# A Methodology for Constructing Compact Chinese Font Libraries by Radical Composition

by
Wai-Yip Tung

supervised by
Dr. Yiu-Sang Moon

A dissertation submitted in partial fulfillment
of the requirement for the degree of

Master of Philosophy
in
Department of Computer Science
The Chinese University of Hong Kong

June, 1993

UL

thesis
QA
76.9
I55T86
1993

# Abstract

The large size of the Chinese character set often results in font libraries of prohibitive size. We are looking for representations which reduce the storage of font libraries. The representations must also result in good-looking character images and give little hindrance to the efficiency of rasterization.

We compose character by radicals. Linear transformation is used to scale similar radicals. To avoid excessive scaling, which distorts the shapes of the radicals, several versions of each shape may have to be built. Decomposing characters by Bushou (部首) and Shengpang (聲旁) is useful as their sizes and shapes are more regular. The reusability rate of Bushou among the Chinese characters is estimated to be very high; the reusability rate of Shengpang is estimated to be moderate.

Hinting intentionally distorts an outline so that important features of a character are visible and uniform at all sizes. Our hinting algorithm ensures that horizontal stroke and vertical stroke width of the character image are uniform in all resolutions. More importantly, it can fix the distortions resultant from transforming the radicals.

# Table of Contents

# Acknowledgements

I would like to thank my supervisor Dr. Yiu-Sang Moon for his support and good ideas. I would also like to thank Chi-Cheung Poon and Hon-Man Law for the advice and valuable information they have given me.

# Chapter 1 Introduction

Text is the most important means of interaction between human and computer. We use computer to collect, store, process and output text. Word processing and desktop publishing are two important applications of computers. In modern graphical user interfaces, the demand of text is even higher. We are no longer satisfied with single styled characters fit in small dot matrix. We demand text with various fonts, styles and sizes, which are displayed gracefully and efficiently.

Building a Chinese font system is difficult because of the large number of Chinese characters. Unlike European languages which only contain small alphabets, the exact number of Chinese characters is hard to determine. It is estimated that at least 50,000 to 60,000 Chinese characters have ever been used [11].

Most of these 50-60 thousand characters are seldom used today. Daily used characters are a subset of them. Two coding systems for Chinese characters are commonly used today. The GB code used in China has a character set of about 7,000 characters, while the BIG-5 code commonly used in Taiwan has a character set of about 13,000 characters. Although the GB character set has about half the number of characters as in the BIG-5 character set, its size is still very large comparing to Latin alphabets.

A typical Chinese outline font of the BIG-5 character set may consume as much as 5 megabytes of storage. If several typefaces are needed, the storage will increase many times. The storage constraint is even tighter in portable computers, which are already short of disk space.

Although the number of Chinese characters is very large, they are composed by a small number of components, called radicals. We may reduce the storage size of the font by making use of the commonness between the characters. Basically, we compose characters by radicals. Linear transformation is used to scale similar radicals. To avoid excessive scaling, which distort the shape of the radicals, several versions of the shape may have to be built.

## 1.1. Previous work

### 1.1.1. A Chinese METAFONT

Hobby and Gu [7] have used METAFONT [2] to build Chinese characters. METAFONT subroutines are built to generate basic strokes, which are then used to compose characters. A few parameters are used to describe the skeletons of the strokes and to transform the strokes.

Only 14 routines are used to represent the basic strokes. Elaborate efforts have been spent to build each of these basic stroke routines. Each type of stroke is controlled by complicated parameters, which include specific parameters for that kind of stroke and global parameters for the overall design. For instance, there are as many as 68 global parameters for Song style font.

Having such a complicated set of parameters is obviously a difficulty in font design. Instead of crafting the character shape graphically, we have to fine tune the parameters of each character which is unnatural for type designers.

### 1.1.2. Chinese character generator

[1] represent characters by their skeletons (stroke paths). The skeletons are extracted from scanned images by thinning. Character is generated by drawing along the skeleton with a pen.

The skeleton extraction method in [1] is unsatisfactory. Thinning results in dented skeletons. Skeletons are often distorted in stroke intersections which deviate from the natural notion of stroke paths. Besides, characters generated from stroke path have uniform stroke widths all over. Slant strokes, which usually have non-uniform widths, and serifs are difficult to describe using stroke path representation.

### 1.1.3. Chinese Character Design System CCDS

CCDS [3] is a METAFONT style font compiler. It is the third generation font compiler from LCCD in 1981 [4]. Font data is represented in CDL (Character Design Language) which is processed by the CCDS font compiler. CCDS can produce three kinds of font data, including bitmap, polygonal outline and curve outline.

Characters are composed by radicals (sub-characters), which are further decomposed until basic strokes are obtained. However, the authors did not mention the method to transform these basic strokes and radicals.

The problem of the font compiler approach is that it is batch-oriented. It is too slow to generate characters at real time. However, the authors emphasize that a font compiler's task is for font design and character generation is a separate issue.

## 1.2. Goals of the thesis

There are three goals we want to attain in building Chinese font libraries. The first goal is to find methods that reduce storage.

Secondly, we are looking for methods that produce good quality fonts. Simple methods, like vector representation, may require only small amount of storage. However, such reduction of storage is often a damaging trade off for font quality. A good font representation should impose little restriction on the design of character shapes.

Thirdly, we are looking for a method that could lead to efficient output. That is the font should be easily decoded, scaled and rasterized. Complicated calculations and algorithms that slow down font output should be avoided. Since text output is a major activity of computer systems, efficiency of font output has a direct impact on the system performance.

## 1.3. Overview of the thesis

Chapter 2 gives a brief review on the construction of Chinese characters and the structural rules of Chinese characters. On the basis of Bushou-Shengpang decomposition, an estimation on the reusability rate of radicals in Chinese character is also presented in chapter 2.

Chapter 3 discusses our scheme to compose characters by radicals. We use linear transformation to scale the radicals. Depending on the structures of the character composed, we may need to build a few variant shapes for a radical to avoid distortion in excessive scaling.

In chapter 4, our hinting algorithm for Chinese font is presented. This hinting algorithm improves the quality of character bitmaps produced at different resolutions. Hinting information is generated automatically before rasterization. Chapter 4 also discusses how this hinting algorithm compensates for distortion due to scaling the radicals.

Chapter 5 presents the functions of our Chinese font editor RADIT. The font editor is built to experiment with the composition by radical scheme.

Finally, conclusions are given in chapter 6 that summarize the results of this piece of work.

# Chapter 2
# Construction of Chinese Characters

## 2.1. Introduction

Chinese script is one of the oldest writing systems in the world. There are other ancient writing systems like the Sumer glyphs, Egyptian hieroglyphs, Creta and Indus scripts but they have long perished. Only Chinese script has survived through history. Today it is not only widely used in China, it has also spread to neighboring countries like Korea and Japan where Chinese characters are mixed with their own scripts.

Through the 4,000 years of evolution, the form of Chinese characters has changed. New characters have also been created from time to time. Knowledge of the evolution and construction of Chinese characters is very important for font making. Many studies have been made on Chinese characters during the last 2,000 years. In addition to these classic materials, modern quantitative analyses also enlighten the efficient design of a Chinese font system.

## 2.2. *liu shu*(六書) Six Principles of Chinese Character Construction

As early as 2,000 years ago, people began to study the construction of Chinese characters. They concluded that there are 6 principles of character construction:

| | |
|---|---|
| ***xiangxing*** (象形) | Pictogram. This is the most primitive method where simplified pictures are used to represent concrete objects. |
| ***zhishi*** (指事) | Ideogram. Characters created with a pointing sign which suggest the meanings of the characters. |

| | |
|---|---|
| ***huiyi*** (會意) | Compound ideogram. Two or more existing characters are arranged together to form a new character. The meaning is implied from its components. |
| ***xingsheng*** (形聲) | Phono-ideogram. This is the most common method of character construction. A new character is formed by two components. The *xingpang* (形旁) or radical component suggests the category of the meaning of the character, and the phonetic component *shengpang* (聲旁) indicates its pronunciation. |

Finally, the last two principles are not really rules to create new character shapes. ***jiajie*** (假借) borrow existing character shapes to represent words with similar pronunciation. ***zhuanzhu*** (轉注) create new shapes by the same rule as xingsheng. They are only different by the motivation of creating new characters.

According to [17] an early analysis of Chinese characters was made in a classical Chinese literature "*shou wen jie zi*" 《說文解字》 written in 121A.D. by Xu Shen. For the 9353 characters referred to in it, 264 of them are *xiangxing*, 129 are *zhishi*, 1260 are *huiyi* and 7700 are *xingsheng*. *xingsheng* is clearly the prevalent character making method. After the time of "*shou wen jie zi*", most new characters created are also *xingsheng* characters. That makes it constitutes 90% of contemporary Chinese characters.

*xiangxing* and *zhishi* are the oldest character construction methods. These primitive methods have limited power and it is difficult for them to represent abstract ideas. Therefore *xiangxing* characters and *zhishi* characters are rare in Chinese. Nevertheless, many *huiyi* and *xingsheng* characters are composed by *xiangxing* and *zhishi* characters. Although small in number, they are really the basic building blocks of Chinese characters.

Structurally, Chinese characters may be divided into two classes. *huiyi* and *xingsheng* characters are *composed characters*. They can be separated into two or more components (e.g. 相 is composed by 木 and 目, 忍 is composed by 刃 and 心). *xiangxing* and *zhishi* characters are *singleton characters* which cannot be separated (e.g. 木, 目, 刃 and 心). Of course there are always exceptions. Some *huiyi* and *xingsheng* characters made in ancient time have eventually evolved into singleton characters in modern writing.

It can be concluded that by far, the majority of Chinese characters are composed characters. They can be divided into singleton components or simpler composed components. Ultimately they are composed from a small number of building blocks that cannot be further divided.

In making a Chinese font, it is ideal if we can draw only a small number of basic building blocks and create other characters by transforming and composing these building blocks. Such design is expected to result in a very small Chinese font library. Practically, this optimal approach can only produce characters that look too childish to be used because the shapes of basic radicals are too diverse among the characters which can hardly be transformed by only one copy of radical template. Our approach is to make several version of shapes as necessary so that we don't have to stretch a radical to a shape that it doesn't fit.

## 2.3. Structural Analysis of Chinese Characters

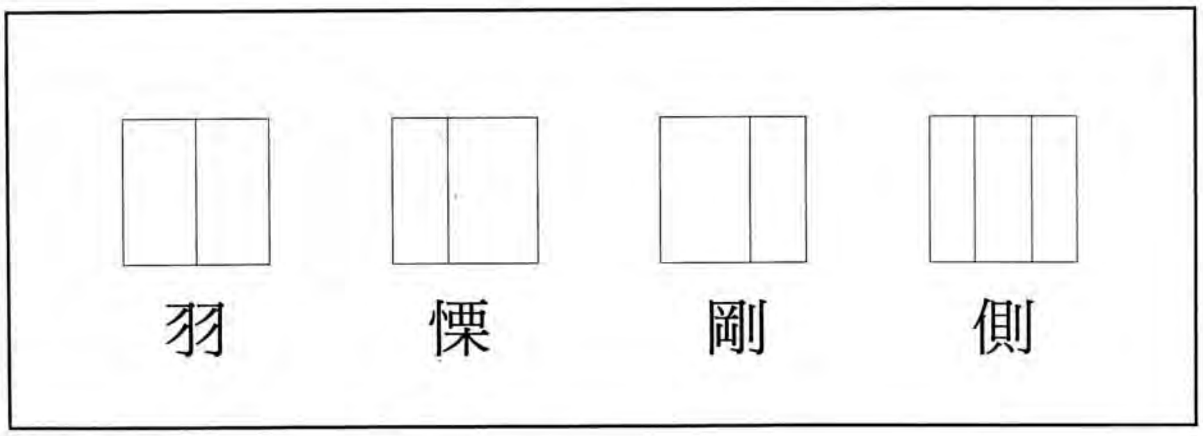
Structural analysis is the study of geometric relations among radicals in a character. The radicals are not combined into a character arbitrarily. There are rules that govern the composition of Chinese characters. For example, "提" is composed as left-right structure; "筒" is composed by top-bottom structure; "圓" is composed by inside-outside structure and "日" is a singleton structure that can not be separated. The left-right structure, top-bottom structure, inside-outside structure and singleton structure are the four basic structures of Chinese characters. These structure rules are not limited to top level only. They can be applied to the components recursively. For example, the right hand side component of "提", "是", itself is top-bottom structured. The bottom part of "筒" "同" itself is inside-outside structured. "員" is the inside of "圓" and "員" itself can be decomposed into top-bottom structure.

Among the several thousand Chinese characters, some are simple characters with small number of strokes, some are complicated characters composed with several radicals and large number of strokes. Nevertheless all characters must fit into a square. A good-looking character must look stable. Its strokes should position evenly in the square, not condensed to either side. It should look harmonious with other characters. This means characters should appear the same size and well aligned on a line. Their color (blackness) should look uniform. There shouldn't be a great contrast of blackness among the other characters. That means the structure of characters should be carefully arranged. It is often necessary to transform a radical, elongating or contracting it, to fit in the character. For example, in "堯", "堡" and "境" the radicals "土" are in different sizes and shapes to accommodate the other radicals and to achieve overall uniformity within each character. Every radical interacts with its partners according to their shapes and weights of their strokes in order to make their combined structure look balanced and harmoniously positioned.

### 2.3.1. Left-Right Structure

Left-right structured characters are composed by a left component and a right component, or sometimes, by left-middle-right components. Left-right structure is the most common structure which amount to 69% of all Chinese characters [18]. Some examples of left-right structured characters are shown below.

Figure 2.1



Left-right structured characters

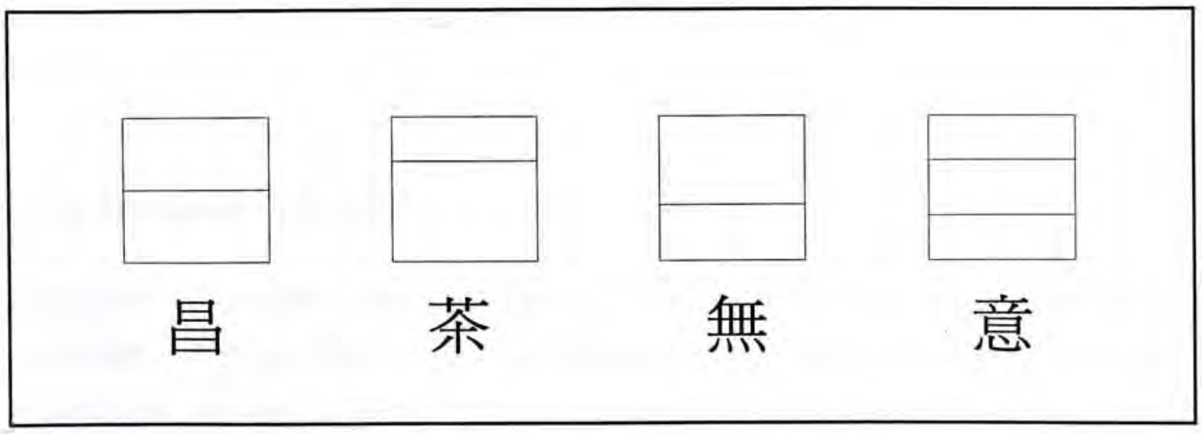
The relative proportions of the components depend on their complexities and stroke density in a character. If the left component is simple and the right component is more complex, then the left component may occupy 1/3 of the width while the right component occupies the other 2/3. On the other hand, if the left component is more complex than the right one, then it may occupy 2/3 of the width while the right component occupies the other 1/3. If they are of similar complexity, then the square should be divided into halves. Characters composed by three components should be arranged by the same principle so that the strokes are evenly distributed.

Note that some components are specifically used on the left or on the right (e.g. 扌, 氵, 亻, 釒 on the left; 攴, 攵, 欠, 刂 on the right etc.) The font maker only needs to make one shape unless the sizes of the component in composed characters are so diverse that more shapes have to made. Some radicals which can be used in different positions are sometimes built in slightly different shapes. For example, the two "木" components in the character "林" are slightly different, and the two "克" components in the character "兢" are also different. The strokes of the left component have changed so that the ultimate character is not too congested and the left component does not cut into the right component. This implies the font makers cannot blindly use the same shapes in all cases. They have to make extra copies of the radical to draw the altered shapes.

### 2.3.2. Top-Bottom Structure

Top-bottom structured characters are composed by two or more components divided vertically. Twenty percents of Chinese characters are top-bottom structured [18]. Again, the components are arranged according to their relative stroke density. If two components have similar density, they are arranged into half and half. If one component is more complex than the other, they are arranged into 1/3 to 2/3 in height or 2/3 to 1/3 in height. Some examples of top-bottom structured characters are shown below.

Figure 2.2



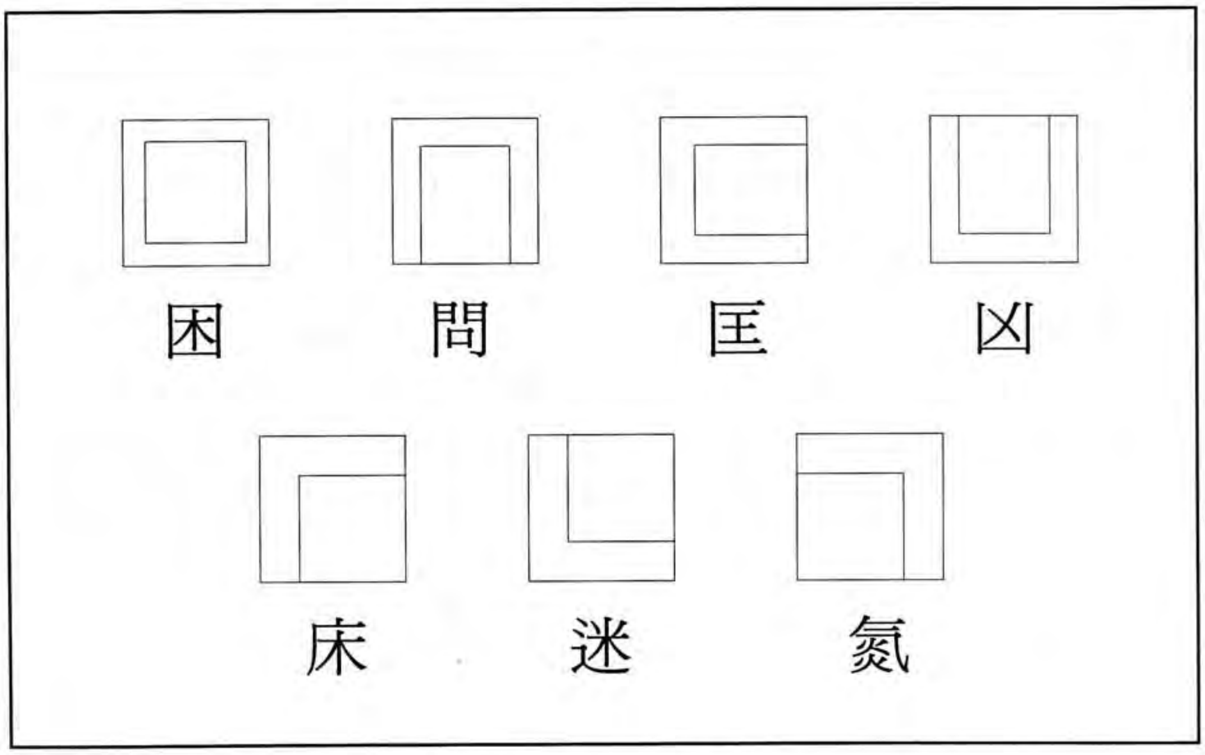
Top-bottom structured characters

Since the size and proportion of a top-bottom divided component is different from that of a left-right divided component, generally two versions of a radical have to be make if it appears in both structures.

### 2.3.3. Inside-Outside Structure

Inside-outside structure is the third most frequently used structure. There are many variations of the inside-outside structure. It can be a completely enclosed, closed on three sides, or closed on two sides structure. Figure 2.3 show the variations of inside-outside structured characters. Generally, the outside envelopes of these characters are more regular in size and shape so that they can be easily reused. The sizes and shapes of the inside components are more diverse since they have to be accommodated within their envelopes.

Figure 2.3



Inside-outside structured characters

### 2.3.4. Singleton Structure

Singleton characters are stand-alone shapes that have no components. Therefore we have little chance to share their shapes. Fortunately, singleton characters are usually simpler than composed characters and contribute less to the overall storage of the font.

## 2.4. Usage frequency of radicals

It is know that the majority of Chinese characters are composed characters. This is the basis of our font composition approach. We need also to find the usage frequency of radicals. How many radicals are needed to compose all characters? The usage frequency of radicals directly affects the storage reduction of the composed font.

To count the usage frequency, we do not just blindly decompose every character and count the radicals used. Instead, we choose to follow the traditional method of decomposing a character into Bushou (部首) and Shengpang (聲旁).

Bushou is the radical of every character that is used as an index in a Chinese dictionary. Since the time of one of the earliest Chinese Dictionaries *"shou wen jie zi"*, Bushou has been the prevailing indexing system.

We use the term Shengpang in the sense as in [16] to mean the part of the character other than Bushou. This is not exactly the same meaning as the Shengpang in xingsheng characters, though in most cases they do correspond. After all, not all Chinese characters are xingsheng characters. However, we still prefer to use the word Shengpang for convenience.

## 2.5. Usage frequency of Bushou

The number of Bushou is rather small. In "*shou wen jie zi*", 9353 characters are classified into 540 Bushous. The classification of Bushou is never standardized. More recently, some influential dictionaries like "*zi hui*"《字匯》, "*kang xi zi dian*"《康熙字典》 and "*ci hai*"《辭海》 use 214 Bushous. Contemporary dictionary like "*xin hua zi dian*"《新華字典》 use only 189 Bushous. Although the numbers are different, we can deduce that the number of Bushous is very small (200-500), comparing to the number of Chinese characters (around 50000).

In making a composed font, we are not only interested in the number of Bushous, we are interested in how many shapes are actually needed to compose the Bushou part of all characters. The correspondence of Bushou and shapes is not one to one, and we may need several shapes to represent one Bushou. Variations of shape are needed for one of these reasons:

1. The Bushou appears in one or more completely different forms. For example, the Bushou "心" is in "忄" shape when the character is left-right structured and in "⺗" shape when the characters is top-bottom structured. Other example include Bushou "水" which appear as "氵" and "氺"; Bushou "火" which appear as "火" and "灬".

2. A Bushou can appear in different shapes in characters of different structures. Generally, each kind of structure needs one variant shape. For example, two variant shapes of "石" are needed to compose left-right structured "硞" and top-bottom structured "碧".

3. The size of a Bushou may vary significantly in characters having the same structures so that more than one shape is needed to avoid serious distortion in scaling the Bushou. For example, the size of the radical "言" varies significantly between characters "詈" and "讐".

4. The shape of Bushou can be altered in a character or is not separable from other components in a character. Bushous are not always a separable component of the characters. Singleton structured characters are also indexed by Bushou. For example, the Bushou "木" has joined with the other component in character "末" to form a singleton structured character.

The degree of reusability of Bushou shapes is influenced by two factors, first the total number of all variant shapes used to compose the character set, and second, the number of odd characters that cannot be composed by separable Bushou shape. Fortunately, this class of characters is small. Besides, singleton structured characters are usually simpler characters, which have less contribution to the overall storage.

The estimated reusability of Bushou shapes is very high. For example, for the 257 characters composed by Bushou "金", only four shapes are needed to represent them (one shape for 244 left-right structured characters, 2 shapes for 12 top-bottom structured characters, and one character is not separable). Therefore the overall reusability of Bushou is estimated to be very high.

## 2.6. Usage frequency of Shengpang

The material that we used to analyse the frequency of occurrence of Shengpang is a Shengpang dictionary [16]. It is a unique source because it arranges 7,000 Chinese characters by their Shengpangs.

The total number of characters referred in [16] is 7272. Among them, 6542 are composed characters, 545 are basic Shengpangs (see the classification of Shengpang below) and 185 are *solitary characters* (孤獨字), which cannot be a Shengpang of other characters and no other character share the same Shengpang with them.

| | |
|---|---|
| Composed characters | 6542 |
| Solitary characters | 185 |
| Basic shengpangs | 545 |
| Total | 7272 |

Table 2.1: Distribution of Chinese characters

The dictionary [16] has adopted 1348 Shengpangs, which are divided into two classes, of which 1172 are proper characters themselves and the other 176 are not proper characters. The 1172 proper Shengpangs are further divided into two classes, 627 *composed Shengpangs* (滋生聲旁) which are themselves composed by other Shengpangs, and 545 *basic Shengpangs* (基本聲旁) which cannot be composed by any other Shengpang.

| | | |
|---|---|---|
| Proper Shengpang | basic Shengpang | 545 |
| | composed Shengpang | 627 |
| Non-proper Shengpang | | 176 |
| Total | | 1348 |

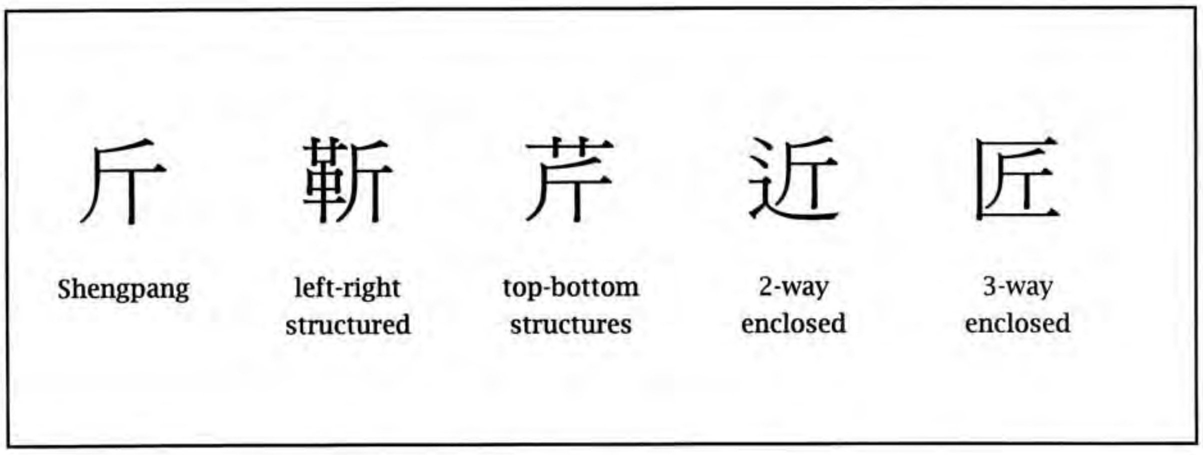
Table 2.2: Classification of Shengpang

Although there are only 1348 Shengpangs, we are interested in how many shapes are actually needed to compose the whole character set. A detailed analysis has been done on [16] to study this.

For every Shengpang entry in [16], we count the number of characters it comprises. Each character is categorized into left-right structured, top-bottom structured, 2-way enclosed and 3 or 4-way enclosed (figure 2.4). If the Shengpang is not separable from the character or its shape is altered so that it cannot share with other characters, it is classified as miscellaneous.

The total number of shapes can be estimated by summing the number of different structures of each Shengpang and the total number miscellaneous structured characters.

Figure 2.4



Classifying the structures of Shengpang

The number of shapes sums to 2230, which are responsible for the composition of 6416 characters. On average, every shape is composes only 3.14 characters. However, the distribution is very uneven. Table 2.3 shows the distribution of number of times a Shengpang is reused, with the figures further broken down by the kind of structures in which the Shengpangs are used. The last 101 shengpangs which are reusable for more than nine times are most impressive which composing 1273 characters altogether. Alas, there are also 1036 shapes which compose one character only.

| Usage Frequency | Left-right structured | Top-bottom structured | 2-Way enclosed | 3,4-Way enclosed | All structures | characters composed |
|---|---|---|---|---|---|---|
| 1 | 315 | 424 | 232 | 65 | 1036 | 1036 |
| 2 | 189 | 144 | 65 | 8 | 406 | 812 |
| 3 | 164 | 62 | 10 | 0 | 236 | 708 |
| 4 | 98 | 33 | 2 | 1 | 134 | 536 |
| 5 | 90 | 18 | 2 | 0 | 110 | 550 |
| 6 | 59 | 12 | 1 | 0 | 72 | 432 |
| 7 | 47 | 4 | 0 | 0 | 51 | 357 |
| 8 | 42 | 2 | 0 | 0 | 44 | 352 |
| 9 | 37 | 3 | 0 | 0 | 40 | 360 |
| >9 | 96 | 5 | 0 | 0 | 101 | 1273 |
| Total | 1137 | 707 | 312 | 74 | 2230 | 6416 |

Table 2.3: Shengpang usage distribution

Two kinds of characters are not counted in the table 2.3. There are 115 characters classified as miscellaneous structured and there are also 545 basic shengpangs. The overall result will be 3045 shapes composing 7272 characters. The reusable rate is 2.39.

Again the degree of reusability of Shengpang is determined by the number of variant shapes and the odd characters that cannot be composed. In this case the large number of odd characters and shapes that compose only one character contribute to the moderate reusability.

Note that the analysis is made on 7,000 simplified characters. For the 13,000 traditional characters in the BIG-5 character set the rate is expected to be higher since the remaining 6,000 characters are almost more complicated characters which can be composed by Bushous and Shengpangs. For example, Shengpang "泰" composes only one character "傣" in [16], but it composes two characters "傣" and "溙" in the BIG-5 character set. Also, Shengpang "昜" composes only two characters "觴" and "殤" in [16], but it composes six characters "揚", "湯", "觴", "殤", "鬺" and "傷" in BIG-5 character set.

## 2.7. Summary

Most Chinese characters are composed characters (*xiangxing* or *huiyi*). There are four classes of basic structures of Chinese characters, that is left-right structure, top-bottom structure, inside-outside structure and singleton structure. Left-right structure is the most common structure used (69%).

We find that the traditional decomposition of characters into Bushous and Shengpangs is very useful for finding the set of radicals. The set of Bushous is regular and estimated to have a very high reusable rate. The set of Shengpangs is more varying and estimated to a moderate reusable rate. This is not surprising since there are lot more Shengpangs (1348 in 7272 characters) than the number of Bushous (around 200).

However, the Bushou-Shengpang decomposition is not the only way to decompose characters. Bushou and Shengpang may decompose into lower level components themselves. Even those characters classified as miscellaneous structured and solitary characters which cannot share their top level components may find it possible to share their lower level components.

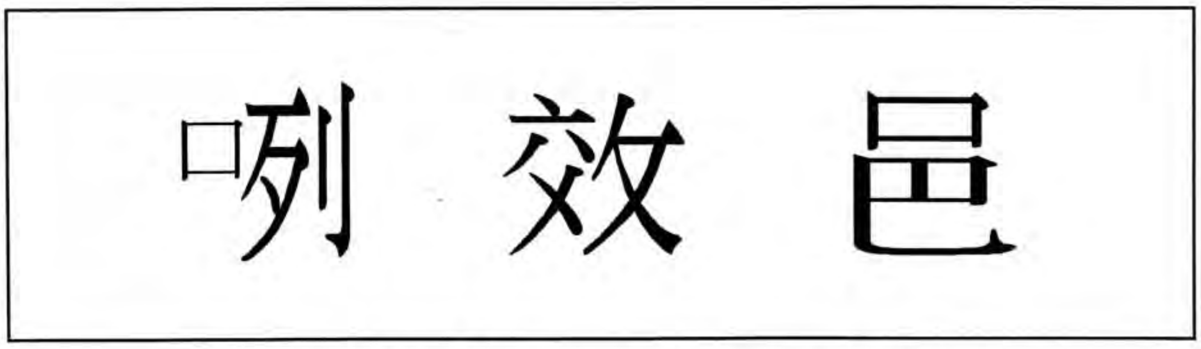
# Chapter 3
# Composition by Radicals

## 3.1. Introduction

Chinese characters are composed by radicals. Although the number of Chinese characters is very large, each is composed by only a small number of basic components. Naturally we want to draw the shape of these basic radicals and then compose new characters by using these radicals , transforming them to the right sizes and positions. Then the task of building a Chinese font would not be so tedious and the storage problem will be greatly alleviated. In fact, some early Chinese systems actually apply this approach [14]. However, they result in some terrible looking characters.

The straight forward approach fails for several reasons. First, linear transformation is used to scale the radicals. Linear transformation distorts the shape of the radical seriously. Since the radical masters were drawn at specific sizes, when they are scaled, the stroke widths changed undesirablly. Moreover, excessively scaling could seriously distort the shape of initially good-looking radicals (Figure 3.1).

Figure 3.1



Outlines distorted in excessive linear transformation

Secondly, radicals in different characters may be different in shape. The radical "土" in characters "堯", "堡" and "境" appear in differently. Very often the shape of a radical is slightly changed to adapt to its neighbors in a character so that strokes are evenly spaced and the character looks balanced.

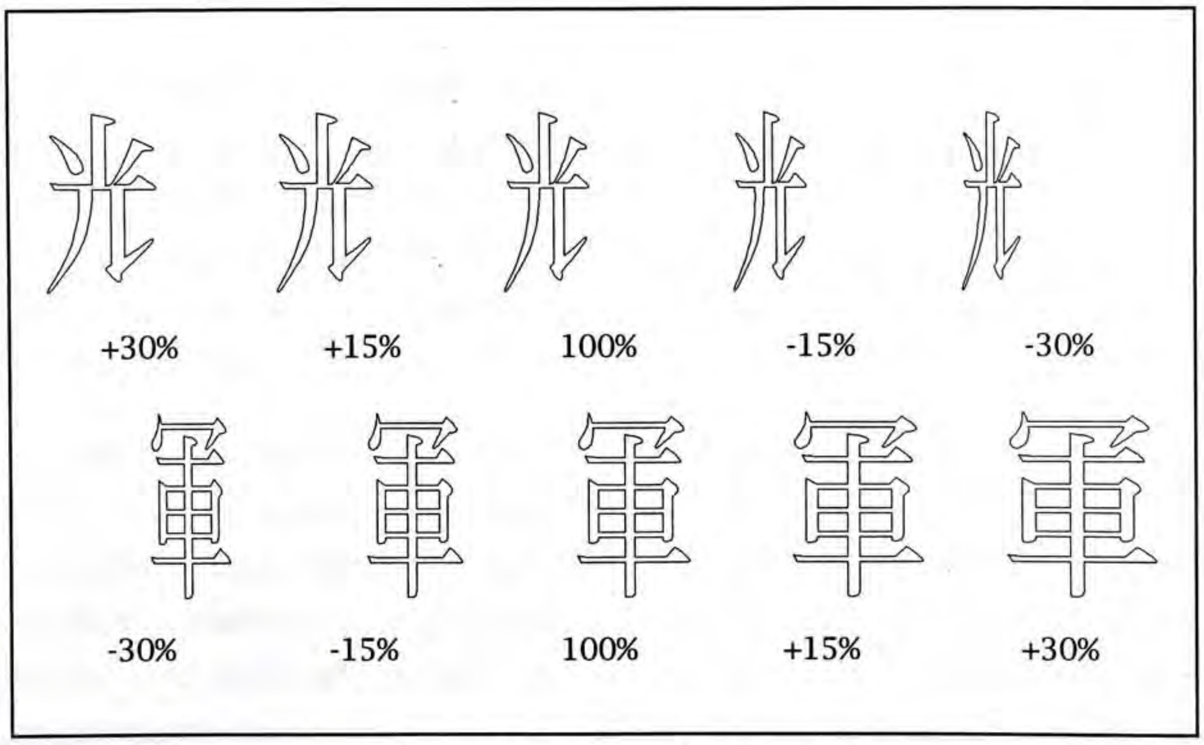
For example, the slant stroke of left radical "木" in character "林" has changed into a dot stroke so that the character does not look too congested in the middle.

## 3.2. Transforming radicals

Although linear transformation has its limitations, we still use it to scale radicals. To avoid serious distortions as in straight forward linear transformation, several versions of radicals are built when necessary so that excessive scaling can be avoided. Radicals are shared only if they satisfy two criteria:

- Their shapes are similar. Very often two radicals are structurally the same but are slightly different in shape (e.g. strokes which are horizontal in one radical become slant in another radical, relative stroke lengths differ, etc.) In these cases, two or more shapes are needed to represent them.
- Their sizes should be similar. Radicals scaled by a large amount are subjected to distortion. Scaling range within ±15% is recommended (Figure 3.2). If the radicals have to be scaled by larger amount, we build two shapes in different proportions.

Figure 3.2



Scaling radicals from -30% to +30%

Since we decompose characters at high granularity (Bushou and Shengpang), the radicals' shapes are rather stable (i.e. similar in size and shape). The number of different versions of each radical should not be too high. Chapter 2 has shown that 2230 shapes are needed to represent 1348 Shengpangs. That is, on average, each Shengpang requires only 1.65 shapes.

The advantages of linear transform radicals are:

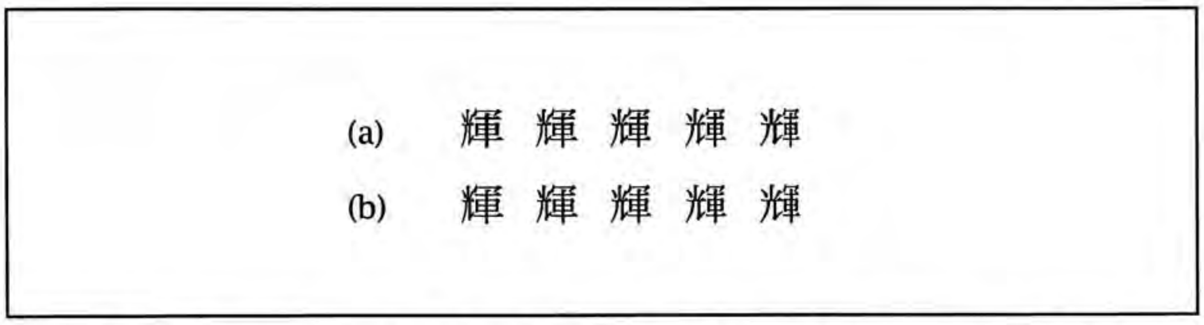
- The transformation is very fast. No complex formula or algorithm is needed to scale the control points except simple multiplications and additions. The impact on rasterization time is very small.
- Very few parameters are needed to encode the usage of radicals. For each radical usage, two points are needed to encode the linear transformation and one word to encode the index of the radical used.
- Type designers can edit the character by direct manipulation. They can adjust the radicals by simply moving their bounding rectangles.

## 3.3. Quality of transformed radicals

Linear transformation will always distort the shape of the radical. Figure 3.2 shows the outline of a character by scaling its two components in different proportions. At the two extreme ends where the radicals are scaled by +30% and -30%, significant distortions occur. However, for the three pair of samples at the middle, their transformation is restricted to ±15% and the result is quite acceptable.

Of course, the quality of the character image depends on the outline. However it is also affected by rasterization. The influence of rasterization is particularly important when the outline is rendered in low or medium resolution. Under such circumstance, the grid of the bitmap is not fine enough to represent the shape precisely. On the other hand, distortion obtained from transforming the radicals is often negligible when the grid is too coarse to expose it. Consider a 60x60 character bitmap which has 1 pixel wide horizontal strokes and 3 pixels wide vertical strokes. The outline can be scaled up by 1/3 before the some vertical stroke become 4 pixels wide to spoil the character image.

Figure 3.3



(a) Unhinted images (60x60) of radicals scaled from -30% to +30%. (b) Hinted images (60x60) of radicals scaled from -30% to +30%

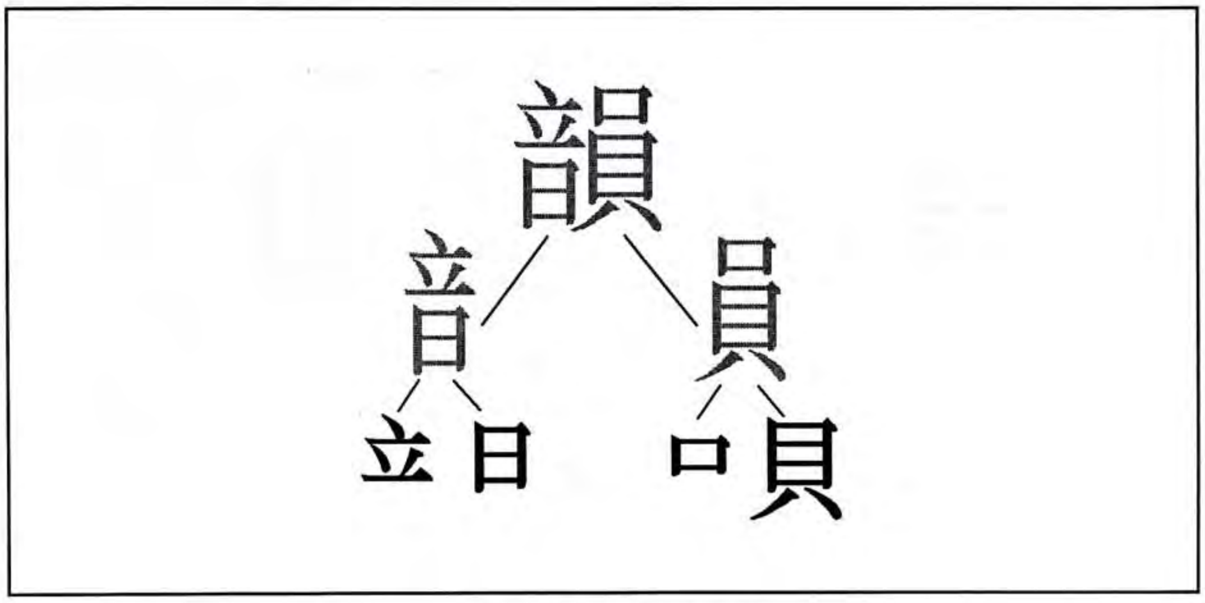
At low and medium resolution, hinting (chapter 4) plays a dominant role in controlling the quality. Hinting intentionally distorts an outline so that important features of a character are visible and uniform at all sizes. Our hinting algorithm ensures that horizontal stroke and vertical stroke width of the image are uniform in all resolutions. More importantly, it can fix the distortions result from transforming the radical.

In Figure 3.3, characters are composed by radicals scaled in Figure 3.2. Pay special attention to the contrast between component "光" and component "軍". In row (a), the first and last characters are composed by radicals scaled by ±30%. The components contrast badly. The second and fourth character are composed by radicals scaled by ±15%. Their contrast is much lower. Character images in row (b) are hinted, which produce much more consistent images in all samples.

## 3.4. Lower level components

The shape of a character is a combination of its own outline and its components'. Since the representation of radicals is treated the same as ordinary characters, a radical can be composed by other radicals. Therefore a character can be represented as a tree of components. For example, "韻" is composed by "音" and "員". Similarly, radical "音" is composed by "立" and "日", and radical "員" is composed by "口" and "貝" (Figure 3.4).

Figure 3.4



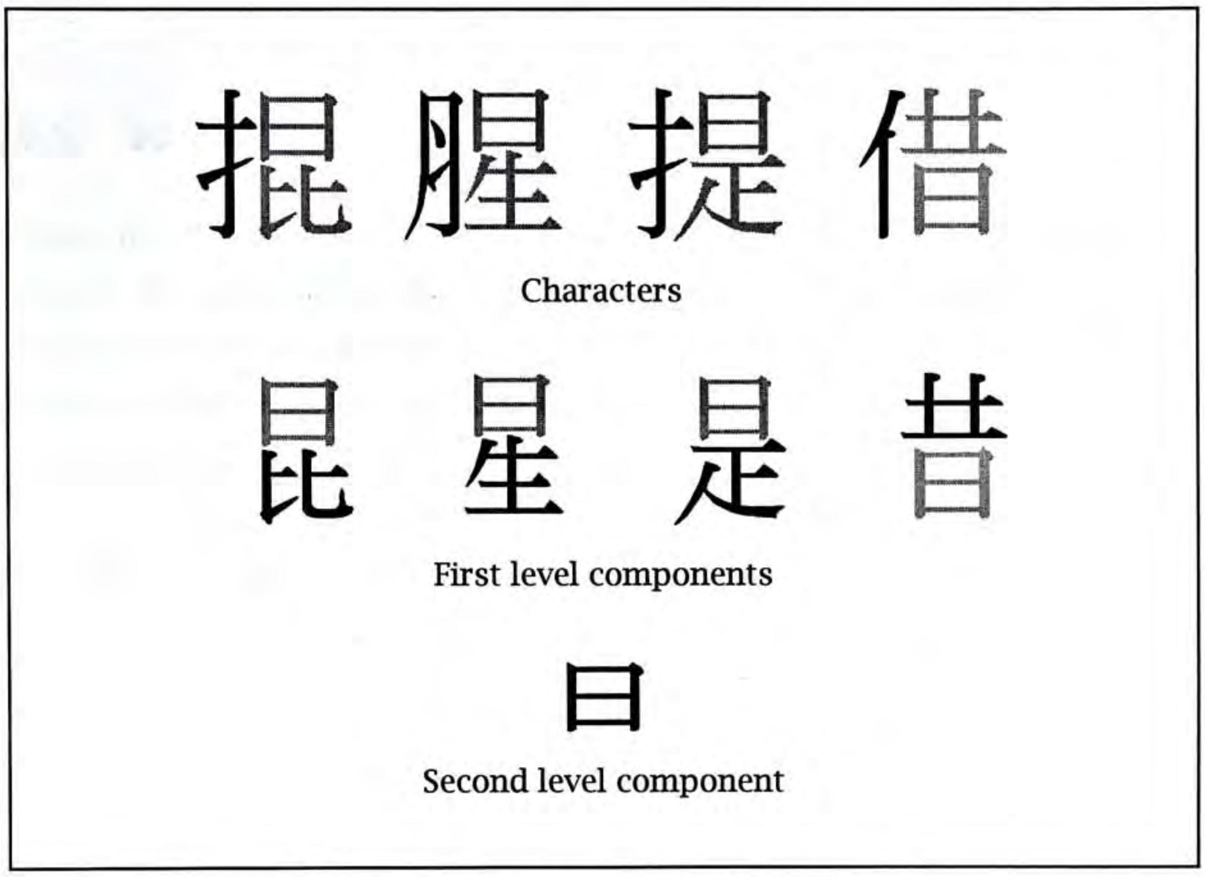
Tree of components

Lower level radicals are useful for reducing font size in two circumstances:

a. They are used to compose higher level components. Chapter 2 estimates that 2230 shapes are needed to construct the Shengpang part of 6,416 characters. Second or lower level components could be used to compose these 2230 shapes (Figure 3.5).

   Since most Shengpangs are also proper characters, they could also be decomposed into Bushou and Shengpang. As Bushous have a higher reusable rate than Shengpang, it would be easier to find common Bushous than common Shengpangs among the first level components.

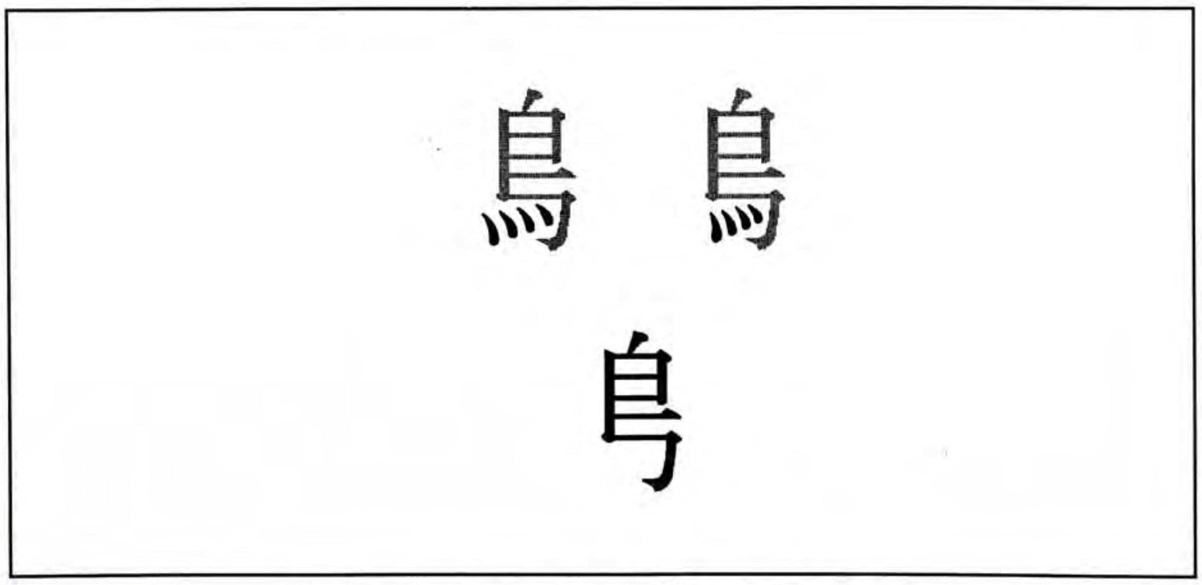
Figure 3.5



Common Bushou of first level components

b. Lower level components could be used to build several variant shapes which are only slightly different. For example, the two "鳥" shapes in Figure 3.6 are only different at the four dots in the bottom. To get the biggest advantages of their commonness, the two variant shapes may be composed with their common part " ⿱自丂".

Figure 3.6



Reuse common part of two variant shapes

## 3.5. Summary

Linear transforming radicals in a straight forward manner results in distorted shapes. To avoid undesirable distortion, several versions of radicals are built when necessary. Radical shapes are shared only if their sizes and shapes are similar, so that excessive scaling can be avoided. The qualities of transformed radicals are maintained by two means. First, by limiting the scaling factor to small amount (±15%), and second, by hinting which compensates the distortion during rasterization.

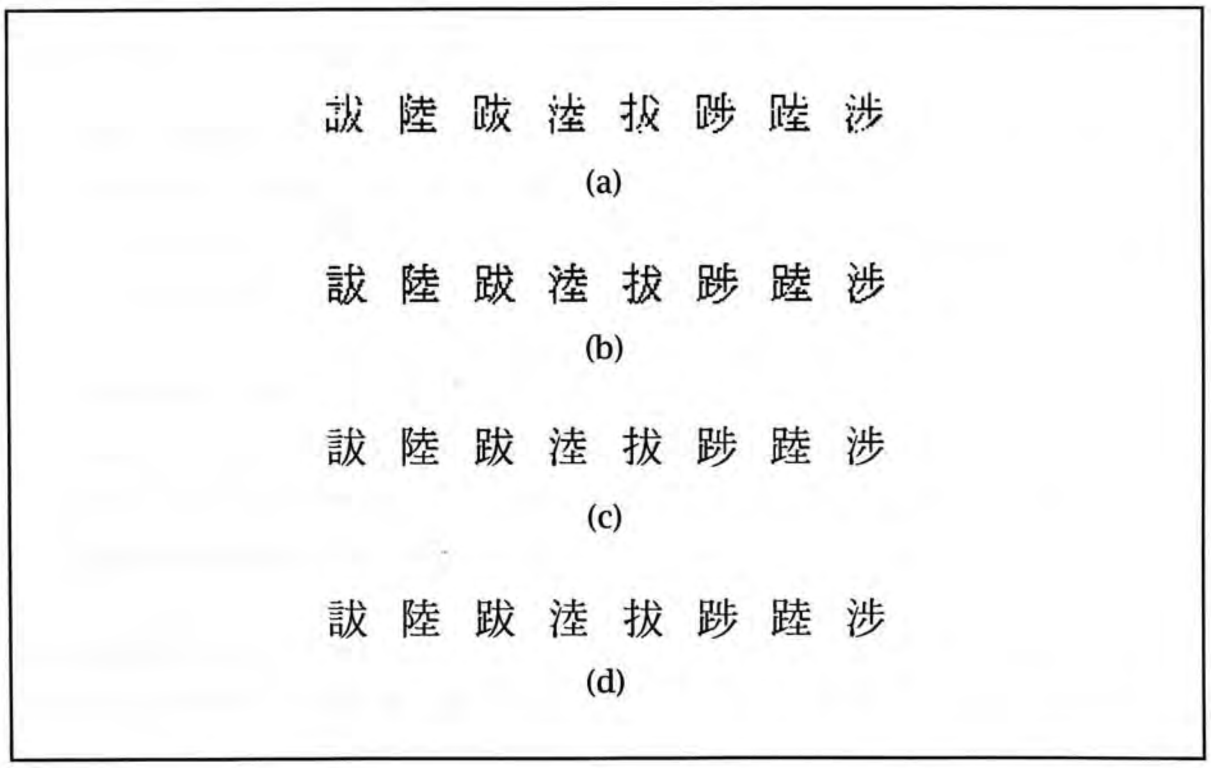
Since radicals themselves can be composed by other radicals, a character can be represented as tree of components. Lower level components can further reduce the storage of composed font.

# Chapter 4
# Automatic Hinting for Chinese Font

## 4.1. Introduction

Outline font is the most common form of font representation because of its many advantages (chapter 1). However, most output devices (CRT, printer, etc.) do not output the outlines directly. Instead, these outlines must be converted into bitmap format before sending to output devices. This conversion process, called rasterization, works fine at high resolution (600 dpi or above). It produces good-looking bitmap that closely render the shape of the outline because the grid is fine enough that difference of one pixel is indistinguishable.

Figure 4.1



(a) Unhinted 20x20 bitmap. (b) Hinted 20x20 bitmap. (c) Unhinted 60x60 bitmap. (d) Hinted 60x60 bitmap.

But at lower resolution, straight forward rasterization cannot produce good-looking bitmaps. The pixel grid is so coarse that misplace one pixel can significantly damage the image's readability. This problem is especially serious for Chinese characters because of their high stroke density. At screen resolution (96 dpi), the characters produced are seriously distorted without correction.

To improve the quality of low and medium resolution bitmaps, we must supply extra information on important features of the character shape that must be rendered. For Chinese characters, these include the stroke widths and the shape and size of the serifs. Such extra information is called *hints.*

[6] used grid constraints to regulate the appearance of characters. [9] and [8] described pattern recognition method for automatic font hinting. However, these methods are not directly applicable to Chinese fonts because they are designed for sets of characters only. To design a hinting algorithm for Chinese font, we must consider the very large size of Chinese character sets. Because of this characteristic, the hinting algorithm should have the following desirable features:

- Hints should be generated automatically with minimal human intervention. Manual crafting of hinting instruction is too tedious to be done for ten thousand Chinese characters.
- Most work should be done at preprocessing time. Recognizing font features during raster time will be too inefficient. Minimizing rasterization time is an important goal for Chinese font system that is not to be sacrificed.
- Extra information for each character should be minimized. Since minimizing the storage of Chinese font library is the goal of this thesis, it does not make sense if encoding the hinting information would have to increase the storage significantly.

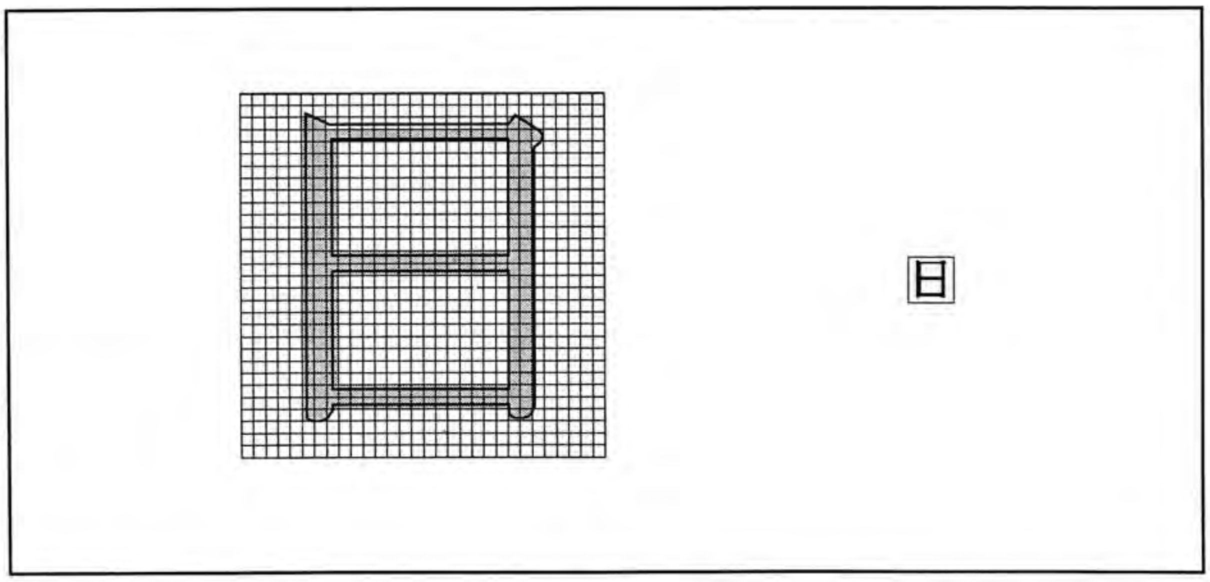
In the following sections, a hinting algorithm for Chinese font is presented. This algorithm works by regularizing the stroke width of the bitmaps it produces. It meets all three criteria above. It automatically recognizes the features at preprocessing time. The impact to rasterization efficiency is, therefore slight. Besides, the extra information to be encoded is insignificant, comparing to the size of the font library.

## 4.2. Automatic hinting for Chinese font

One of the most important feature of Chinese characters is the horizontal and vertical stroke width. Similar strokes within a character should have the same width to give the impression of uniformity. For "*song*" style, horizontal strokes are thinner than vertical strokes. For "*hai*" style, horizontal and vertical strokes are about the same width. The strokes should be drawn a bit wider in simple characters (characters with very few strokes) and should be drawn a bit thinner in complex characters so that different characters would appear in about the same blackness. Nevertheless, horizontal stroke width and vertical stroke width in the same character should be uniform.

When a character is rasterized in low resolution, strokes of equal width may not produce same number of pixels in width (Figure 4.2), depending on where the outline lay relative to the grid. If some strokes render as 2 pixel wide and some as 3 pixel wide, the 50% difference in stroke width can be very irritating to a reader.

Figure 4.2



Rasterizing unhinted outline results in stroke widths of uneven number of pixels.

Our hinting algorithm ensures that horizontal strokes and vertical strokes in a character appear in uniform width. It involves three steps:

1. Automatically recognize all horizontal and vertical strokes in a character.
2. Regulate the outlines of the strokes to uniform width, usually the average width of the strokes identified.
3. When the character is being rasterized to a specific pixel grid, calculate the number of pixels needed to draw the strokes. Grid-fit the outline so that each stroke is rasterized to produce the specific number of pixels in width.

Step 1 and step 2 are preprocessing to the character outline prior to rasterization. Step 3 is done during rasterization (Figure 4.3).

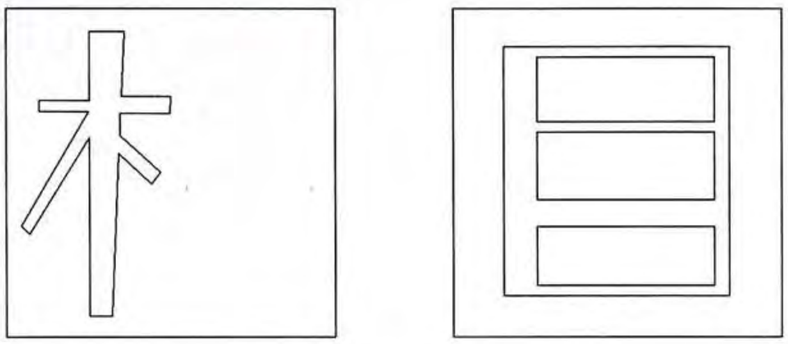
In addition, there is a step 4 to grid-fit the outline of radicals which compose the character, so that the horizontal and vertical strokes of each radical appear in uniform width after their scaling process.

4. Grid-fit the outline of each radical component to the same stroke width as the root component. Take into account that the strokes of the components have been scaled and would require extra compensation.

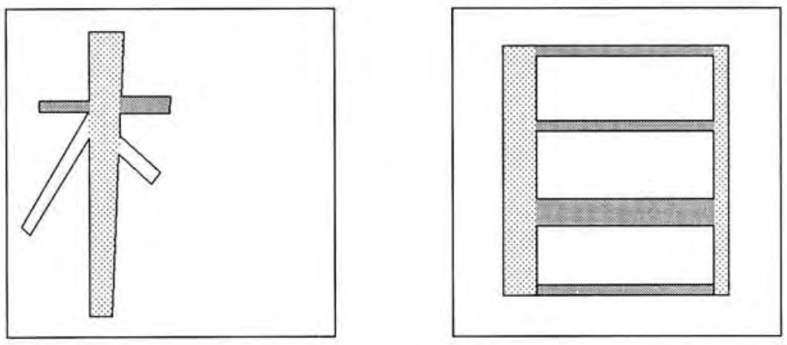
Step 4 not only regulates the appearance of a character, it also eliminates the distortion induced while scaling the radicals.

After the horizontal and vertical strokes are hinted, the overall appearance of the bitmap can improve greatly. Diagonal strokes do not affect the overall impression of a character as much because they are subjected to less quantizing error as the horizontal and vertical strokes. They may appear as staircases but their average number of pixels is closer to their widths.

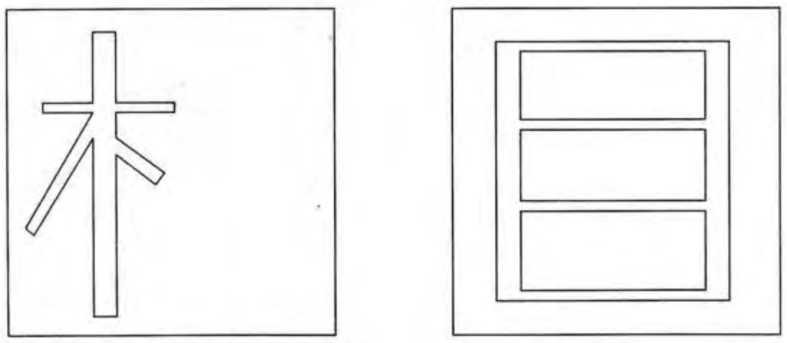
Figure 4.3



(a) Imperfect outline is either handmade or acquired by tracing a scanned image.



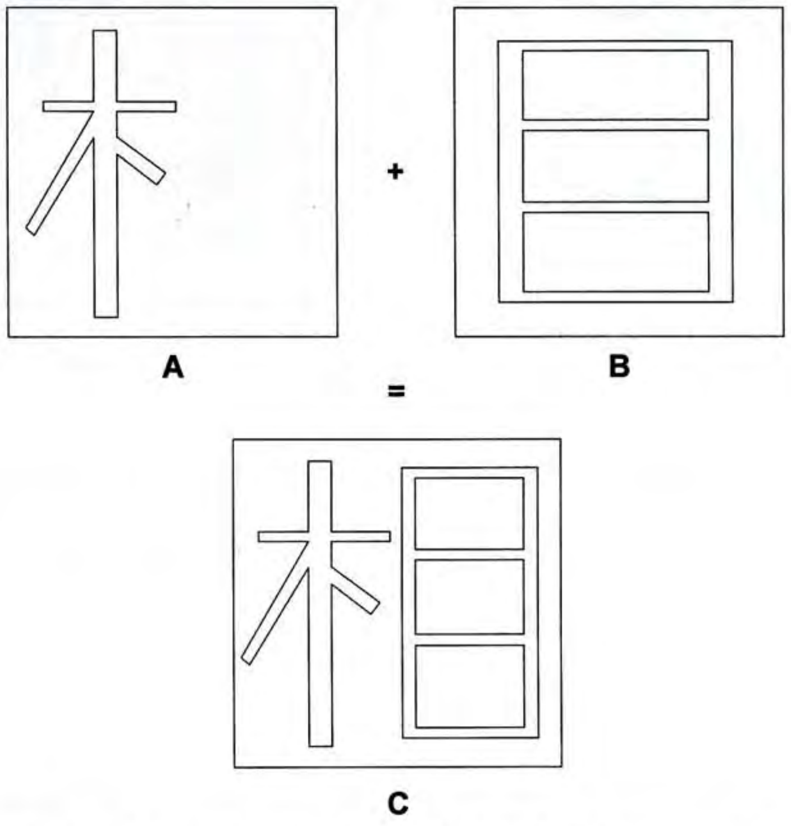
(b) Step 1: horizontal and vertical strokes detected.



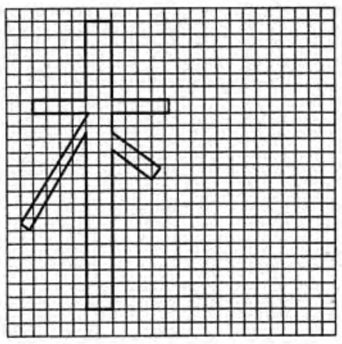
(c) Step 2: stroke width regularized. Uniform stroke width is encoded with the outline for later use in grid-fitting.

Hinting Chinese font: preprocessing of character outline.

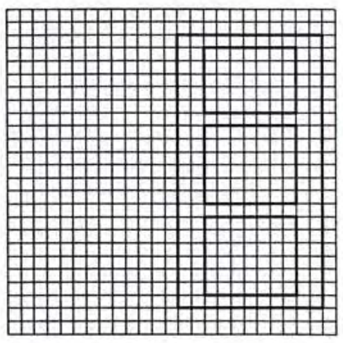
Figure 4.4



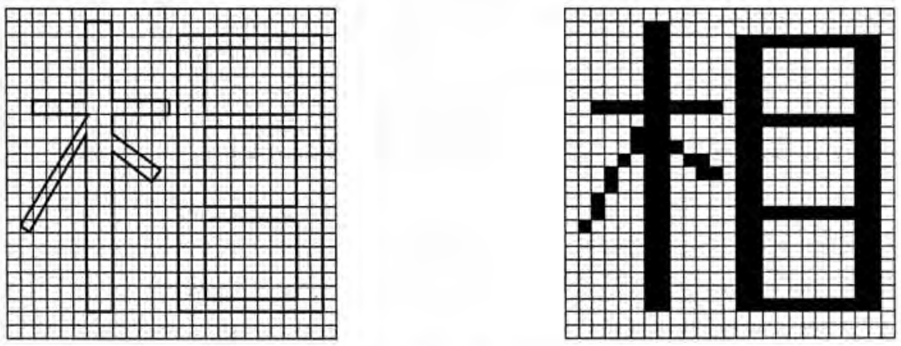
(a) Root character A combined with radical B to form a composed character C. Component B is compressed 40% horizontally.



(b) In a 25x25 grid, 1 pixel is allocated to horizontal strokes and 2 pixels allocated to vertical strokes. Apply step 3 to grid-fit the outline.



(c) Apply step 4 to grid-fit component A. Also allocate 1 pixel to the width of horizontal strokes and 2 pixels to the width of vertical strokes.



(d) The grid-fitted outline renders the character uniformly. The distortion introduced by the transformation of component B is compensated for in grid-fitting.

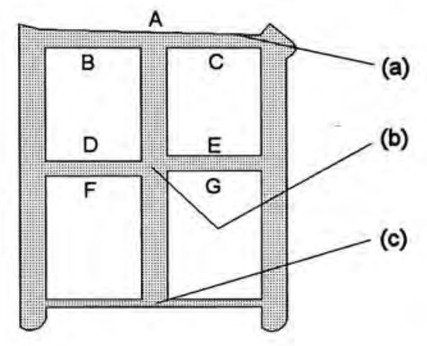
Hinting Chinese font: grid-fitting during rasterization time

## 4.3. Stroke recognition

Since strokes can overlap each other in a character, structural information is not encoded in a character's outline. A stroke may correspond to an odd collection of horizontal lines. For example, the topmost stroke of Figure 4.5 corresponds to edge A, B and C, while the middle stroke corresponds to edges D, E, F and G, which actually belong to four different contours. Although these edges appear independently of each other, they join together to form the ultimate horizontal. For example, edges D and E form a single horizontal line to become the upper part of the horizontal stroke when they are joined. Unfortunately, the outline of a character is often not perfect so that two edges which are supposed to connect to each other do not actually do so. Edges D and E demonstrate such a case clearly as they do not line up on the same horizontal level. Such errors must be taken into account during stroke recognition. Stroke recognition is a simple pattern recognition and its algorithm is presented here. It should be noted that the algorithm considers horizontal strokes only because the recognition of vertical strokes is clearly symmetric by rotating the character by 90 degrees.

The recognition algorithm for horizontal strokes contains three steps. Horizontal lines are identified in the first step. Then these horizontal lines are paired up to form stroke segments in the second step (e.g., D, F in Figure 4.5). Finally, in the third level, stroke segments are joined together to form a stroke.

Figure 4.5



Imperfections in outline: (a) slanted horizontal line, (b) misaligned stroke segments and (c) non-uniform stroke width.

### 4.3.1. Identify horizontal lines

Horizontal lines are identified in this step which are examined in subsequent steps to check if they form horizontal strokes.

### 4.3.2. Identify stroke segments

An isolated stroke has single stroke segment only. An intersected stroke, however, is formed by several stroke segments. Two horizontal lines are paired up as a stroke segment if they satisfy the following criteria:

a. Their vertical distance is smaller than the maximum stroke width allowed.

b. They overlap horizontally.

c. The upper line goes left and the lower line goes right.

d. The area between two lines is solid black; i.e., there should be no lines intersecting or enclosed in the quadrilateral area form between two lines.

Criteria c) follows from the way that the orientation of the contours. The outermost contours is assigned in anti-clockwise direction and inner contours are assigned clockwise and anti-clockwise directions alternatively as shown in Figure 4.7.

Lets us call parts of a stroke segments. Note that one single line may pair up with more than one line to form several segments. In Figure 4.5, edge A pairs up with edge B and C to form segments AB and AC.

### 4.3.3. Stroke recognition

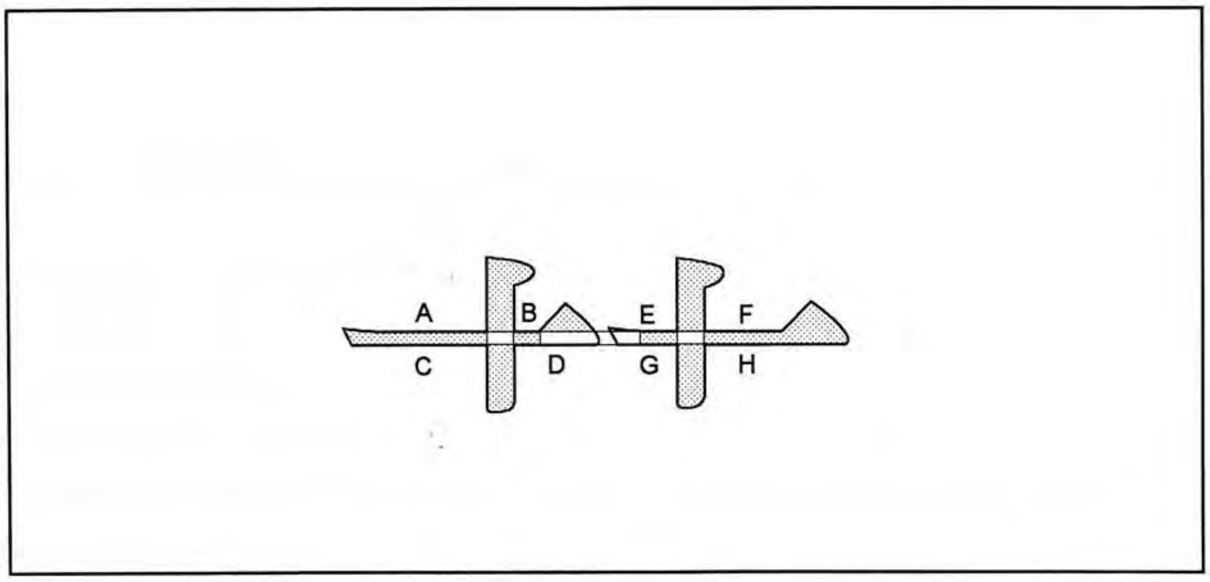
Two stroke segments are joined to form a single stroke if they satisfy the following criteria:

a. Their vertical distance is close to zero, within a margin of error.

b. The area between the stroke segments is solid black. That means there should be not lines intersecting or enclosed in the quadrilateral area form between the two segments.

For example, segment AC and segment BD in Figure 4.6 are identified as part of a single stroke because the area between them is solid. However, segment BD and segment EG do not belong to a single stroke because there are lines crossing the quadrilateral area between them.

Figure 4.6



Connect stroke segments.

## 4.4. Regularize stroke width

The purpose of recognizing strokes in an outline is to regularize their widths. Several imperfections in the outline are corrected in this step (Figure 4.5):

1. **Slightly slanted horizontal lines** are fixed first. They are leveled to their averaged height.

2. **Misaligned stroke segments** are adjusted. If two stroke segments are not exactly aligned, they may appear one pixel different in height in rasterization.

3. **Stroke width** is regularized. Stroke width is adjusted to a uniform distance with respect to each stroke's centerline.

The adjusted stroke width is encoded with the outline. Its value is used later in grid-fitting when the outline is adjusted again so that a uniform number of pixels is allocated for each stroke of the character. As we will see in the following sections, regularizing the stroke width of the outline makes grid-fitting much easier.

## 4.5. Grid-fitting horizontal and vertical strokes

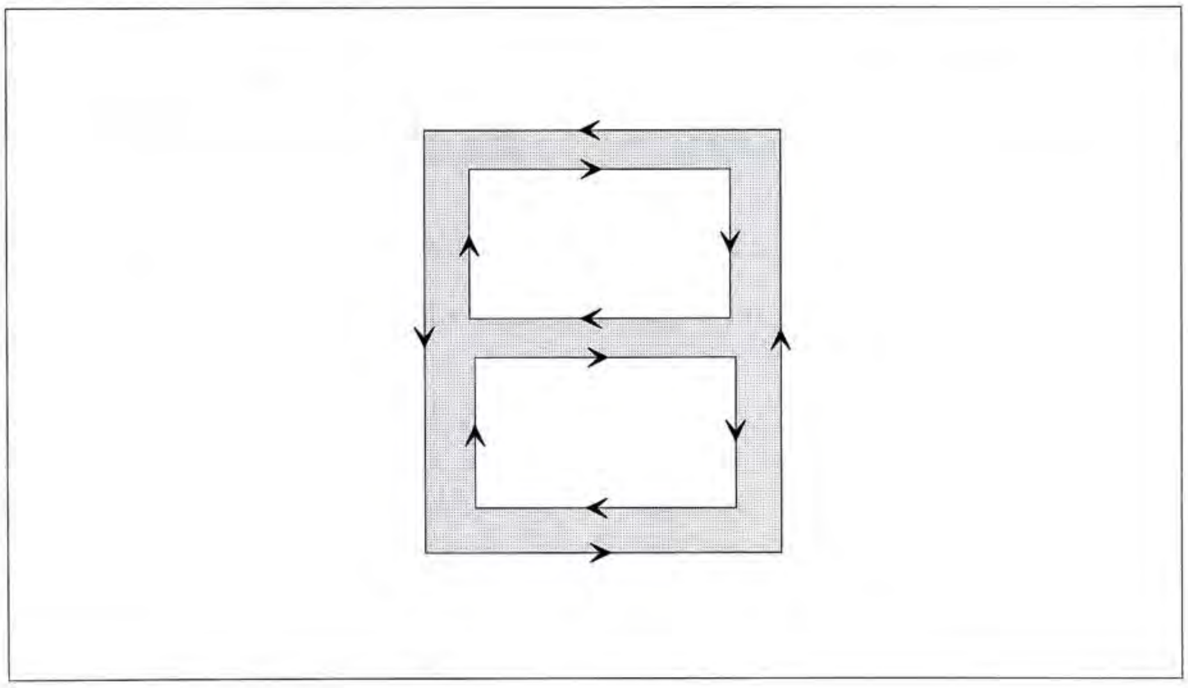
The goal of font hinting is to identify the critical features of a character and ensure that these features are preserved when rendering the character at different sizes. Since it is the outline of the character that determines which pixel should be turned on and which should not, it is sometimes necessary to distort the shape of the outline in order to produce a high quality image. This distortion of the outline is known as grid-fitting.

The algorithm for grid-fitting horizontal and vertical strokes is presented here. Only grid-fitting of vertical strokes is illustrated as the procedure for grid-fitting horizontal strokes is symmetric.

The first step in grid-fitting is to find the part of the contour that corresponds to a horizontal or vertical stroke. This step is simple after stroke regularization because the stroke lines will be straight lines at exactly 0 or 90 degrees. To avoid misinterpreting other exactly horizontal or vertical lines which do not form strokes, we may shift one end of these lines by 1 em-square unit so that they will not mix with perfect horizontal or vertical lines. Line shifts of 1 unit are hardly recognizable. Alternatively, we may use a quadratic Bezier curve to represent the straight line at the expense of an extra off-curve control point.

The second step is to identify the two sides of a stroke. By previous convention, the left edge of a vertical stroke goes down and its right edge goes up; the top edge of a horizontal stroke goes left and the bottom edge goes right (Figure 4.7).

Figure 4.7

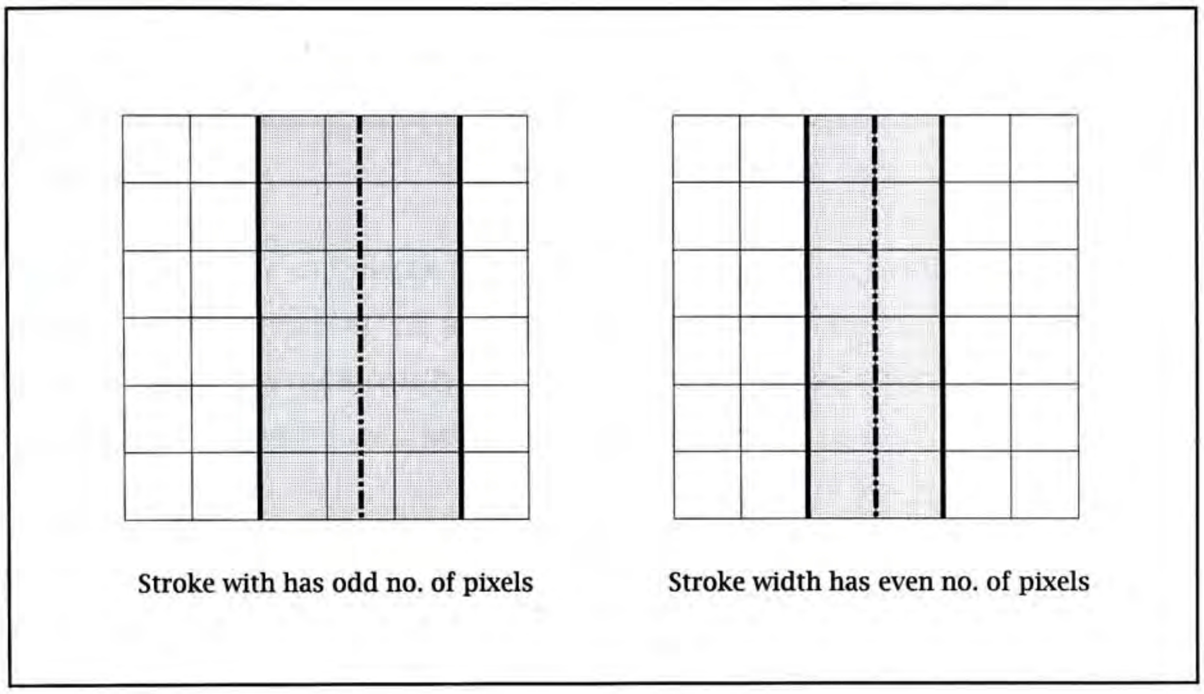


Orientation of Contours

The third step is actually shifting the lines. Very little information is needed to perform this step. It can be done without any information on the opposite side of the stroke. This means the contours can be grid-fitted in one pass and the efficiency is very high.

The stroke width, encoded with the outline (section 4.4), should be rounded into integer number of pixels first. The contour of the strokes should be repositioned so that exactly that number of pixels will be turned on. First we must decide where the location of the centerline of the stroke should be. If the rounded stroke width is even, the centerline should lie on the grid. If the number of pixel is odd, the centerline should lie on the center of the grid as shown in Figure 4.8.

Figure 4.8


Stroke with has odd no. of pixels
Stroke width has even no. of pixels

Aligning the centerline of a vertical stroke

Since the centerline of the stroke is not encoded with the character, we have to compute it. The centerline is defined as the middle of two sides of the stroke, i.e.

$$\text{centerline} = \text{left edge} + \frac{1}{2}\text{ stroke width}$$

or

$$\text{centerline} = \text{right edge} - \frac{1}{2}\text{ stroke width}$$

Then we may shift the centerline, if the stroke width is an even number of pixels

$$\text{centerline} = \text{round}(\text{ centerline })$$

If the stroke width is an odd number of pixels

$$\text{centerline} = \text{round}(\text{ centerline} - \frac{1}{2}) + \frac{1}{2}$$

After the location of the centerline has been adjusted, we may actually move the left or right edge of the contour,

$$\text{left edge} = \text{centerline} - \frac{1}{2}\ \text{rounded stroke width}$$

or

$$\text{right edge} = \text{centerline} + \frac{1}{2}\ \text{rounded stroke width}$$

**Example**

After scaling the outline to the grid, a vertical stroke's width is 2.6, with its left edge positioned at 1.8 and its right edge positioned at 4.4 (Figure 4.9). The fractional stroke width 2.6 is rounded to 3 pixels wide. Without grid-fitting, only pixels in column 2 and column 3 will be turned on.

Now we will grid-fit the left edge first,

$$\begin{aligned}\text{centerline} &= 1.8 + \frac{1}{2} * 2.6 \\ &= 3.1\end{aligned}$$

Since the number of pixels of the stroke width is odd, we round the centerline to the center of the grid,

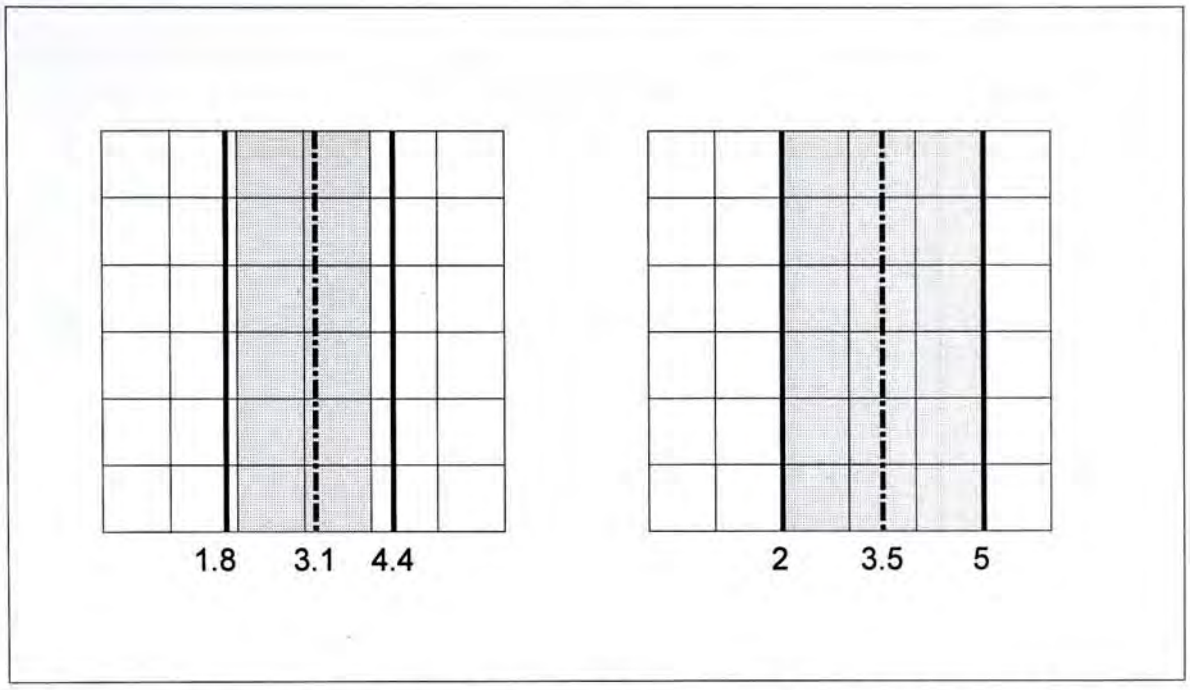
$$\begin{aligned}\text{centerline} &= \text{round}(\ 3.1 - \frac{1}{2}\ ) + \frac{1}{2} \\ &= 3.5\end{aligned}$$

Finally the left edge is shifted to,

$$\begin{aligned}\text{left edge} &= 3.5 - \frac{1}{2} * 3 \\ &= 2\end{aligned}$$

Similarly, the right edge is shifted to 5. This stroke will be correctly rasterized to 3 pixels.

Figure 4.9



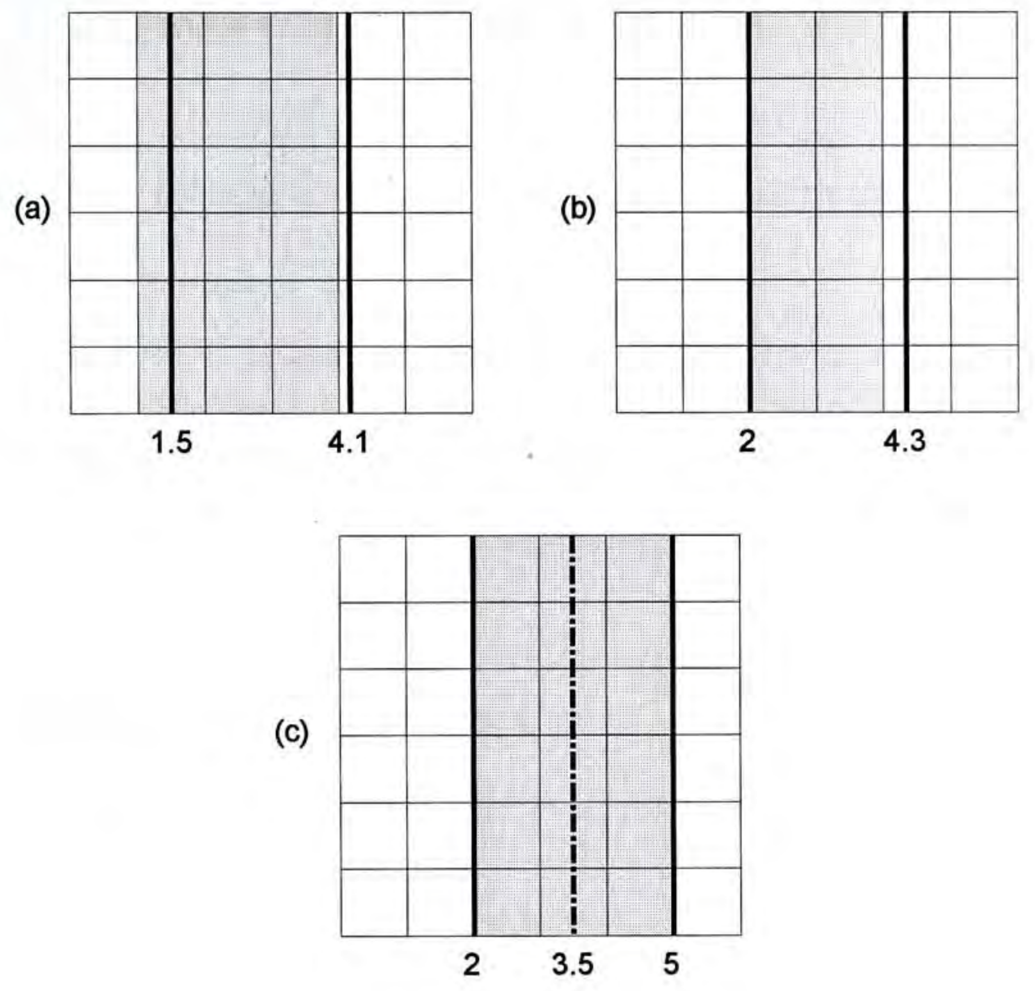
(a) A vertical stroke before grid-fitting (b) A vertical stroke after grid-fitting

## 4.4. Grid-fitting radicals

It has mentioned in chapter 3 that hinting can be combined with composition by radical so that the distortion made in transforming the radical can be significantly eliminated in grid-fitting. This simple procedure of grid-fitting radicals is presented in this section.

First the stroke width rounded into pixels is calculated for the top level component of the character. This stroke width is a character attribute that lower level radicals should conform to. For each radical, its stroke width is distorted in scaling. The procedure of grid-fitting the radical remains the same except that the actual stroke width depends on the radical. The actual stroke width is stroke width of the radical template multiplied by the scaling factor in the composed character. It is used in calculating the centerline.

Figure 4.10



(a) Original template stroke width is 2.6 pixels. (b) After scaling, stroke width is 2.3 pixels. (c) After grid-fitting, stroke width is 3 pixels.

**Example**

A radical of stroke width 2.6 is scaled down to become 90% (Figure 4.10). The stroke width become 2.3 in the composed character. Its left edge is at 2 and its right edge at 4.3. The rounded stroke width 3 is inherited from the top level component. The radical will be grid-fitted so that its stroke is rasterized to 3 pixels. The left edge is grid-fitted as an example.

First the centerline is calculated by using the actual stroke width of the scaled radical

$$\begin{aligned}\texttt{centerline} &= 2 + \frac{1}{2} * 2.3 \\ &= 3.15\end{aligned}$$

This is rounded to the center of the grid

$$\text{centerline} = \text{round}\left( 3.15 - \frac{1}{2} \right) + \frac{1}{2}$$

$$= 3.5$$

Finally the left edge is shifted by using 3 pixels as the rounded stroke width

$$\text{left edge} = 3.5 - \frac{1}{2} * 3$$

$$= 2$$

and similarly, the right edge is shifted to

$$\text{right edge} = 3.5 + \frac{1}{2} * 3$$

$$= 5$$

The final result is shown in Figure 4.10c.

## 4.7. Summary

Hinting is a method to improve the image quality of rasterization. It ensures that important features of a character are rendered in all sizes.

An automatic hinting algorithm for Chinese font is presented in this chapter. This algorithm makes sure that the horizontal strokes and the vertical strokes are rasterized to uniform widths. The advantages of this algorithm are:

- Strokes are detected and regularized automatically. Most work is done prior to rasterization. The information found, called hinting, is encoded with each character individually.
- The size of additional hinting information needed to be encoded with the font library is very small.
- The grid-fitting process is very efficient. It will hardly slow down the rasterization rate.

- The grid-fitting process compensates the distortion resultant from scaling the radical components so that stroke widths remain consistent in the composed characters.

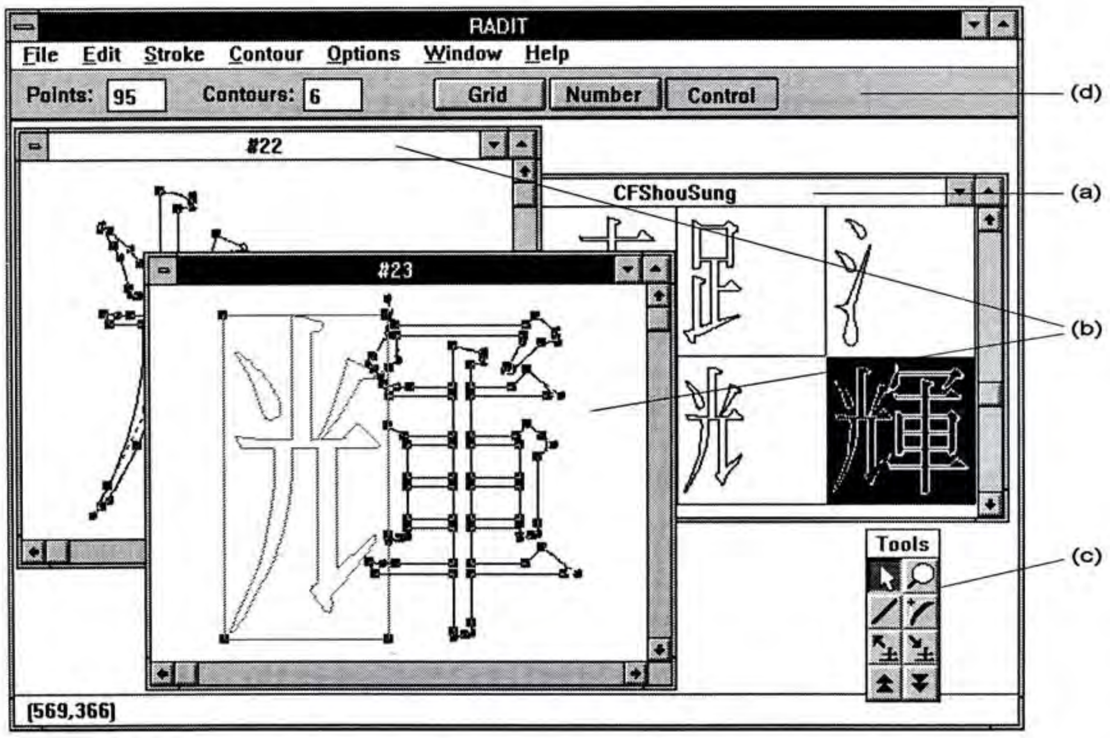
Only horizontal stroke and vertical stroke hinting is considered in this chapter. In the future, we should consider hinting of other features of Chinese characters. For example, we have to consider hinting white space so that spaces between strokes are uniform even at low resolution. We also have to consider hinting of serifs to make them look uniform in a character and to make them disappear when the resolution is too low to accommodate serifs.

# Chapter 5
# RADIT – A Chinese Font Editor

## 5.1. Introduction

RADIT (RAdical eDITor) is a Chinese font editor which supports font composition by radicals. It was built to demonstrate the ideas in this thesis. As a Microsoft-Windows application, RADIT lets the user edit character outlines, and add and remove radicals from characters. It also has tools for stroke recognition and regularization and an internal rasterizer to examine the result of bitmaps produced from outlines.

Figure 5.1



Screen layout of RADIT. (a) Character selection window, (b) Character windows, (c) Tools palette, and (d) Toolbar.

## 5.2. RADIT basics

The font format of RADIT is based on TrueType format [10] with the extension to encoding usage of radicals. Character shapes are described by their outlines, which consist of series of contours.

The contours are composed of straight lines and quadratic Bezier curves. There are two types of control points, those that are on the curve and those that are off the curve. Straight lines are defined by two consecutive on curve points, while Bezier curves are defined by a sequence of on-off-on points.

Figure 5.1 shows the screen layout of RADIT. Four parts of the RADIT screen are of most interest to the user. They are the Character selection window, Character window, the Tools palette and the Toolbar.

### 5.2.1. Character selection window

The main Character selection window shows all characters in the font. A user uses the scroll bar to bring up the characters when necessary. Double click at a character will open a new Character window for it.

RADIT does not distinguish ordinary characters from radicals. All of them are shown in the Character selection window. Whether a shape is a character or a radical depends on how they are used.

### 5.2.2. Character window

The character window is the working place of RADIT. In the Character window you may inspect and edit a character. You may also add and remove radicals from the character.

In the Character window, control points are shown along with the outline (unless you turn off the CONTROL button). On curve points appear as black squares and off curve points appear as black circles. These symbols are called *handles.* We manipulate an outline by selecting the handles, moving them and applying different operations to them.

Besides contours, the character may also contain radicals. Radicals appear in gray color so that they can be distinguished from other contours. Every radical has a bounding rectangle around it. We may move or scale the radical by moving the handles on the bounding rectangle. However, the shape of the radical cannot be edited in the character.

### 5.2.3. Tools Palette

The tools palette is a handy window which lets a user choose tools or apply operations quickly. The tools palette can be set to be visible or hidden from the Option menu. The functions of the eight buttons on the tools palette are:

| | |
|---|---|
| Pointer tool: | Selecting and moving handles |
| Magnifying glass: | Zooming into an area |
| Line tool: | Adding straight lines to the character |
| Bezier tool: | Adding Bezier curves to the character |
| Copy Radical: | Copying radical index to the clipboard |
| Paste Radical: | Pasting radical index into the character |
| Zoom out button: | Zooming out by 50% |
| Zoom in button: | Zooming in by 50% |

### 5.2.4. Toolbar

The toolbar has some buttons which change the display attribute of the current window. It also shows the status of the current character including the number of contours and the number of control points used. The toolbar can be set to be visible or hidden from the Option menu.

The GRID button on the toolbar toggles the display of a grid. The grid is a guide to help the users to adjust the outline. The NUMBER button toggles display of sequence number of the curves. The CONTROL button toggles the display of handles. It helps to preview the outline with the CONTROL turned off so that the handles do not distract the character. However, editing is disabled when CONTROL is off.

### 5.2.5. Zooming the character window

When a character windows is first opened, a character is shown as fit-in scale such that the whole em-square fills the window. Zooming functions let a user inspect details of a character. There are two ways to control the scale of the character displayed. First, simply click at the Zoom-in or Zoom-out button on the Tools palette to set to the desire viewing scale. Secondly, you may choose the magnifying glass on the tools palette and select the area to inspect. The magnifying glass helps to zoom-in an area precisely. To return to fit-in window scale, simply double click the left mouse button.

## 5.3. Editing a character

RADIT uses an interactive interface for editing. Most operations are done by clicking and dragging with the mouse. We edit a character by moving, adding and deleting control points. For most operations, we have to select the control points for the operation first.

### 5.3.1. Selecting handles

To perform operations on one or more control points, the points have to be selected first. A selected handle will appear as a hollow square or hollow circle as oppose to an unselected handle which appears as a solid square or solid circle. To select a single handle, simply click on it. Every time an item is selected, all previous selections will be reset automatically. If you want to select multiple points without resetting previous selections, hold down the SHIFT key on the keyboard while making the selection. A short cut to select a whole contour is to double click on any one of its handle. Holding the SHIFT key while double clicking with select a contour without removing previous selections.

To reset previous selections, click at any empty space.

### 5.3.2. Adding lines and curves

To add straight lines to the character, first choose the line tool by clicking the Line tool button on the Tools palette. Then click at the location for the end point of the line. Successive mouse clicks will append lines from the last point to the location clicked at.

To add Bezier curves to the character, first choose the curve tool by clicking the Curve tool button on the Tools palette. Then click at the location for the end point of the curve. Successive mouse clicks will append curves from the end point to the location clicked at. Note that the curvature of the new curves is initially zero. Their curvatures can be adjusted by moving the off curve control points.

Instead of constructing a new contour, sometime it is necessary to append lines and curves to an opened contour. This can be done by first selecting the end point of the opened contour. The lines or curves will be appended to the selected point.

To close a contour, add the last line or curve from the last point to the starting point so that the end points join together.

### 5.3.3. Delete control points

To delete any control point, select it and then press the DELETE key. Several control points can be deleted at once.

If the deleted point is in a closed contour, the contour will be opened after the deletion. If the deleted point is in the middle of an opened contour, the contour will be broken into two opened contours after the deletion.

### 5.3.4. Moving control points

To adjust the position of a single control point, just drag its handle to the desired location. Multiple points can be moved at once. Select all handles to be moved and drag any of these selected handles. All selected points are moved by the same distance.

### 5.3.5. Cut and paste

Cut and paste in RADIT is applied at contour level. That means if any point on a contour is selected, the whole contour is selected for the operation.

To copy some contours, select them and choose the Copy command from the Edit menu or use the keyboard equivalent CTRL-INSERT. A copy of the selected contours will be made in the clipboard. To cut some contours, select them and choose the Cut command in the Edit menu or use the keyboard equivalent SHIFT-DELETE. Cut is similar to Copy except the selected contours will be removed from the character. To paste the contours stored in the clipboard, choose the Paste command from Edit menu or use the keyboard equivalent SHIFT-INSERT. The contours in the clipboard are added to the current character. The content of the clipboard is unchanged after the Paste command.

### 5.3.6. Undo

RADIT supports one level of undo. Any editing mistake can be undone by choosing the Undo command in the Edit menu or use the keyboard equivalent ALT-DELETE. You can reverse the Undo by choosing the Undo command again.

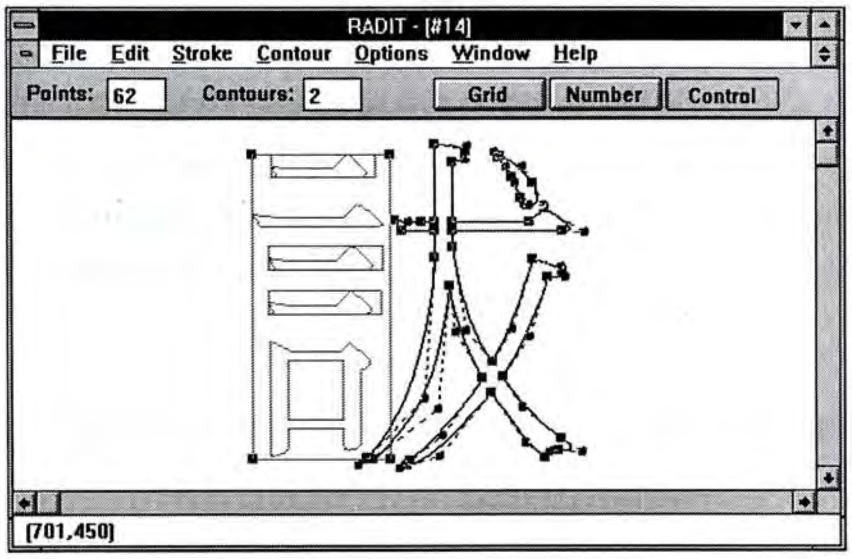
## 5.4. Adding radicals to a character

Radicals can be added and removed from a character with cut-and-paste like commands. It is most convenient to copy radicals from the Character selection window so that you can easily browse through the available radicals.

The procedure for adding radicals to a character is very simple. First select a radical by clicking it in the Character selection window. Use Copy Radical command in the Edit menu to copy the radical. Then choose the window of the destination character and use Paste Radical command in the Edit menu. Unlike ordinary cut and paste, the outline of the radical is not stored in the clipboard. The index to the radical is stored instead.

To remove a radical from the character, select the radical and choose Cut Radical command from the Edit menu. If you do not want to cut the index of the radical to the clipboard, simply pressing the DELETE key will remove the radical without affecting the clipboard.

Figure 5.2



Radical in a character

A radical in a character appears in gray with a bounding rectangle around it. The outline of the radical cannot be edited in the character but it can be moved and scaled.

To move the radical, select all four handles on the corners of the bounding rectangle by double clicking any of them. Then drag the bounding rectangle to the new location. The radical is moved with the bounding rectangle. To scale the radical, drag one corner of the bounding rectangle to the desired shape. The radical is scaled with its bounding rectangle.

Radicals can be cut and pasted along with the outline. The procedure is the same as ordinary cut and paste.

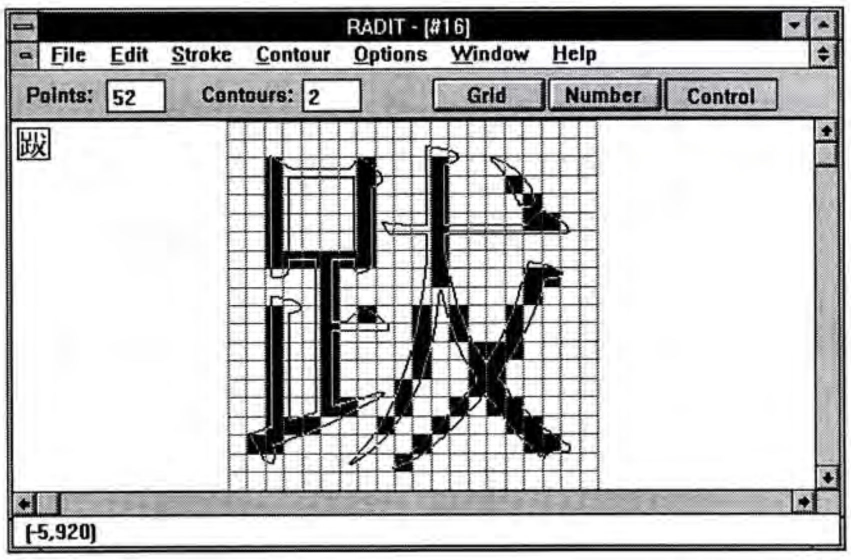
## 5.5. Rasterizing and grid-fitting a character

After you have finished editing the outline of a character, you may want to test its result in rasterization. RADIT provides three functions for rasterization and grid-fitting.

### 5.5.1. Rasterizing a character

RADIT provides a built-in rasterizer for examining the quality of bitmaps produced. The rasterizer will output two views of the bitmap. In the left view, the character is rasterized in true scale. In the right view, the resultant bitmap is displayed on top of the outline so that you can examine the relation between the outline and the grid.

Figure 5.3



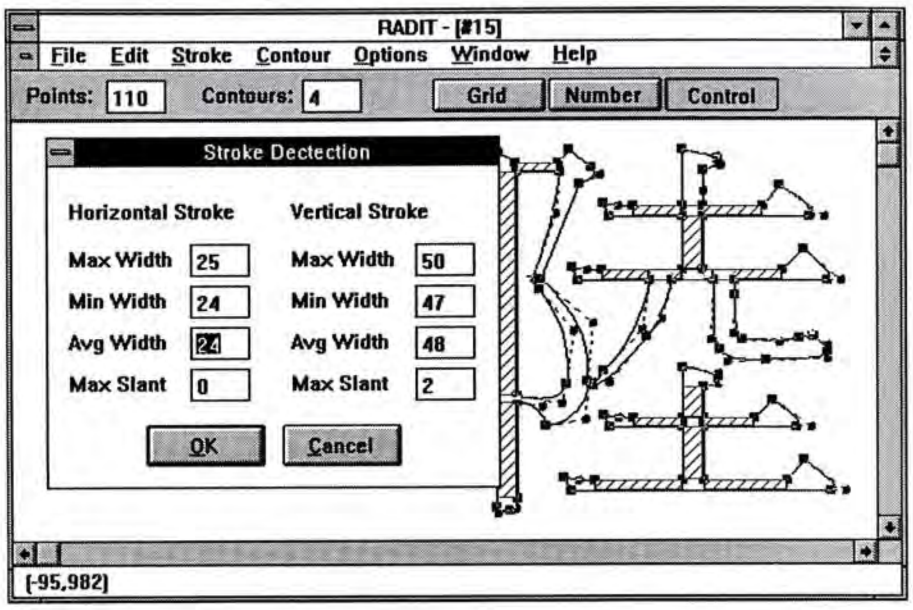
Rasterize with the left view and the right view

The Rasterize dialog box contains choices that control the rasterization effect. Enter the resolution of the bitmap to be rasterized in the Resolution box. Pixel size box controls the size of pixel of the left view. The default value is one, which produces bitmap in true scale. Entering values larger than one will zoom the bitmap. The check boxes below control features to be drawn in the right view. Check the Grid box to draw a grid on the right view; check the Outline box to draw the outline of the character on the right view and check the Pixel box to choose whether or not to draw pixels on the right view.

### 5.5.2. Stroke detection and regularization

Before a character can be grid-fitted, its strokes must be detected and regularized. The Stroke Detect command will automatically recognize strokes in the character. The statistics of stroke widths detected are shown in the Stroke detect dialog box. By default, the stroke widths will be adjusted to the average width detected. You may change the value in the Average width boxes to adjust the widths to the desired values. Press the OK button to adjust the stroke widths or CANCEL button to ignore them. The adjusted stroke widths are recorded in the character for use in grid-fitting.

Figure 5.4



Points: 110 Contours: 4

Stroke Dectection

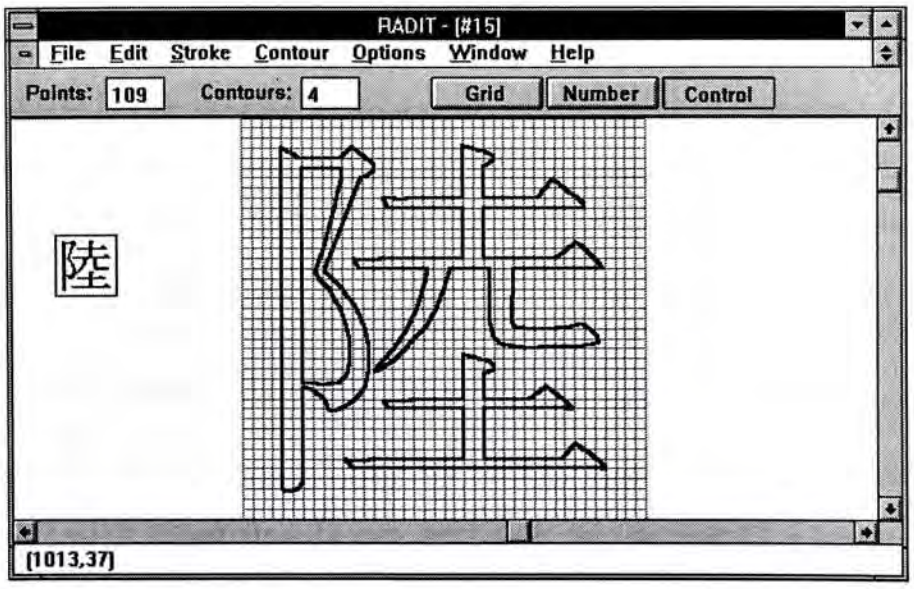
| Horizontal Stroke | | Vertical Stroke | |
|---|---|---|---|
| Max Width | 25 | Max Width | 50 |
| Min Width | 24 | Min Width | 47 |
| Avg Width | 24 | Avg Width | 48 |
| Max Slant | 0 | Max Slant | 2 |

Stroke Detection dialog box

### 5.5.3. Grid-fitting and rasterizing a character

Grid-fitting is similar to the Rasterize command. It will first grid-fit the outline and then show the result as in the Rasterize command. The outline must have been regularized by the Stroke Detect command in order to obtain a desirable effect.

Figure 5.5



Points: 109 Contours: 4

Grid-Fitting command

# Chapter 6
# Conclusions

Sample characters composed by radicals are shown in appendix A. A comparison of storage requirement between pure outline representation and composition by radical representation is also given in appendix A.

The three goals stated in chapter 1 can be accomplished by composition by radicals. First, font storage is reduced significantly. As the decomposition is done at high granularity (Bushou and Shengpang), the size of encoding a transformation is negligible compare to the size of the components it reuses. Chapter 2 has estimated that the reusable rate of Bushou is very high and the reusable rate of Shengpang is moderate. The overall storage of the composed font is estimated to be 1/3 to 1/4 of the equivalent outline representation.

Secondly, the quality of font remains high. Since the radicals are only scaled by small amount, the distortion of the shape of the radicals are slight. Moreover, our hinting algorithm in rasterization can compensate for the distortion of horizontal and vertical strokes which results in scaling the radicals.

Thirdly, the font output efficiency is not impeded. Transforming the radicals and grid-fitting require only simple multiplication and additions. The impact on rasterization time is very small.

We find that hinting is a very powerful tool. The character image produced is largely influenced by hinting. It does not only improve the quality of images of characters at different resolutions, it can also repair the distortion due to scaling the radicals. Although only horizontal stroke and vertical stroke hinting is considered, we may extent the idea to hint other features like hinting of serifs and slanted strokes. In many cases, imperfections in outline can be fixed by hinting so that fine bitmap is still produced.

We conjecture that our approach for compacting Chinese outline fonts is feasible although more details can probably be uncovered when a full scale engineering task to compact complete sets of Chinese fonts is carried out. In any event, our work indicates a clear path for such work which has failed several times due to the lack of understanding of the Chinese characters.

# Appendix A
# Sample Fonts

10 point samples

堆擠躋濟諞騙騮蹓葆堡塾熟湯場揚詼連陣揮逍

屌陪煮著進焦售萑踓騅堆擠躋濟諞騙騮蹓葆堡

12 point samples

掊踣培陸跋浤拔�松踛涉塾熟湯場揚詼連陣揮逍

屌陪煮著進焦售萑踓騅屌陪煮著進焦售萑踓騅

18 point samples

掊踣培陸跋浤拔跕踛涉塾熟湯場揚詼連陣揮逍

屌陪煮著進焦售萑踓騅堆擠躋濟諞騙騮蹓葆堡

36 point samples

騙騮蹓葆堡浤拔跕踛涉

塾熟湯場揚詼連陣揮逍

屌陪煮著進焦售萑踓騅

堆擠躋濟諞騙騮蹓葆堡

Twenty-eight radicals are used to compose the 40 sample characters above. Thirteen of them are Bushous and fifteen of them are Shengpangs.

Bushous:

訁 阝 𧾷 氵 扌 辶 厂 灬 艹 凵 馬 土 土

Shengpangs:

犮 奎 步 車 尚 者 隹 隹 齊 扁 留 保 孰 易 音

Statistics of the storage required are given below. We choose the control point as the unit of storage requirement. For each stroke, the number of control points used to describe it is given first. Then we count the number of usage of the radicals in 40 sample characters. For each usage, 3 units of storage are used to store the parameters. The total storage equals the number of control points used to describe all radicals and the parameter overhead for all uses of the radicals in 40 sample characters, which sums to 2059 units.

When the sample characters are represented by pure outline representation, the storage required is the total of the control points used to describe each radical, multiplied by the number of uses of that radical. The storage requirements of the pure outline equivalent sums to 5130 units. For these 40 sample characters, the storage ratio between these two methods is 100 : 40.

| Radical | # control point | # usage | usage overhead | storage required | outline equivalent |
|---|---|---|---|---|---|
| 言 | 55 | 2 | 6 | 61 | 110 |
| 阝 | 30 | 3 | 9 | 39 | 90 |
| ⻊ | 52 | 7 | 21 | 73 | 364 |
| 氵 | 40 | 4 | 12 | 52 | 160 |
| 扌 | 40 | 5 | 15 | 55 | 200 |
| 辶 | 55 | 3 | 9 | 64 | 165 |
| 尸 | 27 | 1 | 3 | 30 | 27 |
| 灬 | 49 | 3 | 9 | 58 | 147 |
| 艹 | 43 | 3 | 9 | 52 | 129 |
| [illegible] | 22 | 1 | 3 | 25 | 22 |
| 馬 | 98 | 3 | 9 | 107 | 294 |
| 扌 | 30 | 3 | 9 | 39 | 90 |
| [illegible] | 27 | 2 | 6 | 33 | 54 |
| 犮 | 62 | 3 | 9 | 71 | 186 |
| 奎 | 79 | 3 | 9 | 88 | 237 |
| 步 | 57 | 2 | 6 | 63 | 114 |
| 車 | 73 | 3 | 9 | 82 | 219 |
| 尚 | 85 | 2 | 6 | 91 | 170 |
| 者 | 78 | 2 | 6 | 84 | 156 |
| 隹 | 68 | 4 | 12 | 80 | 272 |
| 隹 | 65 | 3 | 9 | 74 | 195 |
| 齊 | 138 | 3 | 9 | 147 | 414 |
| 扁 | 74 | 2 | 6 | 80 | 148 |
| 留 | 96 | 2 | 6 | 102 | 192 |
| 保 | 84 | 2 | 6 | 90 | 168 |
| 孰 | 139 | 2 | 6 | 145 | 278 |
| 易 | 83 | 3 | 9 | 92 | 249 |
| 音 | 70 | 4 | 12 | 82 | 280 |
| Total | 1819 | 80 | 240 | 2059 | 5130 |

Table A.1: Statistic for the 40 radicals used

# References

1. Chan Y. S., Chin F., Leung S. K. Chinese character generator. Proceedings of the 1988 International Conference on Computer Processing of Chinese and Oriental Languages, 1991, pp. 51-56.

2. Donald E. Knuth. The METAFONT book. Addison Wesley, Reading, Massachusetts, 1986.

3. Dong Y. M. and Li K. D. Introduction to function and structure of Chinese character design systems CCDS. Proceedings of the 1991 International Conference on Computer Processing of Chinese and Oriental Languages, 1991, pp. 51-56.

4. Dong Y. M. LCCD - a Language for Chinese Character Design. Software – Practice and Experience, vol 11, 1981, pp. 1273-1292.

5. Dyson P. Building a cast of characters: Three font-making programs. The Seybold report on desktop publishing, vol 6, no 6, February 1992.

6. Hersch R. D. and Betrisey C. Advanced grid constraints: Performances and limitations. Raster imaging and digital typography II, Morris R. A. and Andre J. ed., Cambridge University Press, 1991, pp. 190-204.

7. Hobby J. D. and Gu G. A Chinese METAFONT. TUGBoat, vol 5, no 2, 1984, pp. 1-136.

8. Hobby J. D. Generating automatically tuned bitmaps from outlines. Journal of the ACM, vol 40, no 1, January 1993, pp. 48-94.

9. Karow P. Automatic hinting for intelligent font scaling. Raster Imaging and digital typography, Andre J and Hersch R. D. ed., Cambridge University Press, 1989, pp. 232-241.

10. Microsoft Corporation. TrueType font files draft release version 1.00, 1990.

11. Moon, Y. S. and Shin, T. Y. Chinese fonts and their digitization. EP90, Furuta R. ed., Cambridge University Press, 1990, pp. 235-248.

12. Ou C. and Ohno Y. Font generation algorithms for kanji characters. Raster Imaging and digital typography, Andre J and Hersch R. D. ed., Cambridge University Press, 1989, pp. 123-133.

13. Rubinstein R. Digital Typography: An introduction to type and composition for computer system design. Addison Wesley, Reading, Massachusetts, 1988.

14. Suen C. Y. and Huang E. M. Computational analysis of the structural compositions of frequently used Chinese characters. Computer Processing of Chinese and Oriental Languages, vol 1, no 3, May 1984.

15. Wang X. Development of Chinese electronic publishing system. Proceedings of 1990 International Conference on Computer Processing of Chinese and Oriental Languages, 1990, pp. 272-278.

16. 周有光：《漢字聲旁讀音便查》，吉林人民出版社，1980。

17. 孫鈞錫：《漢字通論》，河北教育出版社，1988。

18. 傅永和：《漢字結構及其構成成份的統計及分析》，《現代漢語定量分析》，陳原，上海教育出版社，1989。

19. 新形像出版公司編輯部：《中國文字造形設計》，北星圖書公司，1987。