++;
         p1 = j th point of Skeleton, p2 = (j+1) th point of Skeleton;
         do {
           t' = ((p2.x-p1.x)*(Z.x-p1.x)+(p2.y-p1.y)*(Z.y-p1.y)) /
             ((p2.x-p1.x)**2+(p2.y-p1.y)**2);

           Vz=displacement vector of Z= BoldLevel<Z.x - p1.x, Z.y - p1.y>;

           Z = Z + Vz;
           i++;
           Z = i th point of List1;
         } while((t'<0) || (t'>1));
         i++;
         continue;
      };

      Z' is a point, and it is the projected image of Z on the Skeleton, such that
         Z'.x = p1.x + t'*(p2.x-p1.x);
         Z'.y = p1.y + t'*(p2.y-p1.y);

      Vz =displacement vector of Z=BoldLevel<Z.x - Z'.x, Z.y - Z'.y>;

      Z = Z + Vz;
      i++;
  };
  /* modify the curve of List2 to attain bold effect */
  ............................
}
```

where the displacement vectors are

$$V_z = BoldLevel\left\langle \begin{matrix} Z.x - p1.x \\ Z.y - p1.y \end{matrix} \right\rangle ; \qquad V_z = BoldLevel\left\langle \begin{matrix} Z.x - Z'.x \\ Z.y - Z'.y \end{matrix} \right\rangle ;$$

3.5 Comparison of Bold Instructions

The idea of the auto bold instructions is that Chinese character is mainly composed of a number of curves pair which are highly correlated in their (x,y) coordinate to each other. In addition, the curves pair can be classified as symmetric curves pair or a asymmetric curves pair. SymmetricBold instruction can be applied to symmetric curves pair to attain bold effect. However, the complexity of SymmetricBold is not desired. The RotateBold instruction is only a special case of SymmetricBold instruction, and it works on the curves pair which is approaching to a pair of straight lines. The complexity of RotateBold instruction is much less than the complexity of SymmetricBold instruction. Because, RotateBold algorithm does not need missing part recovery, but SymmetricBold algorithm needs it. Moreover, RotateBold algorithm is non-iterative, but SymmetricBold algorithm is iterative.

AsymetricBold instruction can be applied to a asymmetric curves pair, but it is necessary to input the skeleton (a polygon of lines) into the function call. Because, it is very hard to compute the skeleton of a asymmetric curves pair, we leave this work to the programmer. AsymmetricBold instruction is efficient, and it can preserve the characteristics of the curves pair. If SymmetricBold instruction is applied to a pair of asymmetric curves pair, the characteristics of the curves pair will no longer be kept.

So, we can summarize the properties of the auto bold instructions in the following table:

Instruction	Curves Pair	Iterative	Complex	Strokes can be applied
SymmetricBold	symmetric	yes	yes	leftIncline stroke rightIncline stroke RightHoke
RotateBold	symmetric	no	no	upperIncline stroke
AsymmetricBold	asymmetric	no	no	dot stroke drop stroke

3.6 Serif Accommodation Instruction

Purpose:

A stroke can be divided as pure stroke part, and serif part. The existence of serif is for decorative purpose. Moreover, it can enhance the readability of a document. The metrics of a serif include serif width and serif height, as shown in fig 3.11a. However, the existence of a serif does cause trouble to auto boldness, because a serif will diminish when bold level is high. In addition, serif cannot be handled simply by interpolation, such as IUPx and IUPy TrueType instructions. Thus, we devise a new method to handle the serif with the purpose of keeping the serif shape, width and height, even though the bold level is high.

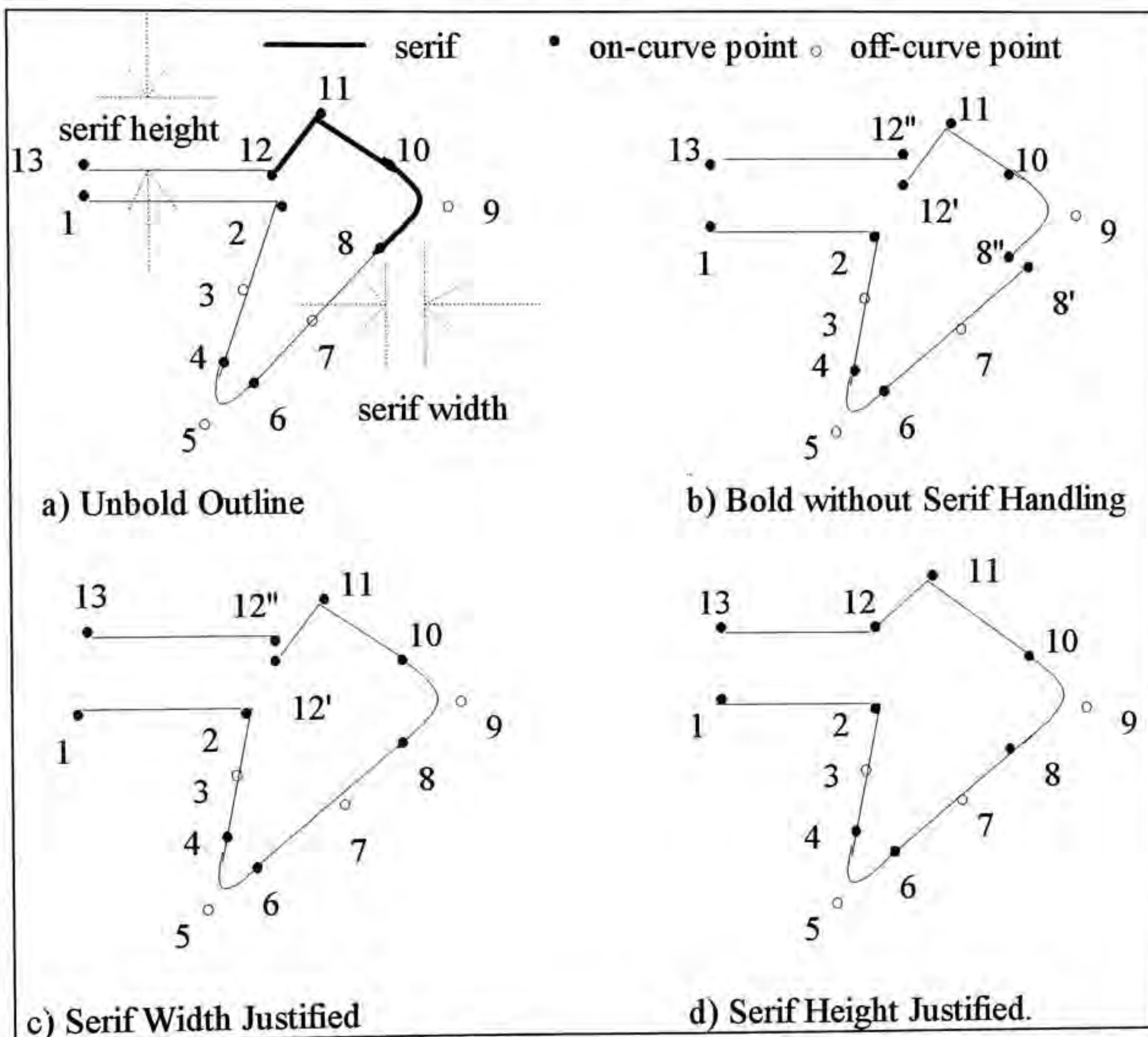


Figure 3.11: Serif Accommodation

Input Parameters:

1. Serif Outline
A segmented serif outline after pure stroke outline is bolded. It is an order list of point numbers for the serif outline, and it is in increasing order of point number.
In fig 3.11b, serif = [8,9,10,11,12]

2. Direction Vector Vertexes
It defines the migrating direction of serif points.

a. First Vertex (V_α) s.t. $V_\alpha \in serif$
   point 10 in fig 3.11

b. Second Vertex (V_β) s.t. $V_\beta \in serif$

Algorithm Description:

As shown in fig 3.11b, pure stroke outline is already bolded, but the serif outline is still not handled. Thus, there is a disparity at the joint between the serif outline and pure stroke outline. Point 12 and point 8 are the joint points. Point 12 is split into two points 12" and 12', because 12" is the modified point 12 and 12' is the original copy of point 12. Similarly, 8' is the modified point 8, and 8" is the original copy of point 8.

1. Direction Vector

It defines the migrating direction of serif control points, such that

$$V = \left\langle \begin{matrix} V_\alpha.x - V_\beta.x \\ V_\alpha.y - V_\beta.y \end{matrix} \right\rangle \qquad (3.11)$$

2. Justify Serif Width

To keep the serif width, point 8",9,10 (V_α) must be moved in the direction of V so that point 8" and point 8' can be re-emerged as a single point.

L1 is a line joining point7 and point 8', s.t.

$$\frac{P8'.y - P7.y}{P8'.x - P7.x} = \frac{y - P7.y}{x - P7.x} \qquad [1]$$

L2 is a line passing through p8" with a slope Vy/Vx, s.t.

$$\frac{y - P8''.y}{x - P8''.x} = \frac{V_y}{V_x} \qquad [2]$$

I is the interception point of L1 and L2, s.t
(I.x,I.y) is the solution of the simultaneous equations 1 and 2.

Thus, the displacement vector for points 8",9,10 = Vd = $\left\langle \begin{matrix} I.x - P8''.x \\ I.y - P8''.y \end{matrix} \right\rangle$

Then, points 8",9,10 is updated by Vd. e.g. P8" = P8" + Vd
Moreover, P8' = (I.x,I.y).

3. Justify Serif Height

To keep the serif height, point 12' (V_β), 11 must be moved in the direction of V so that point 12' and 12" can be reemerged as a single point.

L1 is a line joining point 13 and point 12", s.t.

$$\frac{P13.y - P12''.y}{P13.x - P12''.x} = \frac{y - P12''.y}{x - P12''.x} \qquad [1]$$

L2 is a line passing through point 12' with a slope Vy/Vx, s.t.

$$\frac{y - P12'.y}{x - P12'.x} = \frac{V_y}{V_x} \qquad [2]$$

I is the interception point of L1 and L2, s.t.
(I.x,I.y) is the solution of the simultaneous equations 1 and 2.

Thus, the displacement vector for points 12',11 = Vd = $\left\langle \begin{matrix} I.y - P12'.y \\ I.x - P12'.x \end{matrix} \right\rangle$

Then, the points 12',11 are updated by Vd. e.g. P12' = P12' + Vd
Moreover, P12" = I

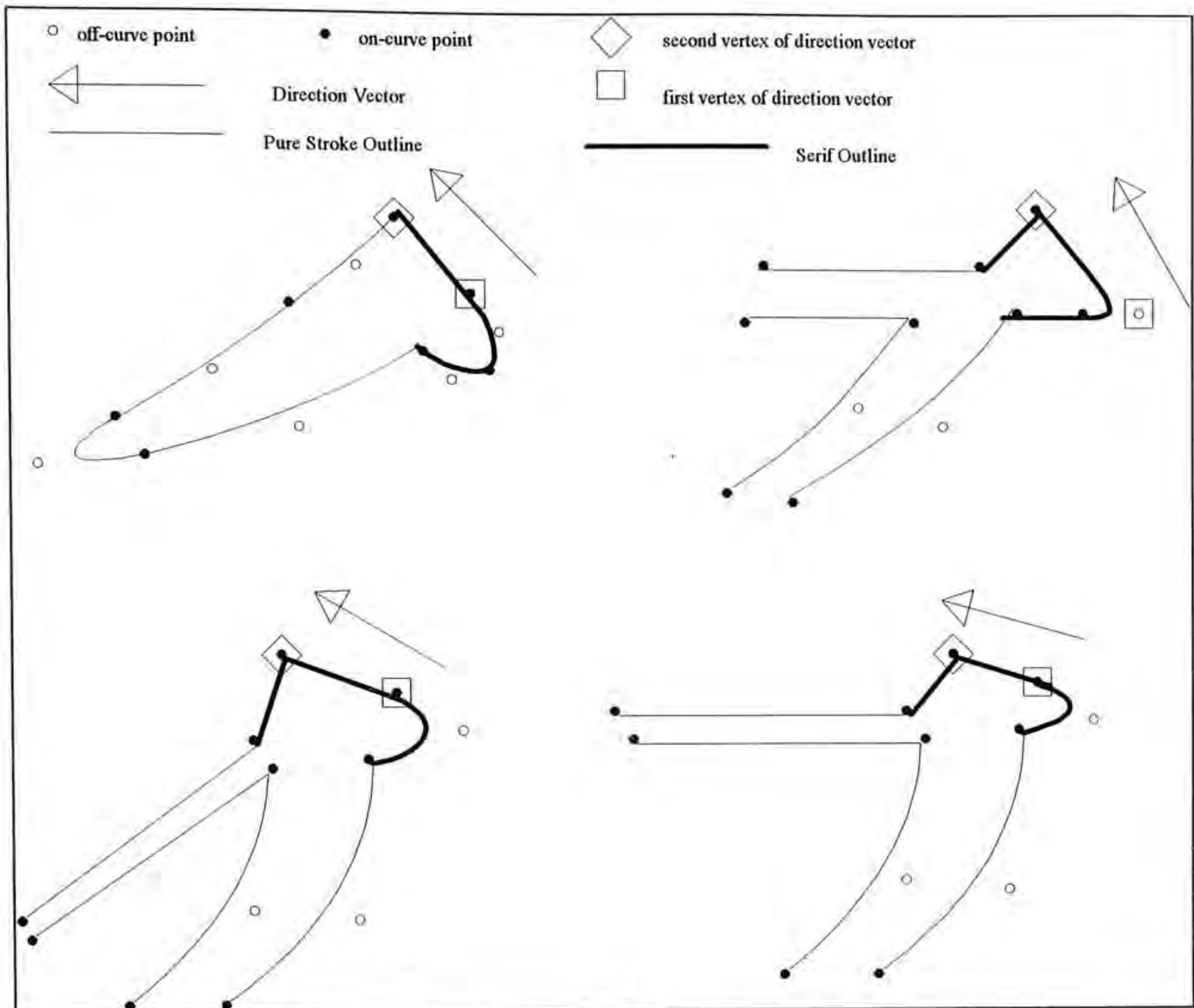


Figure 3.12: Direction Vectors for other Serifs

4. Determination of Direction Vector

Once the direction vector of a serif is determined, it is very easy to process the serif. Thus, the determination of a direction vector for a serif is a crucial matter. As shown in fig 3.12, it is possible to generalize the heuristic rules to govern the determination of a direction vector. However, there are still some cases that are hard to generalize. So, the heuristic rules are listed just for reference.

Suppose, a serif with a list of point number, serif = [s(0),s(1),s(2),.....,s(m)] in increasing order of point number.
1] define an unit vector

$$v' = \frac{1}{\sqrt{(s(m).y - s(0).y)^2 + (s(m).x - s(0).x)^2}} \left\langle \begin{matrix} s(m).y - s(0).y \\ s(m).x - s(0).x \end{matrix} \right\rangle \quad (3.12)$$

2] any k $\in [0,1,2,..,m-1]$, an unit vector can be defined

$$v''(k) = \frac{1}{\sqrt{(s(k+1).y - s(k).y)^2 + (s(k+1).x - s(k).x)^2}} \left\langle \begin{matrix} s(k+1).y - s(k).y \\ s(k+1).x - s(k).x \end{matrix} \right\rangle$$

(3.13)

3] any k $\in [0,1,2,...,m-1]$, a discriminant function Δ(k) can be defined
$\Delta(k) = abs(v'.v''(k) - 1)$ where . is a dot product, and abs is the absolute value
(3.14)
4] if $\exists \rho \in [0,1,2,..,m-1]$ s.t. $\forall \eta \in [0,1,2,..,m-1]$ $\Delta(\rho) \le \Delta(\eta)$
then The first vertex of Direction Vector = $V_\alpha = \rho$

The second vertex of Direction Vector = $V_\beta = \rho + 1$

The serif of a character will diminish or degrade in shape, when the bold level is high. Thus, we derive an instruction SerifAccommodation to handle the problem of serif during auto boldness. Character outline can be divided into pure stroke parts and serif parts. The instruction SerifAccommodation is applied to serif parts after the pure stroke parts are bolded by using the instruction SymmetricBold, RotateBold or AsymmetricBold. The instruction SerifAccommodation can preserve the serif height, serif width, and the shape of serif nicely. Moreover, the algorithm is general enough to be applied to all types of serif, but it needs an input of the Direction Vector (V), characterized by the points V_α and V_β. The points V_α and V_β are the control points of a serif. We have listed some heuristic rules to find V_α and V_β by giving a serif. Nevertheless, in some cases these heuristic rules will fail. Thus, we leave the work of finding V_α and V_β to programmer.

5. Pseudo Code of Serif Accommodation

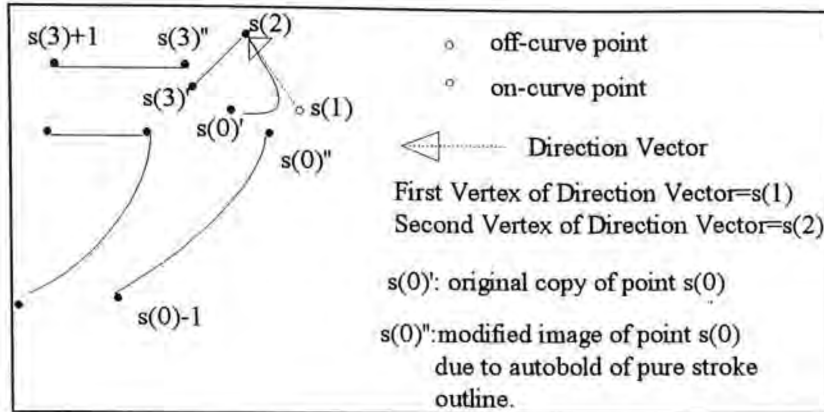


Figure 3.13: Labels of a Serif after Auto Bold of Pure Stroke Outline

SerifAccommodation(Serif, V_α, V_β)

input :

Serif - an order list of points number for the serif, and it is in increasing order of point number.

Serif=[s(0),s(1),s(2),...,s(m)]

V_α - the first vertex of direction vector

V_β - the second vertex of direction vector, see fig 3.13.

{

find the direction vector, $v = \left\langle \begin{matrix} V_\alpha.x - V_\beta.x \\ V_\alpha.y - V_\beta.y \end{matrix} \right\rangle$

/* Justifying Serif Width */

L1 : a line passing through S(0)' with slope =V.y/V.x

s.t. $\frac{y - S(0)'.y}{x - S(0)'.x} = \frac{V.y}{V.x}$ [1]

L2 : a line passing through S(0)'' and $P_{S(0)-1}$

s.t. $\frac{y - S(0)''.y}{x - S(0)''.x} = \frac{P_{s(0)-1}.y - S(0)''.y}{P_{s(0)-1}.x - S(0)''.x}$ [2]

Solve the simultaneous equation 1 and 2 with a solution x = I.x and y = I.y

s.t. I = (I.x, I.y)

Find the displacement vector $Vd = \left\langle \begin{matrix} I.y - S(0)'.y \\ I.x - S(0)'.x \end{matrix} \right\rangle$

for(i=1; i <= V_α;i++) {

S(i).x = S(i).x + Vd.x;

S(i).y = S(i).y + Vd.y;

};

S(0) = I;

/* Justifying Serif Height */

L1: a line passing through S(m)' with a slope = V.y/V.x

s.t. $$\frac{y - S(m)'.y}{x - S(m)'.x} = \frac{V.y}{V.x} \qquad [3]$$

L2: a line passing through S(m)'' and $P_{s(m)+1}$

s.t. $$\frac{y - S(m)''.y}{x - S(m)''.x} = \frac{P_{s(m)+1}.y - S(m)''.y}{P_{s(m)+1}.x - S(m)''.x} \qquad [4]$$

Solve the simultaneous equation 3 and 4 with a solution x = I.x and y = I.y
s.t. I = (I.x,I.y)

Find the displacement vector $Vd = \left\langle \begin{matrix} I.y - S(m)'.y \\ I.x - S(m)'.x \end{matrix} \right\rangle$

```
for(i=Vβ;i<m;i++) {
    S(i).x = S(i).x + Vd.x;
    S(i).y = S(i).y + Vd.y;
};
S(m) = I;
}
```

Chapter Four: Character Shape Parsing and Auto Bold Code Generation

4.1 Compilation Process and Auto Boldness

In the compilation process, a program source code, which is a sequence of ASCII codes, will be translated to a tokens sequence in Lexical Analysis Phase. Then the tokens sequence will be parsed, and a parse tree will be generated in the Parsing Phase. Finally the parse tree will be evaluated to generate the assembly codes in the Code Generation Phase. The idea of auto boldness is very similar to a compilation process [Shape85]. The glyph outline is compiled and finally an auto bold program, which is built on top of the auto bold instructions in the previous chapter, is generated. The similarities between the program compilation and glyph outline compilation is summarized in the following table:

Program Compilation	*Glyph Outline Compilation*
Source program which is a sequence of ASCII codes	Glyph outline, which is composed of contours, and each contour is composed of a number of joined primitives, which are either straight line or quadratic bezier
Lexical Analysis to identify tokens	Shape Lexical Analysis to identify shape tokens
Token (if, for, case...) composed of one or more consecutive ASCII characters in the source program	Shape Token (continuous segment, sharp point,.....) composed of one or more joined primitives in the glyph outline
Parser (to parse the whole program)	Shape Parser(to parse a stroke outline)
Parse Rules(input of parser)	Shape Parse Rules
Parse Tree(output of parser)	Mapping from stroke model key points to input stroke control points
Evaluate parse tree to generate assembly code program	Generate auto bold program for input stroke by using the stroke model key points mapping

Shape compilation is very similar to program compilation, except that the compiled object of shape compilation is a two dimensional object, which is a glyph outline control points, but the compiled object of program compilation is a one dimensional object, which is a sequence of ASCII codes. Moreover, it needs a tremendous amount of parse rules to describe the whole outline of a Chinese character because a Chinese character is very complicated in shape, resulted from the combinations of strokes. Thus, it is awkward to design parsing rules for each Chinese character. However, if the strokes of a Chinese character are segmented first, the segmented strokes can be compiled separately. It is due to the fact that the shapes of stroke is much less complex than the character shape which is a

combinations of strokes in two dimensional space. Therefore, stroke segmentation plays an important role in the auto boldness of Chinese character.

Furthermore, the general format of grammar rules in program compilation is A :- B, where A is the head, and B is the body of the grammar rule. This format is insufficient for describing a two dimensional object. As a result, we design a new parsing rule format to describe a stroke of Chinese character. In the traditional parsing rule, there is an implicit AND conjunction to join two items of the body. Nevertheless, the stroke parsing rule needs two types of conjunction. They are * and + conjunctions. The conjunction * deals with the complete curve segments of a stroke which is not intercepted by other stroke, but the conjunction + deals with the incomplete curve segments of a stroke which is intercepted by another stroke. In addition, each shape grammar rule is attached by a set of condition statements and assignment statements. The input stroke is matched to grammar rules of all classes of stroke one by one, until one of stroke class grammar rules are all matched. Then, the input stroke is classified as that stroke class. If no match occurs, the input stroke is unclassified, and left unprocessed.

So, in the shape parsing phase, the stroke class is ascertained, and a mapping from the stroke model key points to the input stroke control points is established. Each stroke model has his own auto bold program. Thus, the auto bold program for the input stroke can be generated by replacing the key points in the model auto bold program by the control points of the input stroke.

4.2 Shape Lexical Analyzer

The character outline is passed to the Shape Lexical Analyzer for identifying the shape tokens, and finding the attributes value of shape tokens. There are two levels of shape tokens. Level one shape token is constituted by a single primitive which is either a straight line or a quadratic Bezier, and level two shape token is constituted by many level one shape tokens which are joined together in the glyph outline.

Each shape token has an identifier, which is a string of small bold letters. A dot notation is used to denote the attribute of a token. **ident1**.*a1* means the attribute *a1* of a token identified by **ident1**.

Thus, the input of Shape Lexical Analyzer is the control points coordinate of a glyph outline, and the output of Shape Lexical Analyzer is the shape tokens with attributes. Then, the shape tokens are input to the Shape Parser to segment and classify the strokes.

There are several reasons for separating the analysis phase of Shape Compiling into Shape Lexical Analysis and Shape Parser.

1. Simpler design is perhaps the most important consideration. The separation of Shape Lexical Analysis from Shape Parser often allows us to simplify one or the other of these phases. For example, font designer characteristics and minor error due to the rounding of control point coordinates are filtered out by the Shape Lexical Analyzer. Then, the algorithm of Shape Parser can be greatly simplified.

2. The Shape Compiler efficiency is improved: The shape tokens and token attributes are all found once by the Shape Lexical Analyzer. Thus, it is not necessary to find the tokens and token attributes repeatedly in the Shape Parser. Moreover, some efficient algorithm can be used to find the tokens and token attributes in Shape Lexical Analysis Phase.

3. The Shape Compiler Portability is enhanced: The shape token identification algorithm can be easily generalized so that the Shape Lexical Analyzer can be reused in developing the auto bold driver for other Chinese fonts. It meets the principle of reusability and portability in Software Engineering.

The Shape Tokens and Attributes can be summarized by the following table:

Level	Token Ident	Description	Attributes
one	**line**	a straight line primitive with two on-curve control points	*class*: define the class of a line primitive, with values slant, horizontal, or vertical τ*:* tendency of primitive *direction*: define the direction of the vector from the first control point to the second control point, with value 1 if direction is in $[0^o, 45^o)$, value 2 if direction is in $[45^o, 90^o)$,............., value 8 if direction is in $[315^o, 360^o)$ *length*: the length of line in pixel unit.
one	**bezier2**	a quadratic bezier primitive with two on-curve control points and one off-curve control point	*concavity*: define the concavity of a quadratic bezier, with values concave, convex or flat *direction*: define the direction of the vector from the first control point to the third control point τ_1: slope of tangent at the first control point τ_2:slope of tangent at the third control point τ: tendency of the primitive *length*: the length of quadratic bezier in pixel unit
one	**sharp**	a quadratic bezier with a sharp point	*class*: the class of sharp point
two	**concave**	a curve segment which is a set of joined, continuous, and concave primitives	*points[]:* a list of point number for the curve segment *n:* number of points *length*: the length of curve segment τ: tendency of the curve segment

two	**convex**	a curve segment with is a set of joined, continuous, and convex primitives	*points[]:* a list of point number for the curve segment *n:* number of points *length*: the length of curve segment τ: tendency of the curve segment

4.3 Shape Tokens Attributes Evaluation

4.3.1 **line** Token

line token is a straight line primitive with two on-curve control points (x0,y0) and (x1,y1), and it has the attributes *class*, *length*, τ, and *direction*.

Attribute *length:*

length is the length of the primitive, and it can be calculated by the formula:

$$length = \sqrt{(x1-x0)^2+(y1-y0)^2} \quad (4.1)$$

Attribute τ:

τ is the slope of the primitive, and it can be calculated by the formula:

$$\tau = \frac{y1-y0}{x1-x0+0.000001} \quad (4.2)$$

it is noted that 0.000001 copes with the divide by zero error.

Attribute *class*:

The attribute *class* defines the topological property of the primitive
if *class* = horizontal the primitive is a horizontal line segment.
if *class* = vertical the primitive is a vertical line segment.
if *class* = slant the primitive is a slant line segment.

So, if $-0.364 < \tau < 0.364$ (=tan(20°)), it is a horizontal line segment.
if $-2.748 < \tau < 2.748$ (=tan(70°)), it is a vertical line segment.
otherwise, it is a slant line segment.

Attribute *direction*:

The attribute direction defines the direction of the primitive. The Euclidean space is divided into 8 quadrants to classify the direction, as shown in fig 4.1.

The algorithm to find the attribute *direction* is:

```
dx = x1 - x0;  dy = y1 - y0;
if(abs(dx)<abs(dy)) {
  if((dx>0)&&(dy>0))
    direction = 7;
  else if((dx>0)&&(dy<0))
    direction = 2;
  else if((dx<0)&&(dy>0))
    direction = 6;
  else
    direction = 3;
} else {
  if((dx>0)&&(dy>0))
    direction = 8;
  else if((dx>0)&&(dy<0))
    direction = 1;
  else if((dx<0)&&(dy>0))
    direction = 5;
  else
    direction = 4;
}
```

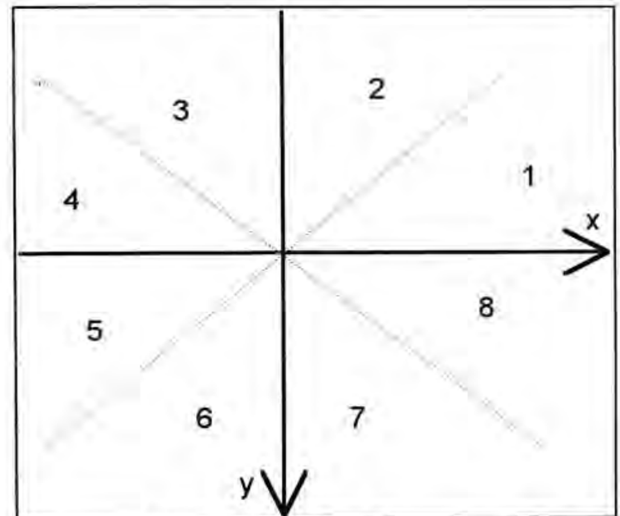


Figure 4.1: Direction Classification

4.3.2 **bezier2** Token

bezier2 token is a quadratic Bezier primitive with control points (x0,y0), (x1,y1) and (x2,y2). (x0,y0) and (x2,y2) are on-curve control points, and (x1,y1) is an off-curve control point.

Attribute *concavity*:

The attribute concavity defines the concavity of a quadratic Bezier primitive. If concavity = convex, the primitive is convex. If concavity = concave, the primitive is concave.

The concavity of a quadratic Bezier can be defined mathematically as:

1. define a set $U = \{(x,y): x = round(x0 + t(x2-x0)), y = round(y0 + t(y2-y0))$, where $t \in (0,1)$ and $x,y \in Z\}$

2. if $\exists \xi \in U$, s.t. ξ is an interior point (ξ is a point in black area of raster image), then the primitive is convex

Otherwise, the primitive is concave.

To use the above rule to find the concavity of a primitive is not efficient, because it needs to determine whether a point is interior or not, and it is computationally

expensive. Thus, we use a heuristic rule to determine the concavity. However, the heuristic rule holds only if the following assumptions are made:

1. The origin of the coordinate system is at the upper left corner.
2. The outermost contour is always in an anti-clockwise direction.
3. The direction of the enclosed contour is the reverse of the direction of the enclosing contour.

The heuristic rule to determine the concavity is:
1. Find the area of the triangle (x0,y0), (x1,y1) and (x2,y2) by using the matrix formula:

$$Area = \frac{1}{2}\begin{vmatrix} 1, x0, y0 \\ 1, x1, y1 \\ 1, x2, y2 \\ 1, x0, y0 \end{vmatrix} = (x1y2+x2y0-x1y0-x2y1)/2 \qquad (4.2)$$

2. if Area < 0, then concavity = convex; else concavity = concave.

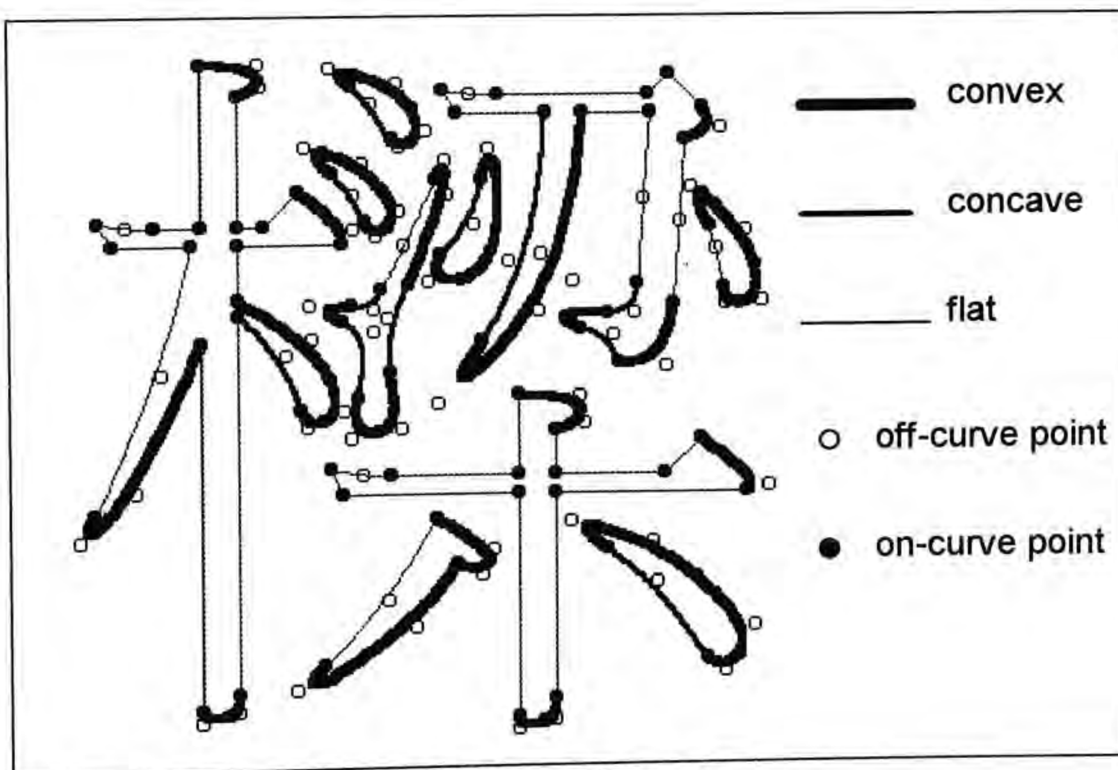


Figure 4.2: Concavity of a Chinese Character

In fig 4.2, the concavity of primitives is found by using the above heuristic rules. It is noted that a quadratic bezier can be flat, if the three control points are approximately collinear.

Attribute *length*:

The attribute length is the length of the quadratic Bezier. There is no simple analytical solution to find the curve length of a quadratic Bezier. Therefore, the quadratic bezier is subdivided as a polygon of lines by using the de Casteljau Algorithm. Then, the approximate curve length can be found.

Attribute τ_1 and τ_2:

τ_1 is the slope of the tangent at the first control point (x0,y0), and τ_2 is the slope of the tangent at the third control point (x2,y2). Because the quadratic Bezier has the property that the line joining (x0,y0) and (x1,y1) is the tangent at (x0,y0), and the line joining (x1,y1) and (x2,y2) is the tangent at (x2,y2),

$$\tau_1 = \frac{y1 - y0}{x1 - x0 + 0.00001}, \quad \tau_2 = \frac{y2 - y1}{x2 - x1 + 0.00001} \qquad (4.3).$$

(0.00001 deals with divide by zero error)

Attribute τ:

The attribute τ is the tendency of the quadratic bezier, that is the average tangent slope of the quadratic bezier.

The quadratic bezier can be expressed parametrically as:

$$x(t) = (1-t)^2 x0 + 2t(1-t)x1 + t^2 x2 \qquad [1]$$

$$y(t) = (1-t)^2 y0 + 2t(1-t)y1 + t^2 y2 \qquad [2]$$

By equation 1, $\frac{dx}{dt} = 2(x0 - 2x1 + x2)t - 2x0 + 2x1$ [3]

By equation 2, $\frac{dy}{dt} = 2(y0 - 2y1 + y2)t - 2y0 + 2y1$ [4]

$$\frac{dy}{dx} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} = \frac{(y0 - 2y1 + y2)t - y0 + y1}{(x0 - 2x1 + x2)t - x0 + x1} \qquad [5]$$

Take a = y0-2y1+y2, b = y1-y0, c = x0-2x1+x2, d = x1-x0

$$\tau = \int_0^1 \frac{dy}{dx} dt = \int_0^1 \frac{at+b}{ct+d} dt$$

Let u = ct+d　　　　　　du = cdt
=> t = (u-d)/c　　　　　==> dt = du/c
when t = 0 ==> u = d
　　　t = 1 ==> c + d

$$\tau = \int_d^{c+d} \frac{\frac{a}{c}u - \frac{ad}{c} + b}{u} \frac{du}{c} = \frac{a}{c^2} \int_d^{c+d} du + (\frac{b}{c} - \frac{ad}{c^2}) \int_d^{c+d} \frac{du}{u} =$$

$$\frac{a}{c^2} c + (\frac{b}{c} - \frac{ad}{c^2}) \log_e \frac{c+d}{d}$$

$$= \frac{a}{c} + \frac{1}{c^2}(bc - ad) \log_e (\frac{c}{d} + 1) \qquad (4.4)$$

4.3.3 sharp Token

A **sharp** token is a point on the contour at which the radius of curvature is less than a threshold value, and the concavity is convex. The finding of sharp token is essential to the stage of stroke classification and segmentation, because a Chinese character has the very elegant property that every stroke contains a sharp point. Thus, if the sharp point can be located, a stroke can also be located, and the located stroke can be further classified, and segmented easily.

Attribute *Class*:

The attribute class is the class of a sharp point, and this attribute is very important for the classification of the stroke to which the sharp point belongs. There are seven classes of stroke : Left Incline Stroke, Right Incline Stroke, Left Hoke, Right Hoke, Upper Incline Stroke, Dot Stroke, and Drop Stroke. The seven classes of stroke are further grouped by the direction vector of the sharp point.

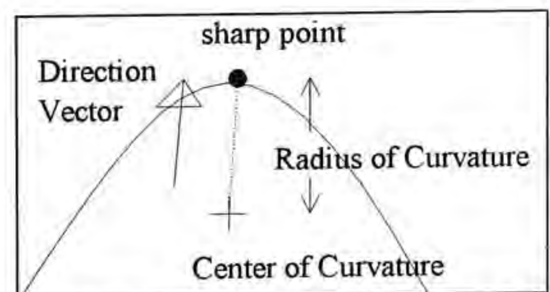


Figure 4.3: Direction Vector of a Sharp Point

For any given sharp point, a direction vector can be defined as a unit vector in the direction of the radius of curvature at the sharp point. Again, the direction of a vector can be divided into 8 quadrants, as shown in fig4.1. Then, the stroke can be grouped by the sharp point direction vector.

sharp.*class*	Direction Vector	Stroke Classes.
1	4,5	Left Incline Stroke, Left Hoke, Drop Stroke
2	1	Right Incline Stroke, Upper Incline Stroke
3	2	Dot Stroke, Right Hoke

For any given sharp point, a conditional probability can be defined:

P(Stroke=StrokeClass|Class=SharpClass) = The probability that if the sharp point belongs to a stroke with class identifier StrokeClass, after that the sharp point' attribute *class* is verified to be equal to SharpClass.

In order to simplify the argument, the occurrence probabilities of the strokes (the probability of a specific class of stroke appearing in a Chinese character outline) are assumed to be equal, and the probability P(a sharp point does not belong to any stroke) = e. Then, P(a sharp point belongs to Left Incline Stroke) =

P(a sharp point belongs to Right Incline Stroke) = ... = (1-e)/7

Thus, the conditional probability

P(the sharp point belongs to LeftInclineStroke | the sharp point is class 1) = (1-e)*1/3

P(the sharp point belongs to RightInclineStroke| the sharp point is class 2) = (1-e)*1/2

P(the sharp point belongs to DotStroke | the sharp point is class 3) = (1-e)*1/2

P(the sharp point belongs to LeftInclineStroke| the sharp point is class 2) = 0

Therefore, the possibility of strokes can be narrowed down by evaluating the class attribute first. For example, if the attribute class of a sharp point is evaluated to be 2, it is only necessary to verify whether the sharp point belongs to RightInclineStroke or UpperInclineStroke.

So we can device an algorithm to find the class of stroke:

```
Algorithm A
input : a sharp point
outline : the class of stroke to which the sharp point belongs
1. evaluate the class attribute of the sharp point
2. switch(class) {
    case 1:
            if( the sharp point matches the heuristic rules of LeftIncline Stroke)
                the sharp point belongs to a LeftIncline Stroke;
            else if( the sharp point matches the heuristic rules of LeftHoke)
                the sharp point belongs to  a LeftHoke Stroke;
            else if( the sharp point matches the heuristic rules of Drop Stroke)
                the sharp point belongs to a Drop Stroke;
            else
                the sharp point is unclassified;
    case 2:
            if( the sharp point matches the heuristic rules of RightIncline Stroke)
                the sharp point belongs to a RightIncline Stroke;
            else if(the sharp point matches the heuristic rules of UpperIncline
Stroke)
                the sharp point belongs to a UpperIncline Stroke;
            else
                the sharp point is unclassified;
    case 3:
            if(the sharp point matches the heuristic rules of Dot Stroke)
                the sharp point belongs to a Dot Stroke;
            if(the sharp point matches the heuristic rules of Right Hoke)
                the sharp point belongs to a Right Hoke;
            else
                the sharp point is unclassified;

};
```

Then, let the computational time for the matching of heuristic rules of any stroke = h and the computational time for the evaluation of the attribute class = c
The expected computational time for algorithm A
= (3/7)(3/2)*h + (2/7)*(2/2)*h + (2/7)*(2/2)*h + c
= (9/14+4/14+4/14)*h + c = (17/14)*h + c = 1.21*h + c

The algorithm can be devised in another way:

```
Algorithm B
input : a sharp point
output : the class of stroke to which the sharp belongs
if(the sharp point matches the heuristic rules of LeftIncline Stroke)
   the sharp point belongs to LeftIncline Stroke;
if(the sharp point matches the heuristic rules of RightIncline Stroke)
   the sharp point belongs to RightIncline Stroke;
.....
else
   the sharp point is unclassified;
```

The expected computational time for algorithm B
= (7/2)*h = 3.5 h

Thus, the algorithm A is much faster than algorithm B, if c is small enough. In addition, Algorithm A can be further accelerated by matching the heuristic rules of the most frequent stroke firstly, because it can avoid many redundant matchings.

For the purpose of reducing the computational time of the attribute class, some heuristics are used to identify the sharp token, and to calculate the attribute class of it, instead of solving the complicated second order differential equation of curvature. Because our problem domain is Windows 3.0 CFSUNG Chinese font, and this font has a distinct property that a sharp point of a stroke always exists in a convex quadratic primitive, of which the three control point can form an acute triangle.

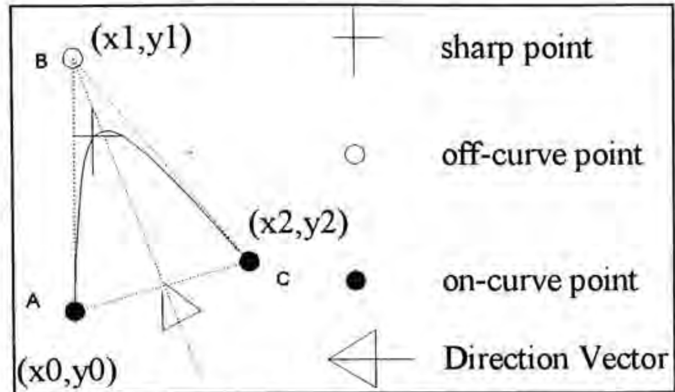


Figure 4.4: Quadratic Bezier with a Sharp Point

As shown in fig 4.4, for any quadratic bezier, let point A = first control point (x0,y0), point B = second control point (x1,y1), and point C = third control point (x2,y2).

Moreover, two vectors can be defined, the first vector v1 from point A to point B, and the second vector v2 from point B to point C.
Then the angle between the vectors v1 and v2 can be found by the formula:

$$\angle ABC = \cos^{-1}\left(\frac{(x1-x0)(x2-x1)+(y1-y0)(y2-y1)}{\sqrt{\left[(x1-x0)^2+(y1-y0)^2\right]\left[(x2-x1)^2+(y2-y1)^2\right]}}\right) \quad (4.5)$$

Thus, a quadratic bezier is considered as a primitive containing a sharp point, if

1. The absolute value of $\angle ABC < 30^o$
2. The primitive is convex.
3. Curve length of the quadratic bezier is small.

The direction vector of the quadratic bezier can also be calculated by some heuristic rules:
1. M is the mid point of point A and point C
2. The direction vector is defined to be an unit vector from point M to point B,
Thus the formula to calculate the direction vector is:

$$DirectionVector = \frac{1}{\sqrt{(x1-(x0+x2)/2)^2+(y1-(y0+y2)/2)^2}}\left\langle \begin{matrix} x1-(x0+x2)/2 \\ y1-(y0+y2)/2 \end{matrix} \right\rangle$$

(4.6)

Then, the attribute class can be calculated by classifying the direction vector by the eight quadrants.

4.3.4 **concave** Token

concave token represents a continuous curve segment which is a list of joined concave primitives. In this context, continuous curve refers to C^1 continuous mathematically. In other words, the continuous curve segment can be regarded as a list of control points, indexed by point number, so that each on-curve point in it is a continuous point.

An on-curve control point P is a continuous point if:

1. P' = a point indexed as n-1 th point, P = a point indexed as n th point, P'' = a point indexed as n+1 th point.
2. P', P,P'' are roughly collinear that is:

$$\angle A = \cos^{-1}\left(\frac{(p'.x-p.x)*(p''.x-p.x)+(p'.y-p.y)*(p''.y-p.y)}{\sqrt{\left[(p'.x-p.x)^2+(p'.y-p.y)^2\right]*\left[(p''.x-p.x)^2+(p''.y-p.y)^2\right]}}\right)$$

and $\angle A$ is roughly equal to 180°

Attribute *length*:

The attribute length is the length of the curve segment, which equals to the sum of the attribute length of the primitives constituting the curve segment.

Attribute τ:

This is the tendency of the curve segment, which equals to weighted sum of the tendency of the primitives constituting the curve segment, and the weight is the attribute *length* of the primitives.

4.3.5 **convex** Token

It is very similar to the token **concave,** except that the primitives constituting it is convex.

4.4 Scope of Shape Parsing

The processing of Chinese characters outline is much more complicated than the processing of English characters. Firstly, there is not much variation in shape among different fonts for an English character, but the outline of a Chinese character can vary much among different fonts. Because of that if the strokes of a Chinese character are not intercepted for a specific font, it is possible to be intercepted for other fonts. In contrast, this is not the case of an English character outline. Therefore, it is possible to define the grammar rules for each English character [Model 91], but it is impossible for Chinese characters. Moreover, there is only twenty four English characters, but there are several thousands of Chinese characters. Consequently, it is extremely labor intensive to find the grammar rules for each Chinese character. It is undoubtedly an infeasible approach.

After observing the Chinese character outline, we can conclude that a Chinese character outline can be segmented as strokes, and only seven classes is sufficient for classifying the strokes for the purpose of auto boldness (Appendix One). The stroke outline of a class has little variations among different characters. Consequently, defining the grammer rules for strokes is a feasible approach. The input character is firstly segmented as strokes, and then the strokes are pased individually in the shape pasing phase.

Strokes segmentation plays an important role in Chinese fonts auto boldness. But, segmentation is always expensive to be implemented in the conventional approach of image analysis, and it defeats the purpose of real time auto boldness. Thus, fast heuristic approach is adopted to segment strokes of Chinese character. The strokes of Chinese characters have the exceedingly impatent property that each stroke contains a sharp point. Thus, if the sharp points of a Chinese character are all found, the locations of the unknown strokes can be ascertained, and the unknown strokes can be easily segmented for pasing.

4.5 Shape Parsing Mechanism

As shown in fig 4.5, in the phase of Shape Lexical Analysis, the sharp point tokens are identified. Thus, the unknown strokes in a glyph outline can be located by the sharp point tokens because the existence of a sharp point implies that a stroke is possibly present at that position. The purpose of the shape parser is to use the stroke grammar rules to parse the outline near the sharp point so that the class of stroke can be identified, and the key points of the stroke can also be found. Key points identification is very crucial to the phase of code generation.

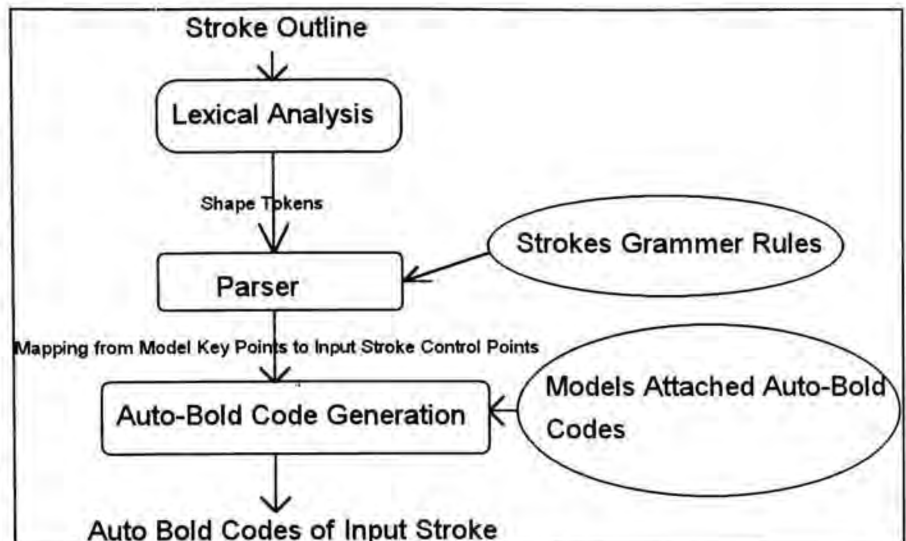


Figure 4.5: Shape Parsing Data Flow

Each class of stroke is associated with a set of shape grammar rules. An input stroke is identified as class c stroke, if and only if it can be parsed by using the set of shape grammar rules associated with class c. Therefore, the input stroke is parsed by using the shape grammar rules of all classes, until a class of stroke parse tree can be built successfully, then the stroke is classified to that class of stroke; otherwise the stroke is unclassified. Furthermore, the search range of stroke classes can be narrowed down by evaluating the attribute *class* of **sharp** token.

During the shape parsing, the key points in the stroke outline are identified. The outline of a stroke class can be varied slightly, but it can be modeled by a stroke model. The stroke model can be viewed as an abstract stroke outline with key points on it. Key points are points on the outline to divide the abstract stroke outline into several curve segments, and they can be an inflexion points, at which a concavity changes, or local maximum points in x, y coordinate, or global maximum points in x,y coordinate etc.

The auto bold program is attached to the stroke model. In the shape parsing phase, a mapping from the stroke model key points to the input stroke control points is built. Thus, the auto-bold codes for the input stroke can be generated by replacing the key points in the auto-bold program of the stroke model with the control points of the input stroke.

4.6 Model Grammar Rules

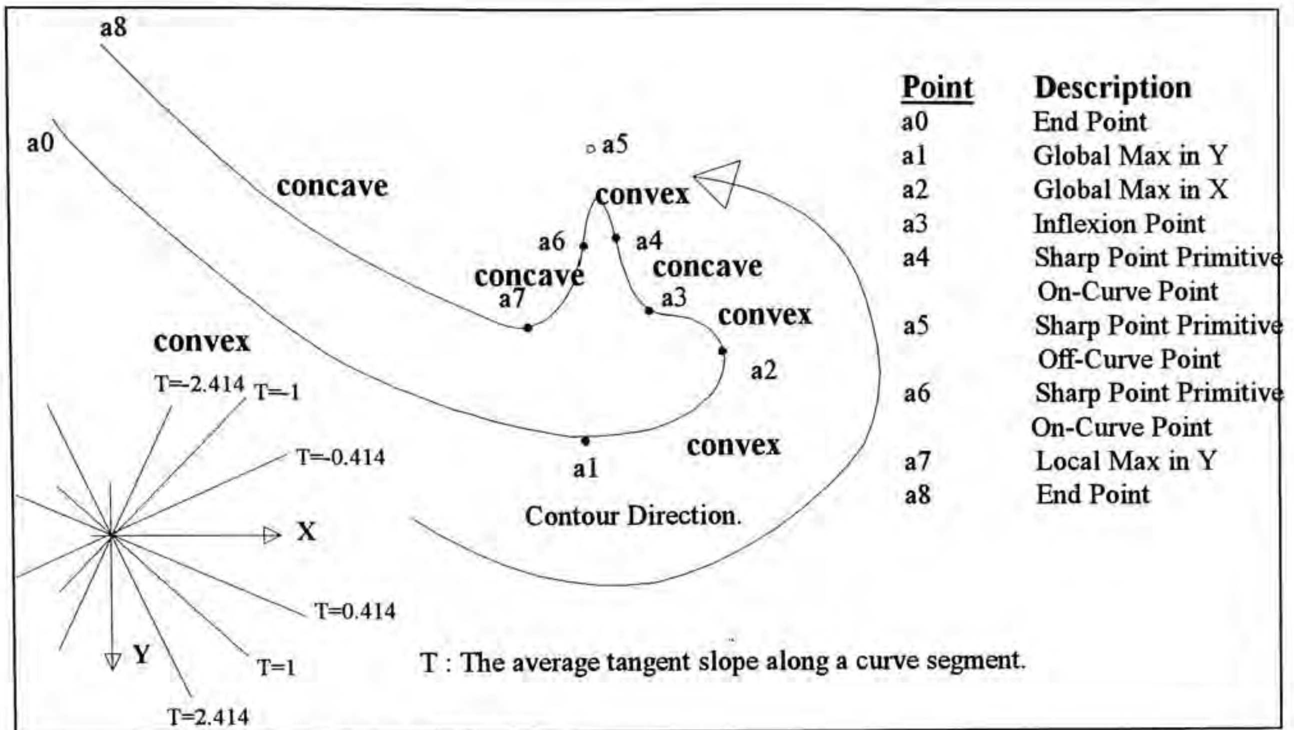


Figure 4.6: Model of Right Hoke

Model Grammar Rules for Right Hoke

R001: <RightHoke> :- <PrevCurve> ***sharp*** <NextCurve>

Assignment:

1. a4 = first control point of the quadratic bezier of **sharp** token.
2. a5 = second control point of the quadratic bezier of **sharp** token.
3. a6 = third control point of the quadratic bezier of **sharp** token.

Condition:

1. **sharp**.*class* = 3
2. r1 = (<PrevCurve>.length - <NextCurve>.length)/<PrevCurve>.length
   s.t. 0 < r1 < 0.3

R002: <PreCurve> :- <ConvexJointCurve1>***convex1***
convex2*concave3

Assignment:

1. a0 = first control point of <ConvexJointCurve1> joined curve segment.
2. a1 = first control point of **convex1** curve segment.
3. a2 = first control point of **convex2** curve segment.
4. a3 = first control point of **concave3** curve segment.

Condition:

1. Discontinuous at a0 AND Continuous at a1 AND Continuous at a2
   AND Continuous at a3
2. a1 is global maximum in y-coordinate.
3. a2 is global maximum in x-coordinate.
4. a3 is a turning point.
5. 1 < <ConvexJointCurve1>. τ < 2.414
6. -1 < **convex1.** τ < -0.414
7. 0 < **convex2.** τ < 0.414
8. vcurve3. τ > 2.414
9. (<ConvexJointCurve1>.length > **convex1**.length AND **convex2**.length AND **concave3**.length.

R003: <NextCurve> :- **concave1***<ConcaveJointCurve1>

Assignment:

1. a7 = last control point of **concave1** curve segment.
2. a8 = last control point of <ConcaveJointCurve1> joined curve segment.

Condition:

1. Continuous at a6 AND Continuous at a7 AND Discontinuous at a8
2. a7 is local maximum in y-coordinate.
3. **concave1.** τ < -2.414
4. 1 < <ConcaveJointCurve1>. τ < 2.414
5. <ConcaveJointCurve1>.length > **concave1**.length

R004: <ConvexJointCurve1> :- **convex2**+<ConvexJointCurve3>

Assignment:

1. <ConvexJointCurve1>.length = **convex2**.length + <ConvexJointCurve3>.length
2. <ConvexJointCurve1>. τ = (**convex2**.length***convex2**. τ + <ConvexJointCurve3>.length*<ConvexJointCurve3>. τ)/(**convex2**.length + <ConvexJointCurve3>.length)

R005: <ConvexJointCurve1> :- **null**

Assignment:

1. <ConvexJointCurve1>.length = 0
2. <ConvexJointCurve1>. τ = 0

R006: <ConcaveJointCurve1> :- **concave2**+<ConcaveJointCurve3> |
Assignment:
1. <ConcaveJointCurve1>.length = **concave2**.length + <ConcaveJointCurve3>.length
2. <ConcaveJointCurve1>. τ = (**concave2**.length***concave2**. τ + <ConcaveJointCurve3>.length*<ConcaveJointCurve3>. τ)/(**concave2**.length + <ConcaveJointCurve3>.length)

R007: <ConcaveJointCurve1> :- **null**
Assignment:
1. <ConcaveJointCurve1>.length = 0
2. <ConcaveJointCurve1>. τ = 0

The first grammar rule for a stroke is always in this form:
R001: <Stroke> :- <PrevCurve> * **sharp** * <NextCurve>
The terminal **sharp** represents the sharp point of the stroke. The non-terminal <PrevCurve> and <NextCurve> represent the previous curve segment of sharp point and next curve segment of sharp point, respectively. The notation * is a relational operator to connect two joined curve segments in the rule body. The outline of a stroke can be divided into two parts. Part one is the curve segment near the sharp point, and it can be assumed to be complete. Part two is the curve segment far away from the sharp point, and it will be incomplete due to stroke interception. Thus, a relational operator + is invented to deal with the incomplete curve segments. As a result, when the rule **R004** and **R006** is matched against input stroke outline, edge tracking to join the unconnected primitives is actually performed.

The shape grammar rule is attached by a set of assignment statements and condition statements. The rule is fired only if the condition statements are all statisfied. The assignments build up the mapping from the model key points to the input stroke control points. Each stroke class has a predefined auto-boldness program. The auto-boldness program is written by using the auto-boldness language defined in the previous chapter. Moreover, the input parameters to call the auto boldness instructions are the key points of that stroke class. Thus, in the code generation phase, the effort required is only to replace the key points in the model auto boldness program with the input stroke control points by using the mapping established in the parsing phase.

4.6.1 Grammar Rule Format

The general format of a Model Grammar Rule is:

R***: H :- a(1) op a(2) op.....op a(n)

where R*** is the label of the grammar rule, and H is the head of grammar rule, and H is always a non-terminal. a(1) op a(2)...op a(n) is the body of grammar rule. a(i) is either a terminal (shape token) or non-terminal. op is a relational operator, and it can be either * or +.

The relation a(i) * a(i+1) is established if the following rules are all satisfied.

1. a(i) and a(i+1) are both the curve segments of the same contour of a glyph outline.

2. p1 is the last control point of a(i), and p2 is the first control point of a(i+1), Then, p1 = p2, that is p1.x = p2.x AND p1.y = p2.y

The relation a(i) + a(i+1) is established if the following rules are all satisfied.

1. a(i) and a(i+1) are the curve segments of a glyph outline.

2. p1 is the control point before the last control point of a(i)
p2 is the last control point of a(i)
p3 is the first control point of a(i+1)
p4 is the control point after the first control point of a(i+1)
Then, p1,p2,p3 and p4 are roughly collinear.

3. p2 and p3 are not the same point that is (p2.x != p3.x) or (p2.y != p3.y)

3. distance between p1,p2 < distance between p1,p3 < distance between p1,p4

4. The distance between p2 and p3 is less than a threshold value (Stroke Width).

Both the relational operator * and + are non-transitive, non-reflexive, non symmetric. The relational operator * deals with the complete curve segments are not intercepted by another stroke, but the relational operator + deals with the incomplete curve segments owning to stroke interception. In the example of Right Hoke, the curve segments from key point a1 to a7 are not supposed to be intercepted by other stroke. Thus, these curve segment are connected by using * operator in grammar rule body. However, the curve segments from key point a0 to a1, and the curve segments from key point a7 to a8 are supposed to be incomplete because of stroke interception. Thus, these curve segments are connected by using + operator in grammar rule R004 and R006 body.

4.6.2 Grammar Rule Item

The basic item of a grammar rule is either terminal or non-terminal. Terminal is always the shape token. There are two levels of shape tokens. Level one shape token is a primitive, which is either a straight line or a quadratic bezier, in a glyph outline. Level two shape token is more than one primitive. **concave** token is the level two primitive to represent a concave continuous curve segment, and **convex** token is also the level two primitive to represent a convex continuous curve segment.

Non-terminal is denoted as <something>, which is a curve segment, and can be derived to a sequence of terminals by using the grammar rules. Moreover, a non-terminal can have some attributes to describe its characteristics.

For example, <ConvexJointCurve>.length is the curve length.
<ConvexJointCurve>. τ is the average slope of the tangents along the curve segment.

In the example of Right Hoke, the grammar rule items are described by the following table:

Grammar Rule	**Item**	**Description.**
R001	<RightHoke>	The outline of a Right Hoke
R001	<PrevCurve>	Curve segment from a0 to a4
R001	sharp	a quadratic bezier with sharp point control points (a4,a5,a6)
R001	<NextCurve>	Curve segment from a6 to a8
R002	<ConvexJointCurve1>	a convex continuous curve segment from a0 to a1. It may have some parts missed owing to stroke interception
R002	**convex1**	convex curve segment from a1 to a2
R002	**convex2**	convex curve segment from a2 to a3
R002	**concave3**	concave curve segment from a3 to a4
R003	**concave1**	concave curve segment from a6 to a7
R003	<ConcaveJointCurve1>	a concave continuous curve segment from a7 to a8. It may have some parts missed owing to stroke interception

4.6.3 Grammar Rule Assignment

The assignment statements are executed before the condition statements are checked. The purpose of grammar rule assignment statements is to calculate the attributes of the head non-terminal, and to build up the mapping from the model key points to input stroke control points.

4.6.4 Grammar Rule Condition

The condition statements are checked after the assignment statements are executed. The grammar rule is fired, only if the condition statements are all satisfied.

4.7 Auto Boldness Code Generation

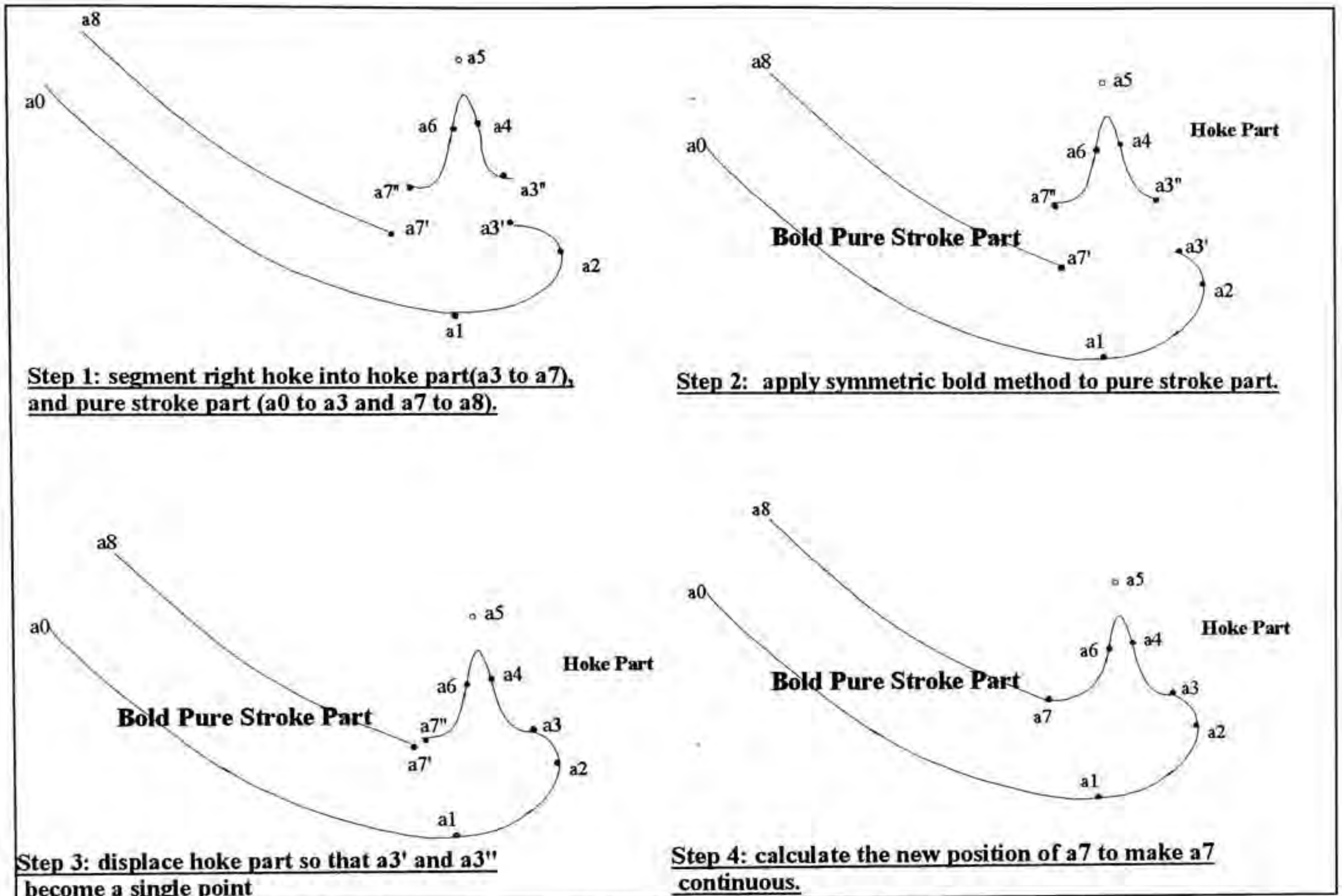


Figure 4.7: Steps to Generate Auto-Bold Code for Right Hoke

The key points of a stroke model divide the stroke outline into many curve segments. Then, the auto-bold codes for the stroke model can be defined by using the key points. In the phase of shape parsing, the stroke model and the mapping from the stroke model key points to the input stroke control points are found. Thus, the actual auto-bold codes for the input stroke can be generated by replacing the key points in the model auto bold codes by the control points of the input stroke.

The model auto bold codes for right hoke stroke are listed as follows:

Step1: Segment the Stroke into Pure Stroke Part and Hoke Part

Pure stroke is the curve segment from a0 to a3', and a7' to a8. The hoke part is from a3" to a7". It is noted that the key point a3 is split into a3' and a3", and the key point a7 is split into a7' and a7". a3" and a7" belong to the hoke part, and a3' and a7' belong to the pure stroke part.

Step 2: Apply Symmetric Bold Method to the Pure Stroke Part

The pure stroke part is composed by a pair of symmetric curves.

```
Generate Code:
Curve1 = [a0,a1,a2];
Curve2 = [a8,a7',a3',a2];
SymmetricBold(Curve1,Curve2,BoldLevel);
```

Step 3: Translate the Hoke Part so that a3' and a3" become a single point

```
Generate Code:
dx = a3'.x - a3".x;
dy = a3'.y - a3".y;
For(point=a3"; point <= a7"; point++) {
   point.x += dx;
   point.y += dy;
};
```

Step 4: Calculate the new position for point a7

```
Generate Code:
slope1 = slope of tangent at a7' along the pure stroke outline;
slope2 = slope of tangent at a7" along the hoke part outline;
Line 1 is the line passing through a7' with slope = slope1;
Line 2 is the line passing through a7" with slope = slope2;
p = interception point of Line 1 and Line 2;
a7' = a7" = p;
```

4.8 Program Methodology of Prototype Auto Boldness Driver

The auto-boldness driver developed by us is only a prototype. In order to fulfill the purpose of demonstrating the feasibility of Chinese font auto-boldness, we have abandoned the approach to develop the auto-boldness expert system shell to interpret the shape grammar rules Instead, the shape grammar rules are hard-coded as procedural knowledge. To enhance the efficiency, only the complete curve segments near to the sharp point of the input stroke are matched against the grammar rules. After the class of the input stroke is found, the edge tracking algorithm is invoked to join the incomplete curve segments of the input stroke.

Chapter 5: Conclusions

5.1 Work Achieved

The auto boldness of TrueType Chinese fonts was hardly ever found in the font market at the start of our project in July, 1992. Some font vendors were doing research on this topic secretly, and it was very hard to get any idea in Chinese fonts auto boldness from past publishing papers. The most valuable resource for us was only the TrueType font book of Microsoft Company.

Moreover, the existing methods of auto boldness is not desirable at all. Double Printing can only attain a slight increase of blackness, and Adobe's Multi-Master requires too much memory, unsuitable for Chinese fonts with large characters set. Thus, we aims at catering the deficits of existing methods to invent a new algorithm for TrueType Chinese fonts auto boldness. The ideas come out in Dec, 1992, and a prototype auto boldness driver has been developed. The algorithm is fast and efficient, supporting multi-level of boldness.

Our ideas of Chinese fonts auto boldness are originated from the paper [Percept92]. The main idea of this paper is to segment the objects from a scenery by firstly locating the symmetries because the boundary of objects are always composed of symmetric curves pair. Human visual system can locate and segment the objects from a scenery fastly because human eyes are very sensitive to symmetries. Similarly, the strokes of Chinese characters are also highly symmetric of their boundaries. Furthermore, we found a general method, which is symmetric bold, to bold a pair of symmetric curves with multi-level of boldness. Thus, if the symmetries of Chinese character outline can be located and segmented automatically, the problem of auto boldness will be solved. Chinese character has a distinct property that each stroke must contain at least one sharp point. Therefore, strokes can be located by finding the sharp points from the character outline, then the symmetries can be easily found.

Finally, it is encouraging that our paper entitled "TrueType Chinese Fonts" was published in Proceedings of the 1994 International Conference on Computer Processing of Oriental Languages on May 10-13 Taejon, Korea (ICCPOL'94).

5.2 The Pros and Cons of Auto Boldness Algorithm

Pros

a. Parametric Approach:

A parameter BoldLevel, which is a rational number between 0.0 to 1.0, directly controls the level of boldness. The larger the value of BoldLevel is, the higher of the bold level is. Thus, a thousands of bold versions with different bold levels can be generated from an unbold master automatically.

b. Efficient Stroke Extraction and Classification:

It is found that Chinese characters have an exceedingly impertant property that each stroke contains at least one sharp point. Therefore, the existence of a sharp point in a character outline implies the existence of a stroke. Searching of stroke locations by sharp points finding is very fast and efficient. After the strokes location of a character are found, strokes extraction and classification can easily be performed.

c. Auto boldness Language:

Standardized auto boldness primitives including **SymmetricBold**, **RotateBold**, and **AsymmetricBold**, are defined, and the auto boldness programs for each character can be built by using these primitives. Chinese character outlines are mainly composed of symmetric curves pairs and asymmetric curves pairs. The primitives **SymmetricBold** and **RotateRold** can bold symmetric curves pairs, and the primitive **AsymmetricBold** can bold asymmetric curves pairs. The algorithms to implement these primitives are general enough to deal with any Chinese outline fonts, in line with the principle of reusability in Software Engineering.

d. Serif Handling:

In order to handle the serifs during auto boldness, serifs must be segmented and classified, but the serif models for Chinese fonts are very complicated, and difficult to define. Thus, we invented a method to handle the serif nicely without knowing what the model of the serif is. The idea is that a direction vector can be defined for each serif, and the direction vector of a serif is the key to do serif handling. The direction vector of a serif can be easily found by using heuristics without knowing the exact model of a serif.

e. Shape Parsing Rule Format:

After the strokes of a Chinese character are located, the strokes will be further classified and segmented. The process of stroke classification and extraction is similar to a compilation process so that the parsing rules for each stroke model must be defined. The interception of strokes make it difficult for representing the parsing rules, because strokes interception will lead to incomplete outline of strokes. Generally speaking, the parsing rule format is H :- B where H is the head of rule, and B is the body of rule. H is a non-terminal. B is a conjunction of terminals and non-terminals. The terminals can be regarded as the basic elements of Chinese character outline, such as bezier primitives, straight line primitives, concave curve segments, convex curve segments, etc. The non-terminals can be regarded as higher level objects such as symmetric curves pairs, asymmetric curves pairs, hoke, serifs, etc. Two operators * and + are invented to relate two items in the rule body. The operator * can relate two joined curve segments of a stroke, and the operator + can relate two unjoint curve segments of a stroke, resulted from stroke interception. During the process of parsing, when a rule with the operator + is encountered, the parser will try to join the unconnected curve segments of the intercepted stroke by using edge tracking algorithm.

f. Auto Bold Code Generation:

After a stroke is parsed, a mapping from the key points of the stroke model to the input stroke control points will be found. Each stroke model is associated with an auto boldness program executing the auto boldness primitives with the key points as passing parameters of the primitives. The auto boldness program for the input stroke can be generated simply by replacing the key points with the input stroke control points in the stroke model's auto boldness program. This process is very fast and efficient.

Cons

a. Stroke Extraction:

Stroke interception leads to incomplete curve segments of a stroke. In order to find out the curve segments of a stroke from the whole character outline, the algorithm of edge tracking is used to join the unconnected curve segments. It is unavoidable to track erroneously in some cases. In our experiment, about five character out of one has stroke outline erroneously tracked.

b. Complexity of Character Outline:

In some cases, a Chinese character outline can be consisted of more than 300 control points. The bitmap of such character has a very small white space. If the character is bolded, the white space will disappear, resulting in a very ugly bitmap. Moreover, the continuity of the contour can hardly be preserved owning to the large number of control points.

c. Regardless of Subjective Factors:

After we tried to run our auto boldness driver on many Chinese characters, we found that the bold character outline would still lose some features. After all, the design of a character outline involves much human subjective point of view, and judgment of the quality of a character design is based on the character percepted by human. Nevertheless, our tools can only treat the objective factors, such as stroke width, serif dimensions, etc., but it does not count on the human subjective factors, and human perception.

5.3 Bold Quality Assessments

BoldLevel=0

BoldLevel=0.2

BoldLevel=0.4

BoldLevel=0.6

BoldLevel=0.8

BoldLevel=1.0

BoldLevel	Width	Height	MinX	MaxX	MinY	MaxY	Black	CG (x,y)
0	278	291	175	453	39	330	20034	(319,182)
0.2	278	291	175	453	39	330	22731	(318,183)
0.4	278	292	175	453	38	330	25476	(319,184)
0.6	278	293	175	453	37	330	28265	(318,184)
0.8	279	295	174	453	36	331	30805	(318,185)
1.0	279	298	174	453	35	333	33332	(318,185)

The character is bolded by the prototype auto boldness driver with different bold levels. Then, the metrics of the raster image are measured in pixel units. The dimensions of the raster image is 1000x1000, and the origin is at the upper left corner. The columns Width and Height are the dimensions of the bounding box. MinX and MaxX are the minimum and maximum x-coordinate of black area. MinY and MaxY are the minimum and maximum y-coordinate of black area. The column Black measures the number of pixels of black area. The column CG measures the center of gravity of the black area.

Thus, the following conclusions can be drawn:

a. Unchange in Bounding Box

The width and the height of bounding box were increased slightly with the increase of the parameter BoldLevel. Therefore, the bounding box was stable in dimensions during auto boldness.

b. Linear relationship of Black and BoldLevel

Let, B be the black area of bold image, B0 be the black area of unbold image (= 20034) and a is a constant

Then, $B = a.BoldLevel + B0$ (5.1)

Rearranging (5.1), we get $a = \frac{B - B0}{BoldLevel}$

BoldLevel	0.2	0.4	0.6	0.8	1.0
Black(B)	22731	25476	28265	30805	33332
a	13485	13605	13718.33	13463.75	13298

The value of a is stable, and the average value is 13514. Therefore, the parameter BoldLevel reflects the blackness of raster image very well.

c. Stable in CG

The CG of the raster image can be found from the formula:

$$x = \frac{\sum_i m_i . x_i}{\sum_i m_i}, \quad y = \frac{\sum_i m_i . y_i}{\sum_i m_i},$$

where m_i is a small area of the character image so that $\sum_i m_i$ is the total image area. (x_i, y_i) is the coordinate of the small area.

Now, we take each pixel as a basic element, and m_i = 1

Thus, $x = \frac{\sum_{i=1}^{N} x_i}{B}$ and $y = \frac{\sum_{i=1}^{N} y_i}{B}$ where N is number of pixels in black area (5.2)

From the column CG, it can be seen that the center of gravity is very stable during auto boldness. Therefore, weight is added evenly throughout the whole character, resulting in unchange in center of gravity.

5.4 Future Directions

a. Enhancements the TrueType Engine

Ideally speaking, TrueType Engine must be enhanced to support auto boldness instructions. So, auto boldness developers can develop the auto boldness program for a TrueType font based on a standardized set of auto boldness instructions, and the effort required can be reduced.

b. Additonal Bold Information in TrueType Font File

Some auto boldness information must be stored in the TrueType font file so as to realize a full set auto boldness. This is because a Chinese character outline is so complicated that some strokes can never be extracted, no matter how perfect the heuristic rules are. Strokes, which can not be extracted automatically, can be extracted manually, then the information of the extracted strokes can be stored in the TrueType font file. Hence, the auto boldness driver can make use of the auto boldness information in TrueType font file to bold the characters.

c. **Expert System Shell of Auto Boldness**

It is feasible to develop the expert system shell of auto boldness. The auto boldness instructions, parsing mechanism and code generation process can be implemented by the shell. Auto boldness developers only need to define the parsing rules of strokes. It can accelerate the progress tremendously, and the system will be more maintainable, because only the parsing rules needs to be modified.

References

1. [True 91] Book: The TrueType Book
(Draft Preliminary, Confidential, 1991 Apple Computer, Inc)
Addison-Wesley Publishing Company, Inc, .

2. [True 94] Paper: TrueType Chinese Font, Mr. Lo I Fan and Mr. Y.S.Moon,
Computer Science Dept. of Chinese University of Hong Kong
Published in Proceedings of the 1994 International Conference on Computer Processing of Oriental Languages (ICCPOL'94) on May 10-13 Taejon, Korea.

3. [Adobe 90] Document: Adobe System Inc., Adobe Type 1 Font Format, 1990.

4. [Adobe 92] Document: Adobe Type 1 Font Format - Multiple Master Extensions,
14 Feb 92.

5. [Final 92] Final Year Project Report: Research in Chinese Outline Font
Chan Chi Lok and Chiu Chong Kan, supervised by Dr. Y.S.Moon
Computer Science Dept. of Chinese University of Hong Kong, 1991-1992

6. [Knuth 86] The *METAFONT* Book. Reading, Mass.: Addison Wesley. A readable introduction to and manual for the *METAFONT* typeface description language.

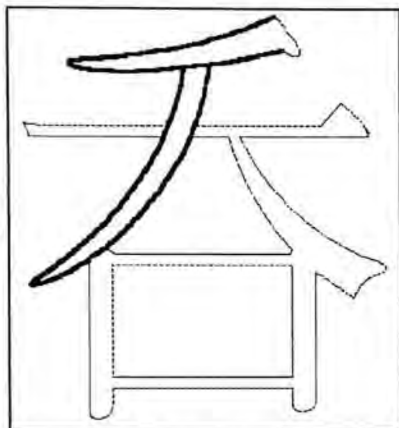
7. [Percept 92] Paper: Perceptual Organization for Scene Segmentation and Description
Rakesh Mohan, Member, IEEE, and Ramakant Nevatia, Fellow, IEEE
published in IEEE Transactions on Pattern Analysis and Machine Intelligence Vol 14,
1992.

8. [Model 91] Paper: Model-based Matching and Hinting of Fonts
Roger D. Hersch, Claude Betrisey, Swiss Federal Institute of Technology(EPFL)
CH-1015 Lausanne, Switzerland
published in Computer Graphics, Vol 25, 1991.

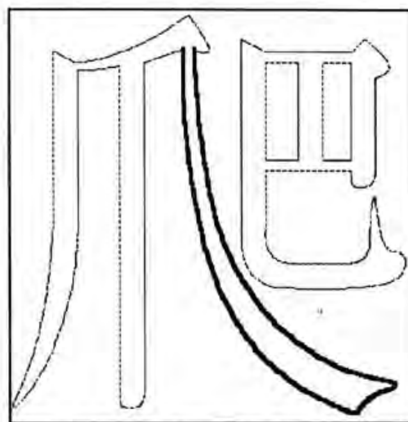
9. [Wife 91] Book: Windows Intelligent Font Environment(WIFE) Volume 1
Outline Specification Rev. 1.00a, Microsoft Corporation, Far East Product Development Group, 1991

10. [Shape 85] Paper: Shape Grammar Compilers
Thomas C. Henderson and Ashok Samal, Dept. of Computer Science, The University of Utah,Salt Lake City, Utah, U.S.A, published in Pattern Recognition Vol 19, 1985.

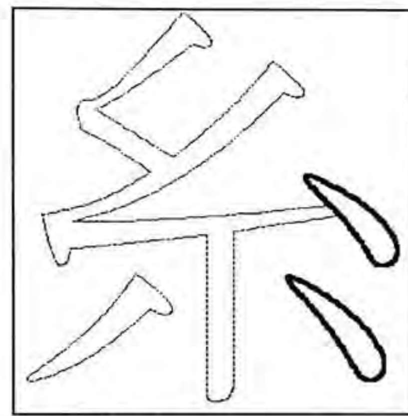
Appendix One: Stroke Classification for Auto Boldness



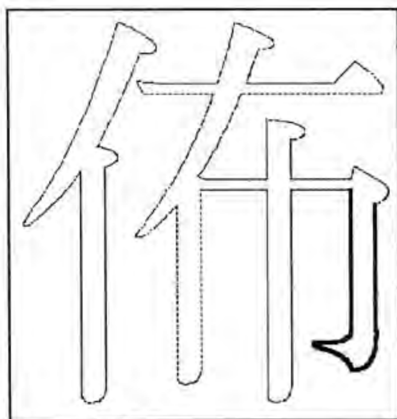
a. Left Incline Stroke



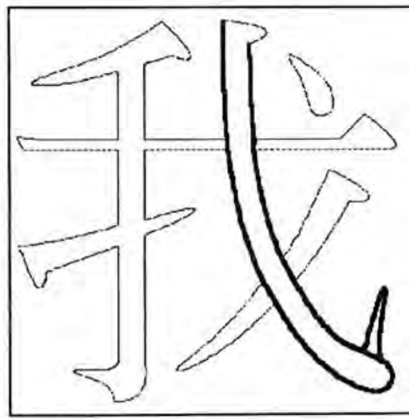
b. Right Incline Stroke



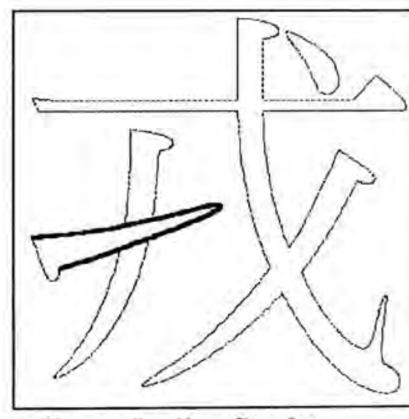
c. Dot Stroke



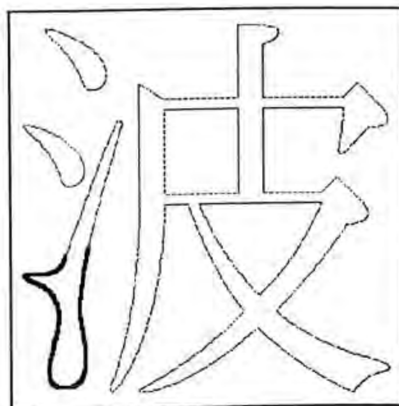
d. Left Hoke



e. Right Hoke



f. Upper Incline Stroke



g. Drop Stroke

Appendix Two: Experiment Results

BoldLevel = 0.0

BoldLevel=1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel = 0.0

BoldLevel = 1.0

BoldLevel=0.0

BoldLevel=1.0

沸

BoldLevel=0.0

沸

BoldLevel=1.0

定

BoldLevel=0.0

定

BoldLevel=1.0

東

BoldLevel=0.0

東

BoldLevel=1.0

成 成

BoldLevel=0.0 BoldLevel=1.0

旭 旭

BoldLevel=0.0 BoldLevel=1.0

多 多

BoldLevel=0.0 BoldLevel=1.0

戓

BoldLevel = 0.0

戓

BoldLevel=1.0

剋

BoldLevel=0.0

剋

BoldLevel=1.0

掏

BoldLevel=0.0

掏

BoldLevel=1.0

CUHK Libraries

000249284