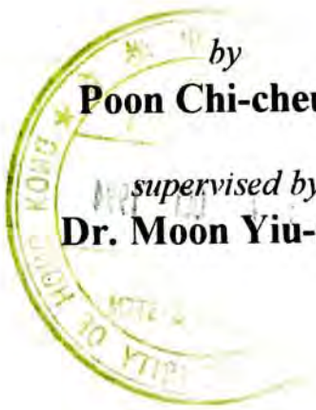
A New Approach to the Generation of Gray Scale Chinese Fonts

by
Poon Chi-cheung

supervised by
Dr. Moon Yiu-sang



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

Master of Philosophy
in
The Department of Computer Science

The Chinese University of Hong Kong

Hong Kong
June, 1993

Table of Content

Abstract

Gray scale fonts have been applied in English characters and proved to be superior in quality than binary bitmap fonts on CRT display. We have tried to produce gray scale Chinese fonts in our research, by a commonly used technique for English gray scale font generation called "filtering method". Preliminary study on gray scale Chinese fonts reveals that certain problems arise due to the characteristics of Chinese characters. A new approach, called gray scale rasterization, is introduced in this thesis. Gray scale rasterization generates gray scale bitmap directly from the character's outline description. It consumes less time than undergoing filtering method which consists of two steps: rasterization to obtain binary character bitmap from its outline description and then filtering process to generate gray scale character from the bitmap. Some implementation technique of gray scale rasterization is described in this thesis. The approximation of Bezier curves by straight lines and the simplification of filling process improve the speed of gray scale rasterization, whereas stroke migration improves the quality of the resultant font. At this stage, the result obtained is quite satisfactory, yet there is still rooms for further improvement. However, the idea of this thesis is only the first attempt. We hope that after some modifications, gray scale Chinese display can be brought to real life application.

Acknowledgment

I am very glad to take this opportunity to dedicate my gratitude towards the following people. Firstly, I would like to give many thanks to Dr. Y. S. Moon, the supervisor of this research project, for contributing ideas to the topic, providing information and related papers, as well as giving guides and comments. Secondly, Dr. Yuk-bun Szeto, who offers me precious advice on the study of rasterization. And Mr. Chong-kan Chiu, who helps me in obtaining outline font source data as experiment input. Finally, a good friend of mine, Sau-ying, for proofreading this thesis and giving me encouragement. Without their help, this research will not be in its progress.

Preface

Due to the fact that a vast amount of information is represented in the form of text, one of our primary means of communication is printed text. On the other hand, as computer technology develops immensely, people become eager to use computer in various fields. Applications on computer-aided design, publishing and WYSIWYG (What You See Is What You Get) word processor require high quality text display. Thus, high quality text output by computer devices is particularly demanding. Traditional black-and-white bitmap font often shows a weakness of staircasing along the edges of the character strokes. Therefore, it is not a suitable choice to provide high quality text.

The resizable outline fonts with the use of high resolution laser printer has enhanced text printing quality to a large extent. But they are still weak on CRT displays because of the low resolution of these devices. The application of gray scale fonts is a possible solution to this problem. Gray scale fonts are proved to be favourable to improve quality of English characters on screen display. This happens because of the reduction of contrast between the character edges and the background colour by using various gray pixels on stroke edges instead of black ones only.

Our interest is to apply gray scale technology to improve Chinese character representation on CRT displays. Certain characteristics of Chinese, such as large character set and congested strokes of character shape, do not exist in English characters. So, the gray scale techniques used to generate gray scale English character may not adopt for Chinese. Some investigations are performed to verify it.

The common method to generate gray scale font is "filtering". A high resolution black and white image is re-sampled by a filter to generate a low

resolution gray scale character. Since the process involve large amount of computations, it is not suitable for "ad hoc" generation. Storing the pre-computed gray scale character images is a possible solution for English character, but it is not feasible for Chinese because of the large character set.

A new approach to generate gray scale Chinese character is introduced in this thesis. A gray scale rasterizer is built which generate gray scale characters directly form characters' outline descriptions. It provides a possible way for real-time generation of gray scale characters.

Some technique is used to implement the gray scale rasterizer to improve the generation speed and quality. Approximation of quadratic Bezier curve by straight lines and simplification of the filling process avoid the complicated floating point arithmetic such as root finding and quadratic equations solving. The rasterization speed is improved much by applying these two operations. The application of stroke migration can overcome the problem of different impression of strokes with the same width to a certain extent.

Results obtained through a series of experiments are presented in photographs in this thesis. A point worth to note is that the effect of gray scale may not shown its best advantage in photographs; there is no substitute for viewing gray scale text directly on a well-calibrated CRT.

This thesis consists of seven chapters. Chapter 1 gives a brief introduction on font systems while Chapter 2 discusses human visual system and gray scale font. Chapter 3 explain the digital filtering method for gray scale font. The new approach to generate gray scale Chinese font directly from a gray scale rasterizer is introduced in Chapter 4 and 5. Chapter 6 presents the results and makes a conclusion.

Chapter 1

Font Systems

There are many different forms of information (such as text, picture and sound) storing and processing in a computer system. Among them, text is the most commonly used. Various encoding methods are created to store text information. ASCII (American Standard Code for Information Exchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code) are two well-known ones for coding English characters. For Chinese character, usually GB coding system is used in Mainland China and Big-5 coding system in Taiwan. All these encoded texts must be converted to character images showing on output devices so that they become meaningful to human readers. A font system is such a system that provides character images for display and printing.

Representations of Character Images

There are two categories of character representation formats : bitmap and outline. To use bitmap format means to store the shape of a character as a array of bits (Figure 1.1). It is the simplest way to represent fonts in most output devices, as the dots of a bitmap matrix can be directly mapped to pixels on screen or printer. While its output process is generally very simple and fast, it has a shortcoming of consuming a sizable amount of computer memory. The storage required increases as the square of the bitmap size. Thus, it is not suitable for computer system with high resolution output devices, such as laser printer.

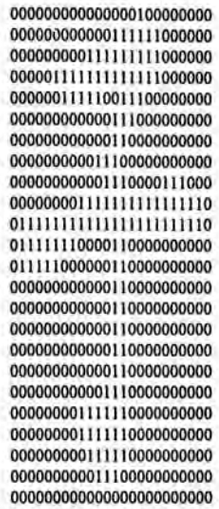


Figure 1.1 Font data stored in bitmap format

When outline format is concerned, character shape is described by the character's edges (Figure 1.2). The outline of a character is composed of such continuous functions as straight lines, arcs, natural spline curves, Bezier spline curves, etc. which are stored in the form of mathematical parameters. For example, a straight line is stored by the coordinates of its two end points. Outline format appears to be superior in quality than bitmap format because an outline character can be scaled to any size without loss of smoothness and completeness. However, when the outline font is to be output to some raster-scan devices, such as monitor and printer, it must be converted to bitmap image, which will be mapped onto the pixels of output devices. This process is called "rasterization" [Hersch 89, Deach 92]. Time is needed to convert outline description into bitmap before output, but the output speed is satisfactory when cache memory is provided to save the bitmaps generated for further use [Moon and Cheang 91]. In this approach, the character images are stored in both bitmap and outline formats. The character images with high usage frequency are to be stored in bitmap format, which can provide an advantage of fast display/printing the character images. The other character images are to be stored in outline format, which can provide different font sizes with limited storage.

Figure 1.2 Character stored in outline format

Characteristics of Chinese Font System

Large Character Set

In English, words can be formed by concatenation of alphabets. But it is not the case for Chinese "words". Each Chinese character alone is a "word", which has its own meaning. It is estimated that at least 50,000 to 60,000 characters have been used since their invention a few thousand years ago [Tung 81].

There are two main coding schemes to encode Chinese characters as 16-bit computer codes. One is the Big-5 Chinese Coding Scheme used in Taiwan. It encodes 13,053 characters including 5,401 commonly used and 7,652 less commonly used ones. The other is the GB Coding Scheme used in Mainland China. It covers about 7,000 commonly used Simplified Chinese characters. The character set of Simplified Chinese character is smaller because some characters are combined into a new simplified one. Obviously, there may be more than thousands of rarely used Chinese characters not included in these two standards. They can be added to the system as user defined codes.

The large character set of Chinese language brings a storage problem in bitmap font systems. Since each Chinese character is formed pictorially and has a distinct description in the font library, all character images must be stored. The required storage space for different bitmap size and number of character is shown in Table 1.1. If a font library of bitmap size 48x48 including 13,053 characters is built, a storage size of 3.6 mega bytes is need for one typeface. Thus, it is practically impossible to create font libraries of bitmap fonts in various sizes and styles.

Number of Character	Bitmap Size				
	16x16	24x24	48x48	64x64	128x128
5400	169 K	380 K	1.48 M	2.64 M	10.55 M
7000	219 K	492 K	1.92 M	3.42 M	13.67 M
13053	408 K	918 K	3.59 M	6.37 M	25.49 M

1 K Bytes = 1024 bytes
1 M Bytes = 1024 K Bytes

Table 1.1 The storage space of Chinese bitmap font library (All storage measured in bytes)

Condensed Strokes

High stroke count is another feature of Chinese characters. Characters with as many as 15 strokes are not uncommon (Table 1.2). Some researches have found that the average number of strokes in a Chinese character is 12.7 [Fu 89]. Thus, Chinese fonts are obviously more densely packed than English fonts (Figure 1.3). This property makes rasterization of outline fonts very difficult, especially if small bitmaps for visual display purposes are required.

Stroke Number	Percentage (%)
1-5	3.207
6-10	30.712
11-15	40.963
16-20	19.506
21-25	5.001
26-30	0.569
above 30	0.042

Table 1.2 The distribution of stroke number of Chinese characters (Data extracted from [Fu 89])

Figure 1.3 Examples of Chinese characters with different number of strokes

Since the outline description of Chinese character is complicated, its size is not necessarily smaller than that of small sized bitmap fonts. For example, the size of the outline font library (without hinting information) coming with Microsoft Chinese Window 3.0 is about 6 mega bytes, while that of a commonly used Chinese system 24x24 bitmap font library is 918 Kbytes only.

Low Repetition Rate

Beside having large character sets, the average repetition rate of Chinese character is far less than that of English characters [Moon and Shin 90]. The cumulative usage distribution of Chinese characters is shown in Figure 1.4 [Bei 85]. This uneven distribution of the usage frequencies of the Chinese characters brings a problem on outline font systems. For English, font caching has been applied in typesetting and laser printing to reduce the time spent on rasterization [Fuchs and Knuth 85]. In Chinese, the font caching strategy is not very straightforward because of the low character repetition rate. Nevertheless, the careful use of cache memory can help a lot [Moon and Cheang 91].

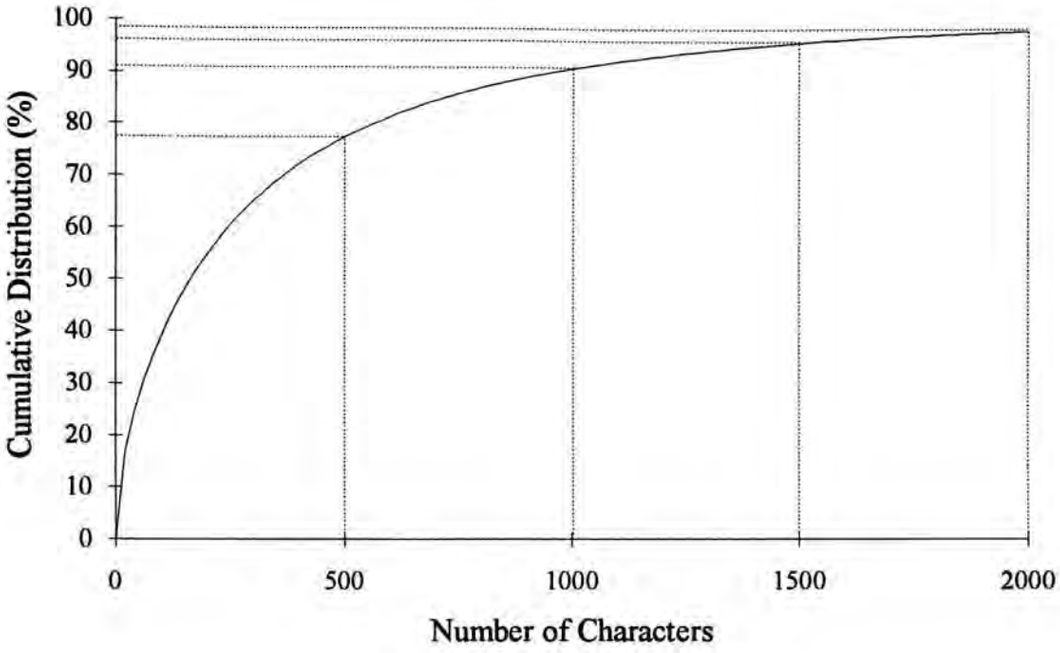


Figure 1.4 Cumulative usage distribution of 2000 Chinese characters (Data extracted from [Bei *et al.* 85])

WYSIWYG (What You See Is What You Get)

The text information will be read on screen and also read as printer output on paper. The relation between screen text and printer text is the subject of intensive research, with recent efforts attempting to integrate screen and printer in ways described as WYSIWYG (What You See Is What You Get) [Bigelow 85].

The WYSIWYG principle is that the screen should show exactly the look of the printed document. The resolution of output devices vary from 72 dpi (dot per inch) display monitors to 300 - 600 dpi laser printers, even to 2000 - 4000 dpi typesetters. To standardize the size of output character for human view, a resolution independent measurement is needed. A commonly used unit to measure the font size in printout is *point*. One point measures 0.013837 inches. Thus an inch contains approximately 72.27 points and a point is taken to be 1/72 of an inch [Rubinstein 88]. The size of bitmap for given point size and device resolution is calculated by

$$\text{size of bitmap} = \frac{\text{device resolution} \times \text{point size}}{72.27}$$

To achieve the WYSIWYG effect, two sets of character images of the same shape but different sizes are needed. Suppose a user has a display monitor with resolution 72 dpi and a laser printer with resolution 300 dpi. If he requires a font with point size 24, a pixel size of 24x24 character image is used for display monitor while a 100x100 character image is needed for laser printer. For a bitmap font system, a point size is available when the font library contains character images for both display monitor and printer (Figure 1.5). As mentioned above, the required storage is very large, especially when high resolution printer is used.

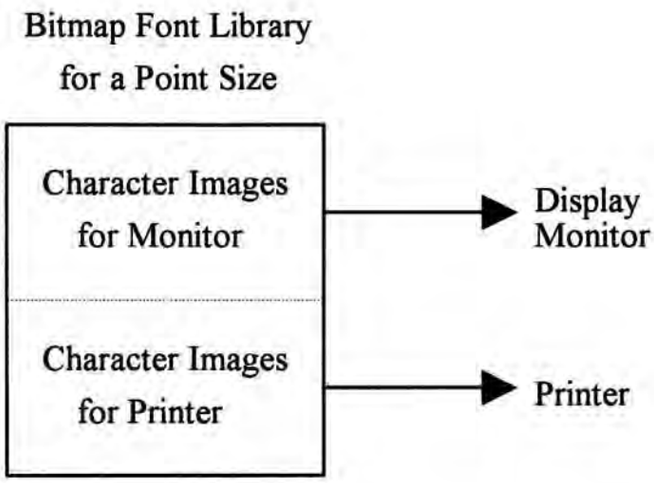


Figure 1.5 A font system using bitmap font library

For an outline font library, one set of character outline descriptions is enough for each type style because of the scalable characteristic of outline font. To obtain different sizes of character image bitmaps for output devices with different resolution, we just require the rasterizer to generate them (Figure 1.6). For example, if a system has a display monitor with resolution 72x72 dpi and a laser printer with resolution 300x300 dpi, when a user requires a font with point size 24, the rasterizer will generate a 24x24 character image for display monitor and a 100x100 character image for the laser printer.

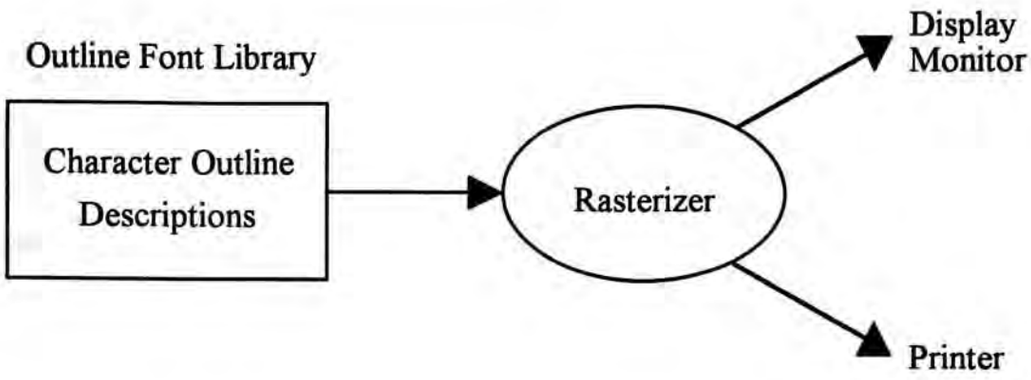


Figure 1.6 A font system using outline font library

Due to the limitation of low resolution of screen display, the character images displayed on screen does not correspond exactly on paper printout. Using gray scale technique will virtually increase the resolution of the display devices [Rubinstein 88]. It can improve much to achieve the WYSIWYG effect.

Chapter 2

Human Visual System and Gray Scale Font

Human Visual System

Physiology

In the human eye, the cornea, iris, pupil, and lens comprise an optical system that forms an image on the retinal surface. The aperture of the refracting system of the eye is controlled by the iris. It contains muscles that control the opening and closing of the pupil. The size of pupil can vary in diameter from 2 to 8 millimeters. This variation helps to maintain the amount of light reaching the retinal at a roughly constant level and optimizes visual acuity [Campbell and Gregory 60]. The lens is a transparent biconvex body. It focuses the image on the retina by changing its refracting power, that is thickening or thinning its structure depending on the closeness of the object [Davson 90].

Light is focused on the retina and then passes through the highly transparent cells of the retina, where it is absorbed and converted into chemical energy. In the whole human retina, there are said to be about 7 million cones and 75 to 150 million rods photosensitive cells [Davson 90]. The centrally-located region, called fovea, has highest cell density and is populated almost exclusively with colour-sensitive cone cells. The fovea is small compared with the whole visual field, only 1 to 2 degrees of visual angle. Outside the fovea, the retina is populated mostly by rod cells, that can detect light down to very low levels but do not detect colour [Rubinstein 88, Davson 90].

Spatial Frequencies

To discuss typography, it is necessary to talk in terms of the information processing abilities of the human visual system. Spatial frequency, or grating-frequency, corresponds to sampling resolution and is measured in cycles per degree of visual angle. For a given spatial frequency, as the contrast is reduced, the grating becomes more difficult to detect, and finally, it appears as a uniform gray field. The sensitivity of the eye to a specific spatial frequency is the reciprocal of the threshold contrast necessary to detect it. Sensitivity to spatial variation of light and dark depends on the frequency of the variation [Naiman 91]. The relation between contrast sensitivity and spatial frequency is shown in Figure 2.1.

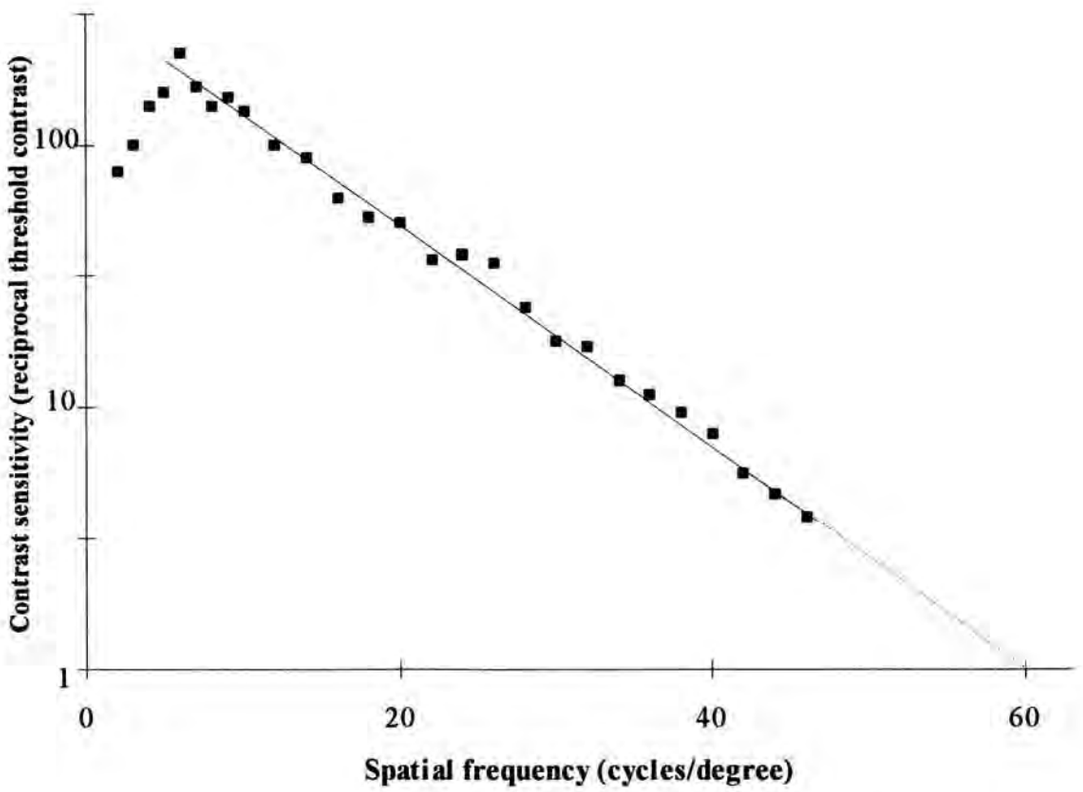


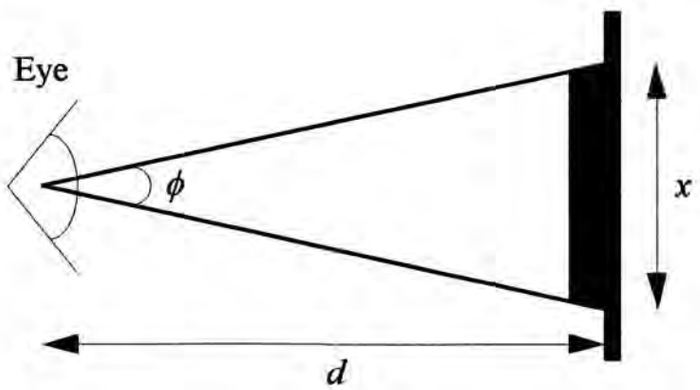
Figure 2.1 The "contrast-sensitivity function" of a human observer [Campbell and Green 65]

Experiments by perceptual psychologists suggest that the visual system cannot detect spatial frequencies greater than 60 cycles per degree of visual angle at unity contrast [Campbell and Green 65]. That is, if the spacing is so fine that more than 60 black and white line pairs are imaged in one degree of visual angle,

the visual system perceives a bar grating of regularly spaced black and white lines as solid gray.

How much resolution is enough

To provide good-quality digital text, the character image must contain as much resolution as the eye and brain can receive and interpret but need not contain more information than that [Bigelow 85]. There is a way to estimate the theoretical minimum resolution for good-quality text. As mentioned in the above section, the human visual system cannot detect spatial frequencies greater than 60 cycles per degree of visual angle, this provides a measure of the upper limits of the visual system's ability to resolve the kind of detail produced by a digital raster. At a reading distance of approximately 12 inches, 60 cycles per degree of visual angle is equal to a resolution of 300 cycles per inch at the screen (Figure 2.2) [Rubinstein 88].



$$x = 2d\tan\frac{\phi}{2}$$

If $\phi = \left(\frac{1}{60}\right)^{\circ}$, and d = 12 inches, then x = 0.0035 inch, or 286 cycles per inch.

Figure 2.2 The relation between visual angle, distance and spatial frequency

We can now estimate the minimum resolution necessary for good-quality digital text by using a principle of digital signal-processing theory developed by Harry Nyquist at Bell Laboratories in the 1920s. It states that a signal can be properly reconstructed from its samples if the original signal is sampled at a

frequency that is greater than twice the rate of the highest frequency in the original signal. This minimum sampling rate is known as the Nyquist frequency or Nyquist rate [Bigelow 85, Foley *et al.* 90]. If we sample below the Nyquist limit, the samples we obtain may be identical to what would have been obtained from sampling a lower-frequency signal. This phenomenon tells why high frequency components of the original signal are erroneously reproduced as spurious lower-frequency components of the reconstructed signal. This is known as aliasing. In digital typography, one form of these aliases is the jaggies -- the jagged stair-step patterns that fringe the edges of digital object. Thus, to faithfully sample and reconstruct a signal of 300 cycles per inch, you need a minimum sampling rate of 600 lines per inch [Bigelow 85].

Screen and Printer

Many devices are in use for computer font output (Table 2.1). Concerning the surface forms in which the output appears, the devices fall into three categories. They are screens, paper and photographic emulsions such as film or photographic paper [Rubinstein 88].

Technology	Type	Colour Output	Approximate Resolution	Typographic Quality	True Gray
Cathode Ray Tube	screen	yes	50 to 300	fair to good	yes
Liquid Crystal Display	screen	no	50 to 80	fair	no
Plasma Display	screen	yes	50 to 100	fair	no
Electrographic Printer	printer	no	about 200	fair	no
Dot Matrix	printer	yes	about 150	fair	no
Ink Jet	printer	yes	240 to 300	fair to good	no
Laser Printer	printer	yes	240 to 600	good to excellent	no
Digital Typesetter[1]	printer	no	725 to 5000	excellent	yes[2]

[1]Photographic paper or film output.
[2]Rarely used.

Table 2.1 Various output devices and their different typographic characteristics [Rubinstein 88]

Raster Display Devices

In the early seventies, the development of inexpensive raster graphics is based on television technology. Raster displays store the display primitives, such as lines and characters, in a refresh buffer in terms of their component pixels, as shown in Figure 2.3 [Foley *et al.* 90]. In some raster displays, there is a hardware display controller that receives and interprets sequences of graphical output commands. But in more common systems, such as those in personal computers, a software, called graphics library package, is used to output image to refresh buffer, which is part of main memory. And the refresh buffer can be read out by the image display subsystem, often called the video controller, that produces the actual image on the screen.

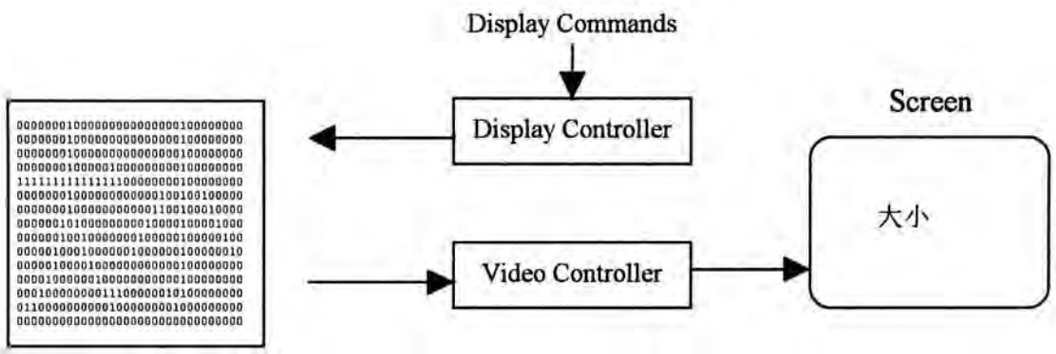


Figure 2.3 Architecture of a raster display

The complete image on a raster display is formed from the raster, which is a set of horizontal raster lines, each a row of individual pixels. The raster is stored in refresh buffer as a matrix of pixels representing the entire screen area. The entire image is scanned out sequentially by the video controller, one raster line at a time, from top to bottom and then back to the top (Figure 2.4). The beam's intensity is set to reflect the pixel's intensity. In colour systems, three beams are controlled--one each for the red, green and blue primary colours--as specified by the three colour components of each pixel's value [Foley *et al.* 90].

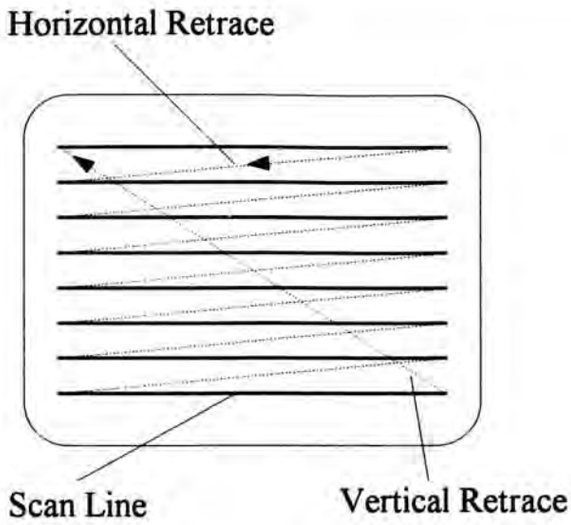


Figure 2.4 Raster scan

Printer

The above two output methods also used for paper output device. The output device using vector method is called a plotter, which can produce outputs that consist of continuous unbroken lines or pictorial forms. It produces its output by controlling pens moving on a 2-dimension plane.

Three types of raster printing devices are in common use (Table 2.1). The first one, dot matrix printers have long met the requirement for low resolution and low cost paper output. Second type is laser printers, which are recently improved and reduced in cost. The laser printers provide better quality printed output for personal computer users. And the last one, at the high end of performance and price, is digital typesetting equipment that provides high-resolution output on photographic paper, creating a master image for printing by offset or other photographic means [Rubinstein 88].

Some older printer devices, like phototypesetters and daisy-wheel printers, use a master image that is transferred by some means onto paper. Phototypesetters do this optically, projecting a master image onto the film or photographic paper. Daisy-wheel printers, of course, transfer the image mechanically. Such devices are excluded by definition from digital typography because the letters are not formed under digital control [Rubinstein 88].

Resolution

Generally speaking, the resolution of screen output is much lower than most of the modern printer output (Table 2.1). The resolution of commonly used display devices is about 72 to 90 dpi (dot per inch), which is far from the resolution required to display a good quality text. To improve the quality of output text on CRT screen, we make use of its ability to display true gray output (Table 2.1). Instead of outputting black-and-white character images, character images with gray scale will be displayed on screen.

For printer devices, gray scale output cannot improve the quality of output text because of the lack of ability of true gray output. These bi-level output devices use a method, called "Dithering", to approximate different intensity levels output. For example, a 2x2 pixel area of a bi-level display can be used to produce five different intensity levels at the cost of halving the spatial resolution along each pixels (Figure 2.5) [Foley *et al.* 90]. It means that a 360x300 dpi bi-level printer can approximate the output of a 180x150 dpi printer with 5 intensity (gray) levels.

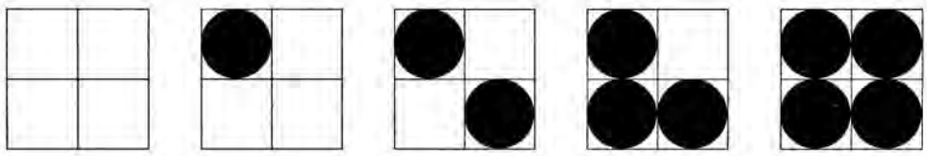
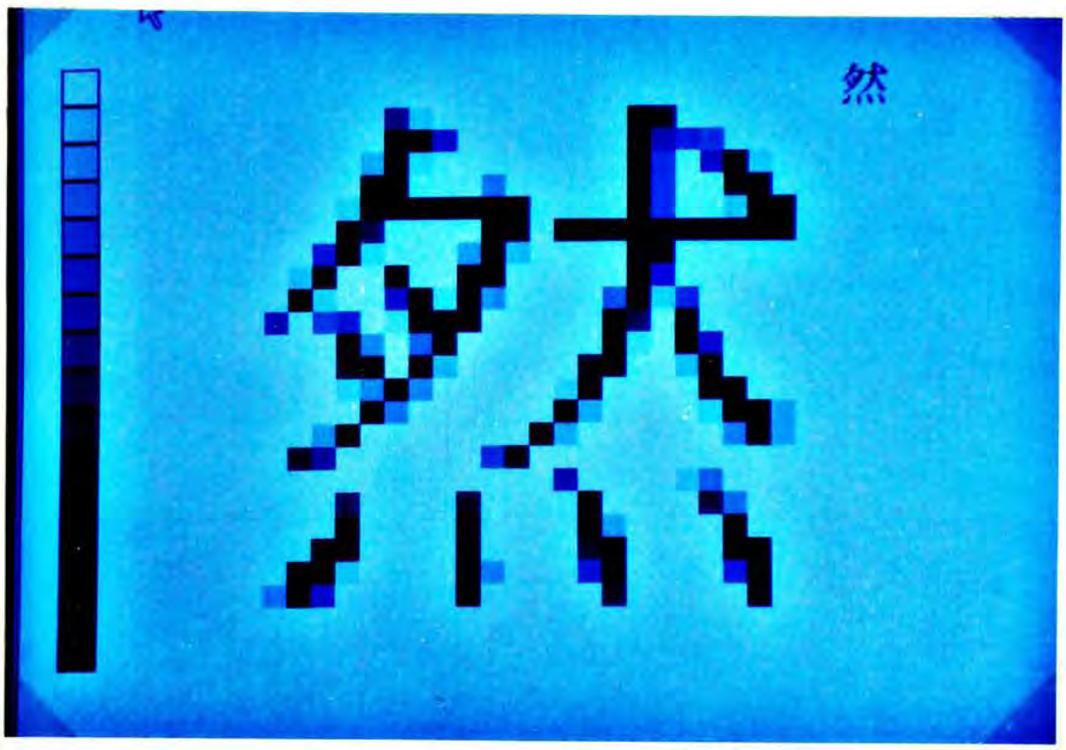


Figure 2.5 Five intensity levels approximated with four 2x2 dither patterns

Gray Scale Font

Beside the need of sophisticated graphical images, applications on computer-aided design and publishing require high quality text display as well. Traditional black-and-white bitmap fonts often show a weakness of staircasing along the edges of the character strokes. The resizable outline fonts with the use of high resolution laser printer has enhanced the text printing quality to a large extent, but they are still weak with CRT displays because of the low resolution of these devices. The application of gray scale fonts is a possible solution to this problem.

In general, Chinese characters displayed on CRT, LEDs are usually represented by black-and-white bitmaps. Each pixel can be either black or white. However, gray scale fonts allow pixels to appear in different levels of gray colour (Figure 2.6). Evidence shows that the application of gray pixels at suitable positions in a English character can produce a visual effect of smoothening the edges of character strokes [Rubinstein 88, Warnock 80]. Gray scale techniques are particularly useful on display devices with very limited spatial resolution.



(a) Size 24x24

Figure 2.6 Gray scale character bitmaps and its enlarged appearance

(b) Size 48x48

Figure 2.6 Gray scale character bitmaps and its enlarged appearance (continue)

Gray value determines the intensity of a pixel. If 16 gray levels are in use, the gray value of the pixel will vary from 0 to 15. The value 0 gives the darkest intensity, i.e. the background colour, while the value 15 gives the brightest intensity, i.e. the foreground colour. The character image of gray scale font is represented by a matrix of gray values. We called it "gray scale bitmap". Actually, more than one bit is mapped to a pixel. The number of bits depends on how many gray level are used. If 16 gray level are in use, 4 bits are mapped to a pixel.

As mentioned on above section, the spatial frequencies sensitivity of the eye depends on contrast. At high frequencies, high contrast is required to allow the eye to distinguish edges. A low-contrast edge of small dimension (high spatial frequency) is not perceived as such, but it does help us perceive the location of the larger object of which it is part [Rubinstein 88]. For English characters, as few as

4 gray levels may be enough to improve appearance [Schmandt 80], but 8 or 16 gray levels give better results [Warnock 80]. Some investigation of applying gray scale on Chinese character is done and will be presented in this document.

Generation of Gray Scale Font

Designing gray scale fonts manually is difficult because of the amount of information that must be specified for each bit. It is difficult for people to invent consistent gray values that create the right effects, although incremental improvements can be made to an existing design [Rubinstein 88]. Good results can be attained computationally from rasterized high resolution fonts. A common method to generate a gray scale character is "filtering" [Warnock 80, Naiman 91, Naiman and Fournier 87]. A large size black and white image is re-sampled by a filtering process to generate a small size gray scale character. Because of speed problem, we suggested a new approach to generate gray scale font. A gray scale rasterizer is built to produce gray scale font directly from outline font data.

Chapter 3

Digital Filtering Method for Gray Scale Font

Filtering Process

The common method for generating gray scale character is "filtering" [Foley *et al.* 90]. A high resolution black and white image is re-sampled by a filter to generate a low resolution gray scale character. The filter is the computations performed to simulate an ideal camera pointing at the black-and-white character (the source). The source is divided into sampling areas, each of which will contribute a resulting gray-scaled pixel to the target character image. Figure 3.1 shows a high precision Chinese character using as source of filtering process. Each circle corresponds to a sampling area, or called filter (Figure 3.2), where W is the width of the filters and d is the distance of centers of two adjacent filters.

The gray value of each pixel is obtained from a function considering the portions of black and white subareas in the sampling area. In particular, if only black/white is within the sampling area, then a black/white pixel is produced. If any combination of black and white colours appears in the sampling area, a gray pixel is produced [Crow 78, Naiman and Fournier 87, Warnock 80].

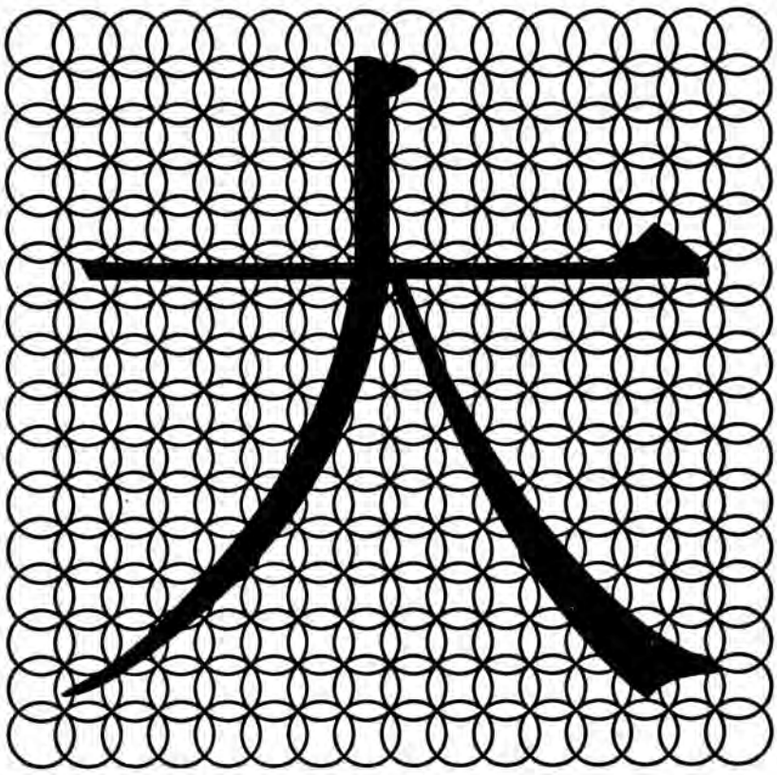


Figure 3.1 A Chinese character using as source of filtering process

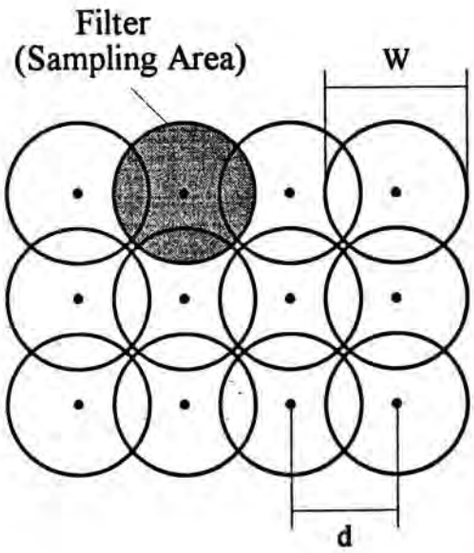


Figure 3.2 The enlarged filters

A simple sampling method is to take the ratio of the white subarea to the total sampling area to determine the gray value. Another approach is to weigh each point within the area as some function of the distance between the point and

the center of the sampling area, so that the black subareas near the centre of the pixel may contribute more darkness to the pixel than that near the edge of the pixel [Warnock 80]. The first approach is simple but does not give results as good as using weighted functions. Some experiments have been conducted using these two approaches [Moon *et al.* 92].

Weighted Functions

Various weighted functions are described in computer graphics literature [Kajiya and Ullner 81, Warnock 80]. The best results are obtained only when the characteristics of the output device and the eye are taken into consideration in the filtering function. Such a filtering function is suggested by Kajiya and Ullner. Unfortunately, as mentioned in their article, it is expensive, slow and relatively hard to implement [Kajiya and Ullner 81]. It requires much memory during processing. For example, to generate gray scale font of size 30x30, a matrix of almost a million entries is required. And, the calculation involves a lot of integration operations.

Warnock introduce the following weighted function families that worked surprisingly well in practice to generate gray scale English characters [Warnock 80].

[1] $$f(x,y) = S\cos^e\left(\frac{\pi\sqrt{x^2+y^2}}{W}\right) \qquad \text{for } \sqrt{x^2+y^2} \le \frac{W}{2}$$

$$f(x,y) = 0 \qquad \text{for } \sqrt{x^2+y^2} > \frac{W}{2}$$

[2] $$f(x,y) = S\left(\cos\left(\frac{\pi x}{W}\right)\cos\left(\frac{\pi y}{W}\right)\right)^e \qquad \text{for } -\frac{W}{2} \le x,y \le \frac{W}{2}$$

$$f(x,y) = 0 \qquad \text{for } x,y > \frac{W}{2} \text{ or } x,y < -\frac{W}{2}$$

[3] $$f(x,y) = S \times g(x) \times g(y)$$

where $$g(t) = -\frac{4}{W^2}t + \frac{2}{W} \qquad \text{for } t \ge 0$$

$$g(t) = \frac{4}{W^2}t + \frac{2}{W} \qquad \text{for } t < 0$$

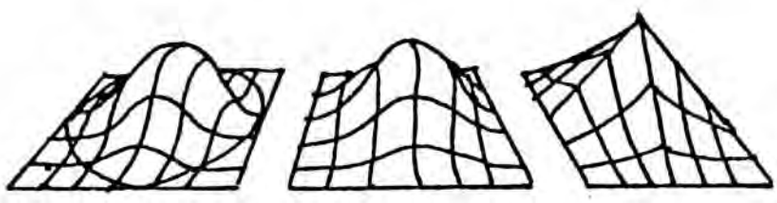
$$g(t) = 0 \qquad \text{for } t > \frac{W}{2} \text{ or } t < -\frac{W}{2}$$

The first one is called 2-dimensional Hamming filters while the third one is known as bi-linear filters. In these formulae, W, e and S are constant parameters whose functions are summarized as follows :

W -- indicates the width of the filter. This width corresponds to the diameter of the sampling area of a single pixel. It should be pointed out that with these families of filter functions, the wider filters are lower pass filters and tend to blur the resulting characters. The narrower filters tend to undersample, leaving unwanted gaps in the characters.

e -- controls the relative weighing of the filter near the center. Larger values of e imply a heavier weight near centre. Lower values of e make lower pass filters, and therefore tend to blur the resulting characters. Note that when e=0, the first two families become the simple averaging cases and large values of e cancel the effect of making W large.

S -- is a suitable value chosen so that the sum of all the elements in the array containing filter values comes to a desired maximum intensity I. For example, S is chosen to $\frac{(g-1)}{W^2}$ for plane filters, where g is the number of gray levels of the resulting gray scale font.



1 2 3

Figure 3.3 Weighted functions

Some experiments have been conducted using these families to generate gray scale Chinese characters. In addition, a filter without weight is used to compare the weighed filter effect.

[4] $f(x,y)=S$ for $-\frac{W}{2} \le x,y \le \frac{W}{2}$

$$f(x,y)=0 \qquad \text{for } x,y>\frac{W}{2} \text{ or } x,y<-\frac{W}{2}$$

Generation of Gray Scale Character

To produce a character of a given size, we need to generate a high precision master character bitmap and store it in an nxn array. It is obtained by performing rasterization of an outline font description of this character. Let mxm be the size of the resultant gray scale character and $d=\frac{n}{m}$ which is the increment size used between pixels of the master character. It is assumed that all the resultant character fonts are scaled down versions of the master characters, i.e. $n>m$. The filter width W and value of e are first chosen. Then, a WxW weighing array (filter), F, is constructed. To compute the weighed gray value for a given sampling point (x,y), the filter matrix F is centered at (x,y). Then we selectively sum up all values of the filter cells whose pixels at the corresponding positions in the master bitmap is ON (Figure 3.4). Repeating this sampling process for each pixel and then their gray values can be found. The cost of this operation is $O(m^2xW^2)$, which requires long processing time [Naiman and Fournier 87, Warnock 80].

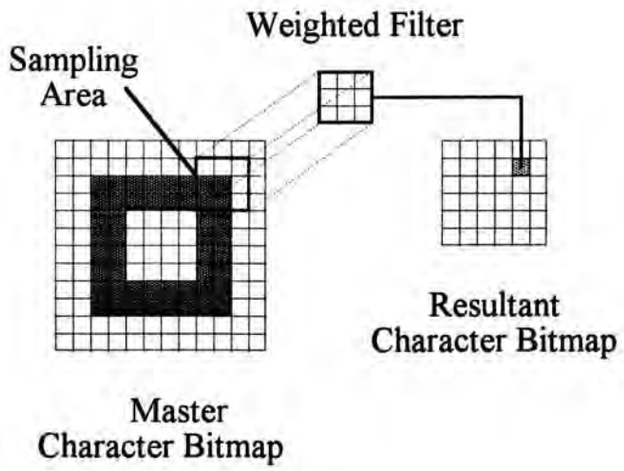


Figure 3.4 Filtering process

Results

Part of the results from the above experiments are summarized as follows [Moon *et al.* 92]:

[1] A comparison with a binary bitmap font, generated directly by rasterization from an outline font system, and the gray scale font produced has been made. For font size ranging from 20x20 to 100x100, both the quality and readability of the gray scale fonts are obviously better than the binary bitmap fonts. The character quality is very good for the font of size 48x48 or above. It is because the aliasing problem has been overcome.

[2] It has been found that the plane filter generates blurred character images when the filter width is $1.5d$ or above. Therefore, a weighing filter is necessary to produce good quality gray scale fonts.

[3] It has been found that values of W in the range $1.5d$ and $2d$ for large font sizes (32x32 pixels or above) give good results. For smaller font sizes (28x28 pixels or below) the range d to $1.5d$ of W is more suitable.

[4] Different ranges of gray level have also been tried. The fonts have been built with 4, 8, 16 and 64 gray level. 16 gray levels are found to be adequate but 64 gray levels give slightly better results.

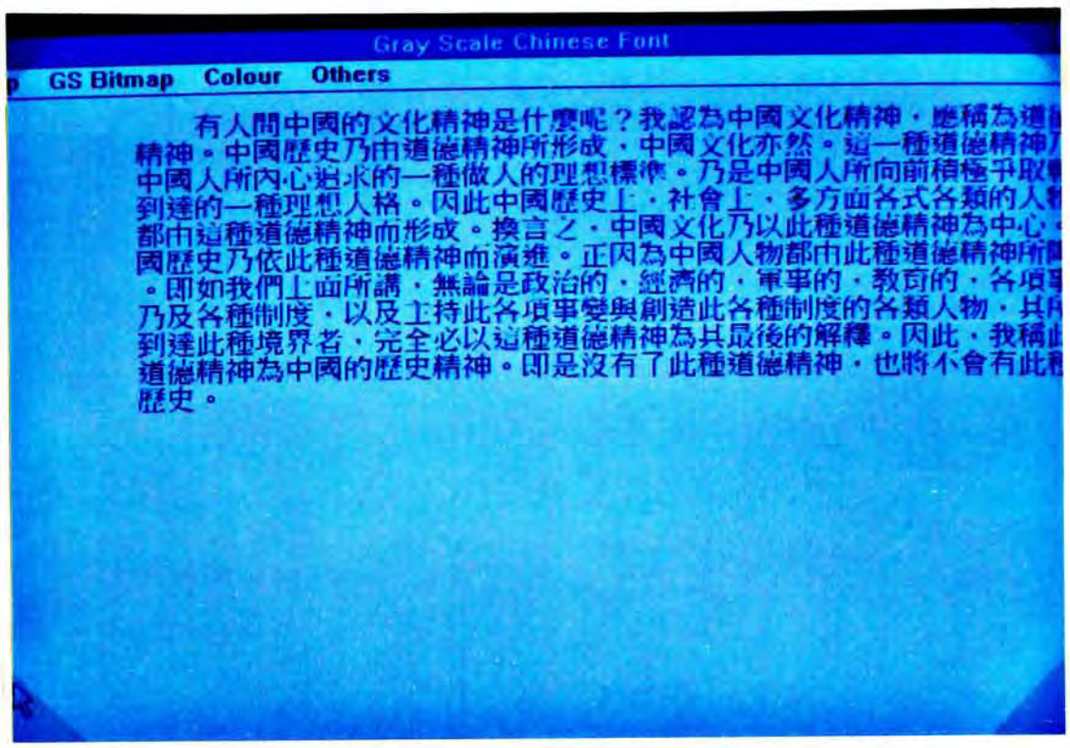
More Experiments

More studies on gray scale fonts were performed using an existing Chinese system [Moon and Poon 92]. Microsoft Chinese Window 3.0 (CWindow) is our choice because it has an outline font system which can provide various sized binary bitmaps as source data for the filtering process. Due to the long processing time of filtering method, this process is performed on a DEC Workstation[1]. It is done by transferring the binary bitmap fonts generated by CWindow to the DEC Workstation.

A gray scale font library containing 13,053 characters of size 24x24 is produced. It occupies about 3.6 MBytes of storage. Moreover, gray scale bitmap

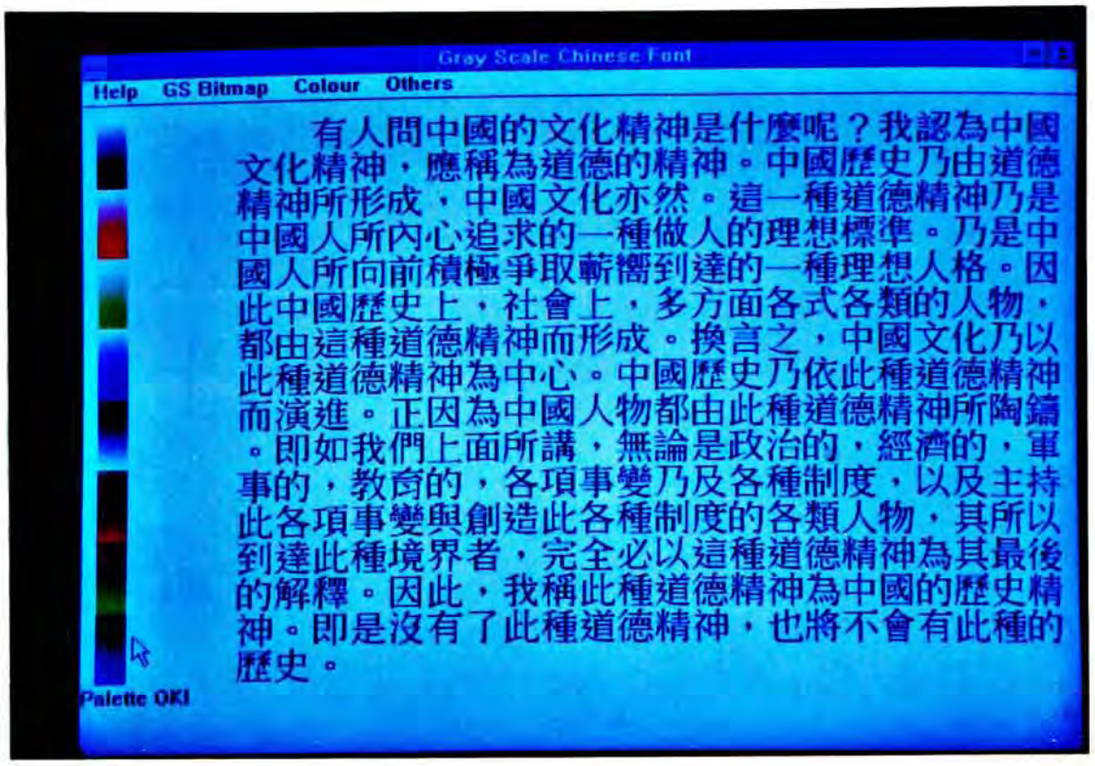
[1]DECstation 3100 with clock speed 16.67 MHz and processing power 16.2 MIPS

images of a passage in Chinese is constructed, in samples of different bitmap sizes of 16x16, 20x20, 24x24, 40x40, 48x48, 64x64 and 100x100. Results can be demonstrated through a CWindow program written for this purpose. Figure 3.5 shows the results of sizes 16x16 and 24x24.



(a) Size 16x16

Figure 3.5 Results of filtering process



(b) Size 24x24

Figure 3.5 Results of filtering process

Problems

Speed and Storage

The generation of gray scale fonts using filtering method involves two steps. Firstly, the outline description of the required character is converted into a master binary bitmap. The size of this master binary bitmap is two times or more larger than the size of the target gray scale font. The process of conversion introduces inefficiency especially when many curves exist in the outline description because their conversions involve large amount of computations. Then, a filtering method is performed using the binary bitmap as the source data. The filtering method also involves large amount of computations. The cost of this operation is $O(m^2xW^2)$, where m is the size of the resultant gray scale character and W is the width of the filter [Naiman and Fournier 87, Warnock 80]. Comparing with English character, the minimum size of character bitmap to represent a Chinese character is larger. It is known that at least 16x16 pixels are required to display

Chinese character and larger size is needed to express differences between typefaces. Thus, larger m is needed and the filtering method requires a long processing time [Moon and Poon 92]. Some experiments found that time needed to generate gray scale font using filtering method is triple as that required to scan-convert to generate a black-and-white bitmap image only [Abe *et al.* 89]. Thus, speed is the main problem that concerns us.

Storing the generated gray scale character images to create a gray scale bitmap font library may solve the speed problem. However, it is suitable for English but not Chinese because of the large size of the Chinese character sets. For storing gray scale fonts, more than one bit are needed to represent a pixel. To create a library of Chinese characters made up of 48x48 pixels, if we use 4 bits to represent 1 pixels (having 16 gray levels), the total storage needed to hold 13,053 characters is 14.3 mega bytes (Table 3.1). So it is impossible to create libraries of gray scale bitmap fonts in different sizes and styles.

Number of pixels of each character	Black and White (1 bit per pixel)	16 gray levels (4 bits per pixel)	64 gray levels (6 bits per pixel)
24 x 24	918 K Bytes	3.6 M Bytes	5.4 M Bytes
48 x 48	3.6 M Bytes	14.3 M Bytes	21.5 M Bytes
100 x 100	15.6 M Bytes	62.2 M Bytes	93.4 M Bytes

1 K Bytes = 1024 bytes
1 M Bytes = 1024 K Bytes

Table 3.1 Storage space of Chinese font library with 13,053 characters

Impression of Strokes

Some problems about quality are observed from experiment results. The first one is about the impression of parallel strokes. Even if two strokes having the same thickness in the master character are converted to rows with the same widths, they may not give the same impression to the reader. For example, 16 gray levels are applied on a character. Suppose a horizontal stroke is converted to two rows of pixels, where both rows contain pixels of gray value 7. If another stroke with the same thickness is also converted to two rows of pixels but one of the rows has all pixels of gray value 3 and the other has all pixels of gray value 11, then these two rows will not look similar (Figure 3.6). Unless the size of the generated gray scale character is large enough, say, 40x40, the difference may be

unnoticeable, otherwise these two strokes will give an impression of nonuniformity to human readers [Moon and Poon 92].

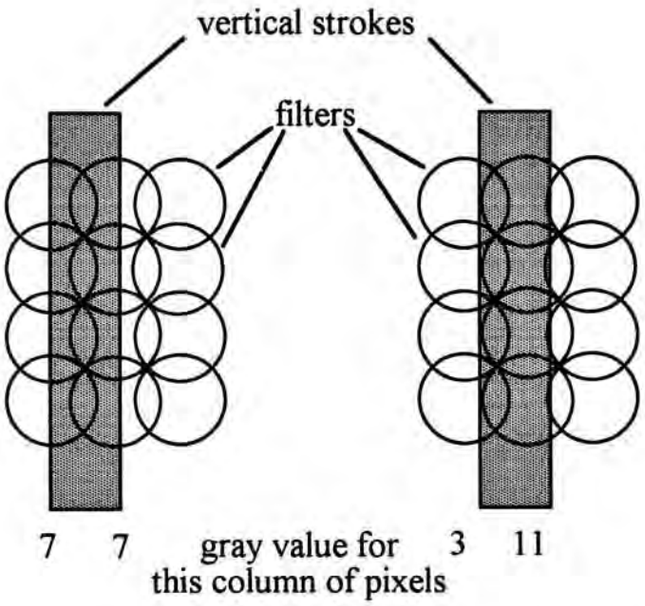
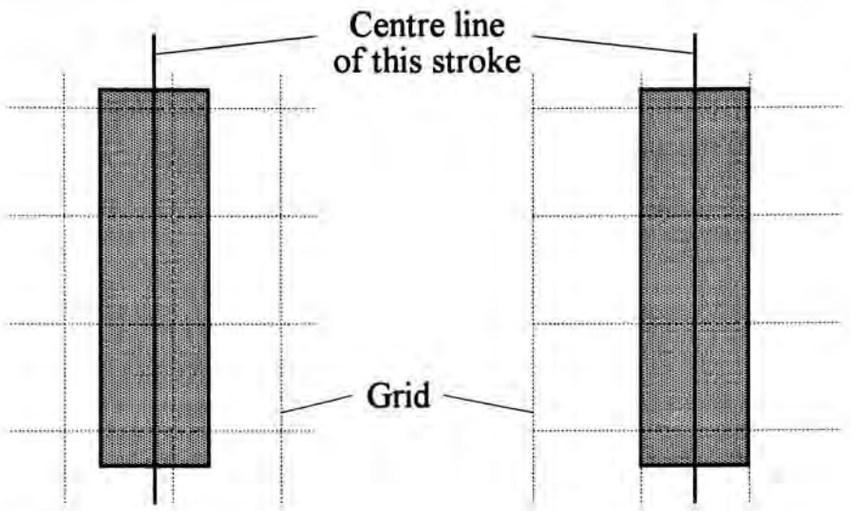


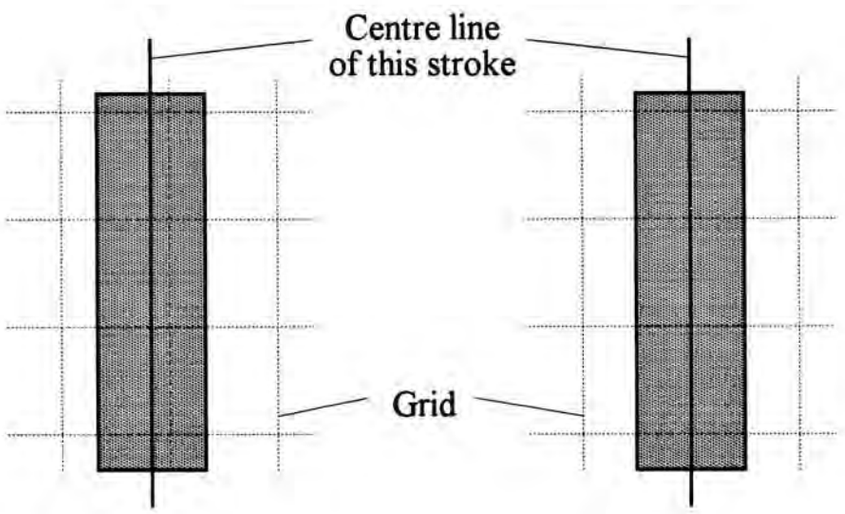
Figure 3.6 The formation of different impression of strokes

An algorithm is introduced to overcome the above problem [Abe *et al.* 91]. This algorithm is the gray scale extension of a binary bitmap based method called "centering operation" introduced by Hersch [Hersch 87], by which the strokes with same thickness in its master character are generated using the same number of rows of pixels in a gray scale bitmap.

The algorithm consists of two operations : centering and anti-centering [Abe *et al.* 91]. The centering operation forces the centre line of a horizontal or vertical stroke to pass through the middle between two grid lines (Figure 3.7a). The anti-centering operation, on the other hand, moves the centre line of a horizontal or vertical stroke to pass along a grid line (Figure 3.7b).



(a) The centering operation of a vertical stroke



(b) The anti-centering operation of a vertical stroke

Figure 3.7 Stroke displacement algorithm

The algorithm for displacing the horizontal and vertical strokes in the master character can be stated as follows:

[1] Extract horizontal and vertical strokes from the master character.

[2] Decide the centre line of those strokes.

[3] When the width of stroke is less than or equal to the size of a pixel, the centering operation is applied to the stroke. When the width of the stroke

is larger than the size of dot, the anti-centering operation is applied to the stroke.

[4] After the stroke displacement, the parts of the character which are directly attached to the displaced strokes are also displaced to preserve the local shape.

[5] The modified version of master character is convoluted with a filter to generate the gray scale font.

This method has some limitations. First, the extraction of horizontal and vertical strokes from the master character is not a simple job. Second, the algorithm is more suitable for fonts having mainly horizontal and vertical strokes than those having curved strokes [Abe *et al.* 91].

Thin strokes in the small-size character

Another problem occurs with the thin strokes. A problem with gray-scaled fonts of small size happens with the representation of thin strokes. Small gray scale characters often look fuzzy or ambiguous because the characters are almost made up of many gray dots only, while black and white pixels seldom appear as the thickness of most strokes is less than the size of pixel. This phenomenon takes place when the font size is small enough, say, 16x16. Even in medium-sized characters, say, 24x24, similar problems occur with some thin strokes.

New Approach to Generate Gray Scale Font

Instead of generating the gray scale font by filtering method, we can generate gray scale font directly from outline font data (Figure 3.8).

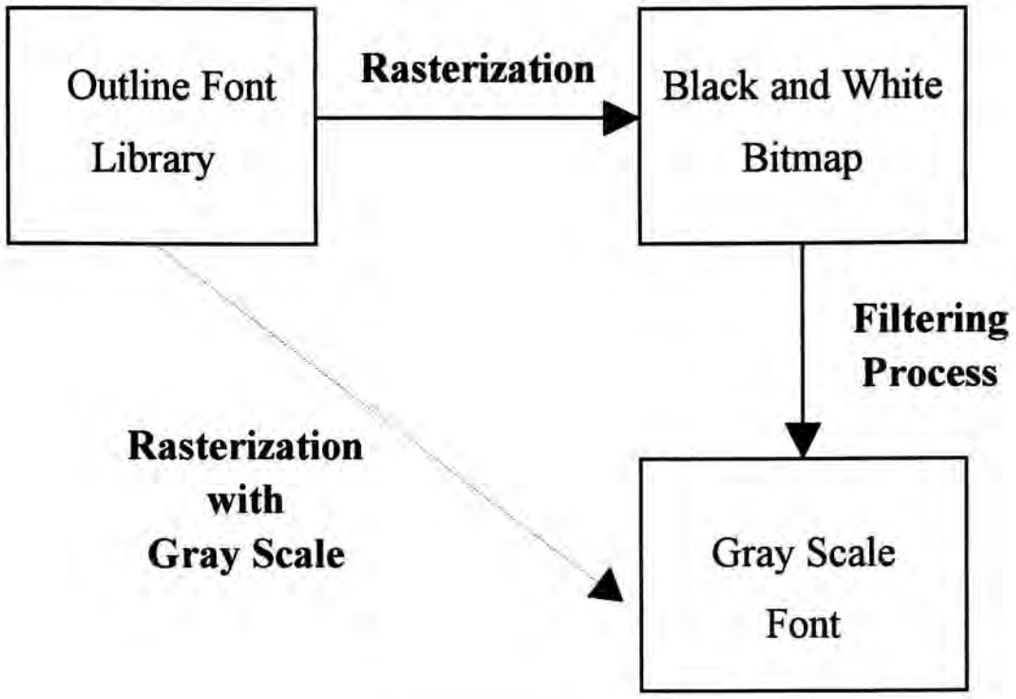


Figure 3.8 A new approach to generate gray scale font

When generating gray scale fonts, long processing time is needed during the filtering process. For example, to generate 20 gray scale characters of size 48x48 using a 200x200 master character bitmap, 33 seconds is needed. However the rasterization process consumes much less time. Comparing the time required for the two generation paths shown in Figure 3.8, although the conversion by rasterization from outline fonts to gray scale fonts should take longer time than that to binary bitmap fonts, the former saves the filtering step which is a dominant factor to the speed problem mentioned above. Therefore if we produce gray scale fonts directly from the outline fonts without going through the intermediate step of generating the binary bitmap fonts, the speed problem is probably reduced. If the speed problem is solved, there is no need to generate a gray scale font library for screen display but the gray scale character can be displayed by ad hoc generation.

Chapter 4

Rasterization Algorithms

Outline Font

In an outline font system, the shape of a character is described by its outline. An outline is composed of straight lines and curves, such as arcs, natural splines, Bezier splines, etc. Font library stores the character shapes in the form of mathematical parameters. For example, a straight line is stored by the coordinates of its two end points. Thus, the shape of character can be preserved when we scale it to other sizes [Bigelow 85]. Sometimes, outline fonts are called "resolution-independent fonts" [Andler 89].

Since the font is stored in the form of outline description only, a hardware or software, called rasterizer, is need when the font is to be output to some raster-scan devices, such as monitor and printer. The rasterizer takes an outline description of a character and produces a bitmap image for that character. This process is called "rasterization" [Deach 92, Hersch 89]. The process of rasterization introduces inefficiency especially when many arcs and curves are consisted in the descriptions, since their rasterization involve large amounts of computation [Moon and Cheang 91]. This chapter will introduce some implementation techniques to improve the speed of rasterization.

Different manufacturers uses different proprietary formats for storing their font outlines. Some manufacturers have two or three formats (Table 4.1). No apparent quality difference has been found between the various formats. All have sufficient resolution for storing high quality font outlines [Deach 92].

File Format	Curve Form
Adobe Type 1	Bezier
TrueType	Quadratic-spline
Bitstream:	
Definitive	Arc
Fontware	Arc
Speedo	Bezier
Intellifont	Arc
Nimbus-Q	Bezier

Table 4.1 The curve form of the common font file formats

TrueType Font

TrueType Font is a new format to describe an outline font. It is introduced by Apple Computer, Inc. and Microsoft Corporation [Microsoft 92]. In our experiments, we use an outline font library CFSung as sample data, which is in the format of TrueType Font. In a TrueType font, glyph shapes are described by their outlines. A glyph outline consists of a series of contours, where a contour is a closed curve. A simple glyph may have only one contour, while more complex glyphs can have two or more contours. Figure 4.1 shows glyphs with one, two and three contours.

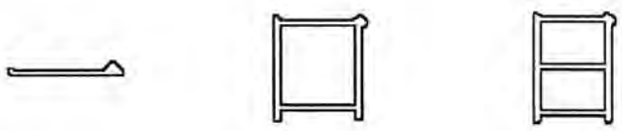


Figure 4.1 Glyphs with one, two and three contours respectively

Contours can be composed of straight lines and/or second order Bezier curves [Microsoft 92]. They are stored by their control points. A control point will be described by its coordinates and a flag which determines whether this point

lies on curve or off curve. A character outline and its control points are shown in Figure 4.2.

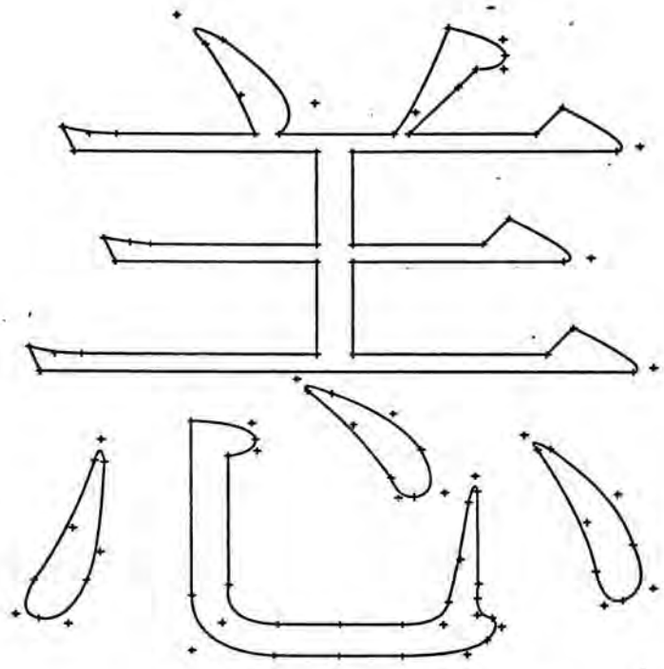


Figure 4.2 A TrueType outline character and its control points
Source : CFSung in Microsoft Chinese Windows 3.0

A straight line is defined by two consecutive "on" curve points (A_x, A_y) and (B_x, B_y). The points (X,Y) on the line segment between (A_x, A_y) and (B_x, B_y) is expressed as

$$\begin{cases} X = A_x + B_x t \\ Y = A_y + B_y t \end{cases} \qquad \text{where } 0 \le t \le 1$$

A curve is defined by three control points (A_x, A_y), (B_x, B_y) and (C_x, C_y). They describe a second order Bezier curve (i.e. quadratic Bezier curve). Two end points are "on" curve points and the middle point is an "off" curve point. The point (X,Y) on the curve is expressed as

$$\begin{cases} X = A_x(1-t)^2 + 2B_x(1-t)t + C_x t^2 \\ Y = A_y(1-t)^2 + 2B_y(1-t)t + C_y t^2 \end{cases} \qquad \text{where } 0 \le t \le 1$$

In order to get a fixed size bitmap from the outline of a character, firstly we have to scale the outline to the size wanted. For example, to obtain a bitmap with size 64x64 from the outline with size 512x512, we have to scale down the

outline by multiplying it with the factor 0.125 (i.e. 64/512). Then, rasterization is performed on the resulting outline.

Scan Conversion

Basic Outline-to-Bitmap Conversion

The basic idea of scan-converting a character outline to generate a bitmap font is illustrated as follow (Figure 4.3):

```
for scan-line = 0 to bitmap size-1 do
        calculate the intersections X_i of this scan-line to the font outline
        for each pair of X_i, X_{i+1} do
                determine which pixels should be turned ON
        end
end
```

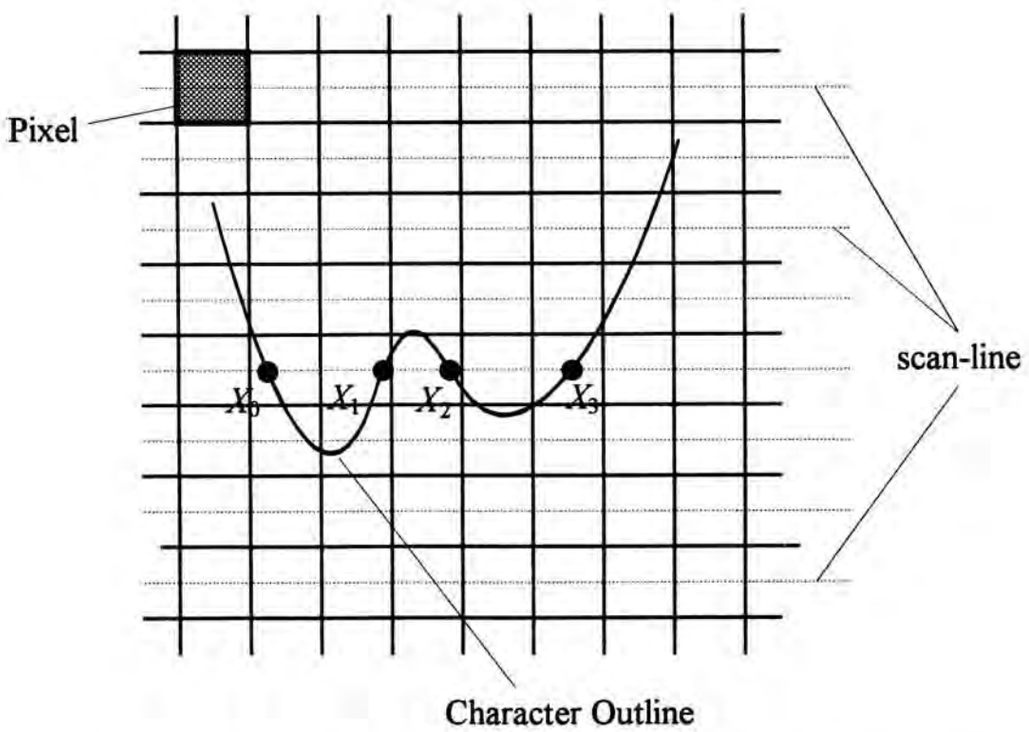


Figure 4.3 Basic idea of scan conversion to generate a bitmap character

Scan-converting Polygon

Scan-converting of polygons has been used in raster device coordinate system for a long time. The basic idea is to create and fill the polygon one scan line at a time [Foley *et al.* 90]. The process consists of three steps.

1. Find the intersections of the scan line with all edges of the polygon.
2. Sort the intersection by increasing x-coordinate, and
3. Turn ON all the pixels between pairs of intersections.

For example, in Figure 4.4 the sorted list of x-coordinate is (2, 4, 10, 13). We therefore turn ON the pixels in the intervals from 2 to 4 and from 10 to 13.

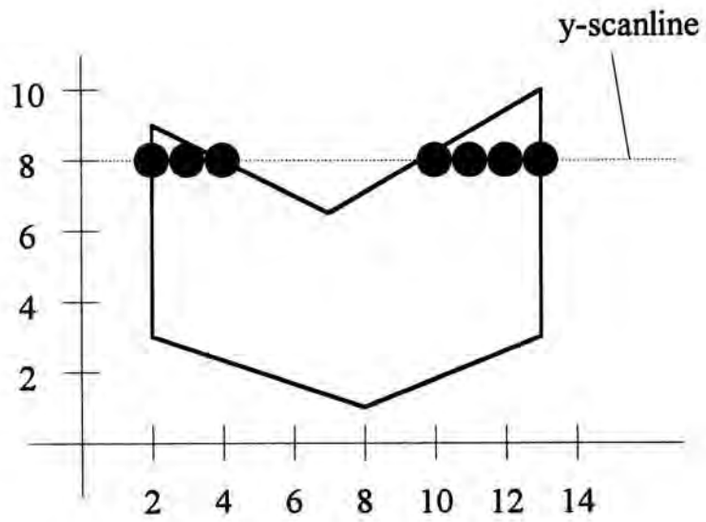
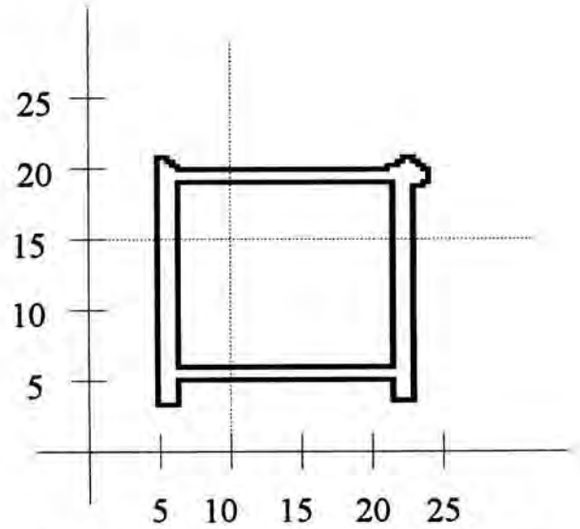


Figure 4.4 Polygon and y-scanline 8

Rasterization of a character

The scan line method discussed above is not suitable for rasterizing a Chinese character. To rasterize a Chinese character which is composed of many closed curves, integer computation is not accurate enough. This is especially true for small size character bitmaps. Therefore, real number is used in our rasterizer for higher precision. Moreover, scan conversion has to be performed in both horizontal and vertical direction instead of a single direction in the traditional method. An example is illustrated in Figure 4.5.



Scanline X=10 have intersecting points 5.14, 5.96, 19.25 and 20.05

Scanline Y=15 have intersecting points 4.87, 6.34, 21.53 and 22.97

Figure 4.5 Character and scanline in both direction

Intersecting Points and Ranges

Straight Lines

A straight line in the font outline is represented by its two end points (A_x, A_y) and (B_x, B_y). The intersection of this straight line and the scanline $Y = S_y$ can be represented by the coordinate (X, Y) where

$$\begin{cases} X = A_x + (B_x - A_x) \times \dfrac{(S_y - A_y)}{(B_y - A_y)} \\ Y = S_y \end{cases}$$

The intersection of this straight line and the scanline $X = S_x$ can be found by similar method.

The ranges of a straight line are (X_{start}, X_{end}) in x-direction and (Y_{start}, Y_{end}) in y-direction, where $X_{start} = \min(A_x, B_x)$

$$X_{end} = \max(A_x, B_x)$$
$$Y_{start} = \min(A_y, B_y)$$
$$Y_{end} = \max(A_y, B_y),$$

Therefore, $Y_{start} \le S_y \le Y_{end}$ and $X_{start} \le S_x \le X_{end}$.

Quadratic Bezier Curves

A quadratic Bezier curve in the font outline is represented by its three control points (A_x, A_y), (B_x, B_y) and (C_x, C_y). The points on this curve can be found by evaluating

$$\begin{cases} X = A_x(1-t)^2 + 2B_x(1-t)t + C_x t^2 \\ Y = A_y(1-t)^2 + 2B_y(1-t)t + C_y t^2 \end{cases} \quad \text{where } 0 \le t \le 1$$

The intersection of this quadratic Bezier curve with the scanline $Y = S_y$ is calculated by solving

$$\begin{cases} Y = A_y(1-t)^2 + 2B_y(1-t)t + C_y t^2 \\ Y = S_y \end{cases}$$

$$\Rightarrow A_y(1-2t+t^2) + 2B_y(1-t)t + C_y t^2 = S_y$$

$$\Rightarrow (A_y + C_y - 2B_y)t^2 + 2(B_y - A_y)t + A_y - S_y = 0$$

$$\Rightarrow t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{where } a = (A_y + C_y - 2B_y)$$

$$b = 2(B_y - A_y)$$

$$c = A_y - S_y$$

$$\text{if } 0 \le t \le 1,\ X = A_x(1-t)^2 + 2B_x(1-t)t + + C_x t^2$$

The point (X, Y) is the intersecting point of this Bezier curve and the scan-line $Y = S_y$. And, the intersection of this straight line and the scanline $X = S_x$ can be found by similar method.

The ranges of a Bezier curve are (X_{start}, X_{end}) in x-direction and (Y_{start}, Y_{end}) in y-direction. Finding of (X_{start}, X_{end}) and (Y_{start}, Y_{end}) is same as calculation of minimum and maximum values of a quadratic equation. And, they can be obtained by differentiation. Comparing the calculations of range covered by a Bezier curve and a straight line, the former is more complicated. To avoid complicated calculations, we can approximate the minimum and maximum values by

$$\begin{aligned} X_{start} &= \min(A_x, B_x, C_x) \\ X_{end} &= \max(A_x, B_x, C_x) \\ Y_{start} &= \min(A_y, B_y, C_y) \\ Y_{end} &= \max(A_y, B_y, C_y) \end{aligned}$$

But, this approximation will introduce a waste of effort to solve the intersecting points of the curve and the scan-line between the actual range and the approximated one. For example, in Figure 4.6(a), the actual maximum is different from the approximated one. It will waste the work done to find out the intersections between these two maximums because no intersecting points can be found along these scan-lines.

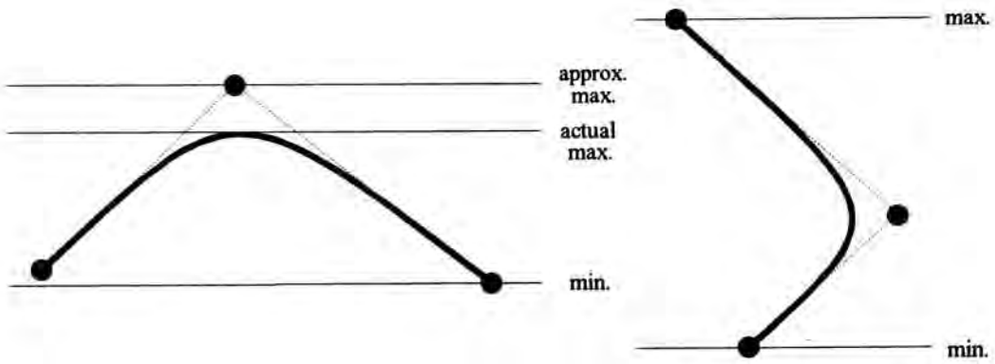


Figure 4.6 Range of y-coordinate of a Bezier curve

Implementation Techniques

Two techniques are used to speed up the rasterization process.

Approximation of quadratic Bezier curve by straight lines

It is very time-consuming to find the intersecting points of a quadratic Bezier curve and a scan-line because a quadratic equation has to be solved. This involves such time consuming calculations as finding the square and square root of a floating point number. Of course, the range covered by the Bezier curve can also be obtained by differentiation and approximation. Yet, both methods have their weaknesses. To prevent these problems caused by the Bezier curve calculation, a technique is adopted. We can use straight lines to approximate a quadratic Bezier curve. A straight line joined by the first and third control points is used if the tolerance is acceptable (Figure 4.7). If the error is large, the quadratic Bezier curve will be divided into two quadratic Bezier curves (Figure

4.8). Then the error of each of the partitioned Bezier curves is calculated again. The partition process repeats until the error is small enough.

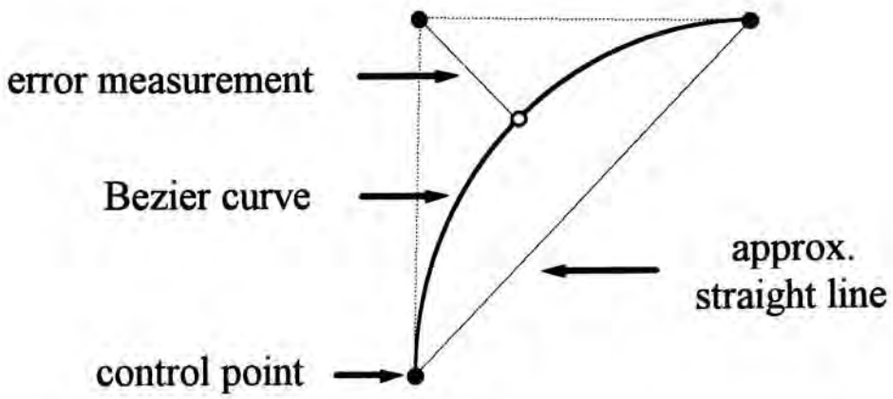


Figure 4.7 Approximation of quadratic Bezier curve

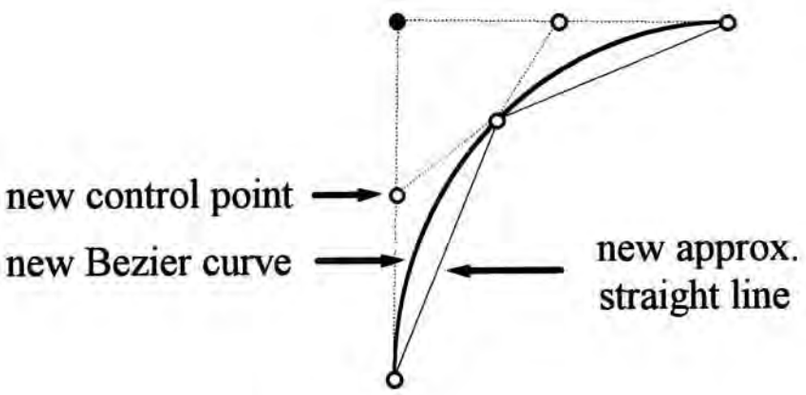


Figure 4.8 Partition of quadratic Bezier curve

For convenience, the error is approximated by the distance of the second control point and the mid-point of the quadratic Bezier curve (Figure 4.7). Since the calculations of the distance involve finding the square and square root of a value, the following approximation is used:

$$
\begin{aligned}
\text{Error} &\approx D \\
D &= \text{distance between } (x_1, y_1) \text{ and } (x_2, y_2) \\
D &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\
D^2 &= (x_1 - x_2)^2 + (y_1 - y_2)^2
\end{aligned}
$$

Require $D < M$

$\Rightarrow D^2 < M^2$

Let $M^2 = 2N$, then

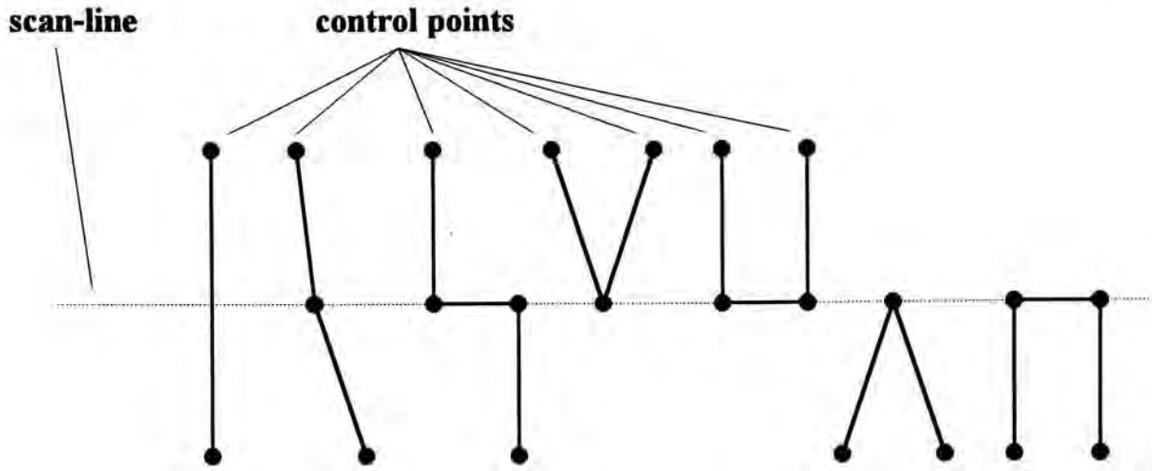
$(x_1 - x_2)^2 < N$ and $(y_1 - y_2)^2 < N$

$\Rightarrow |x_1 - x_2| < N'$ and $|y_1 - y_2| < N'$

Therefore we can control the error by setting N' to a small value. In our rasterizer, we choose the value of N' to be 0.5. This value can be set smaller to achieve higher precision, but the number of partition steps will increase, and thus longer time will be consumed.

Simplification of the Filling Process

We encounter another problem when attempting to fill the outline of a character. Figure 4.9 shows the possible cases that the outline of a character may cross a scan line. When we include the two end-points of each line into the intersecting point list, case C will be counted as having four intersecting points. Then, the pixels between the two points lying on scan line will be turned OFF mistakenly. However, if we do not include the two end-points into the intersecting point list, there will be no intersecting point in case B. As a result, the filling process will be wrong again. It causes an ambiguity in filling an outline of a character.



case	A	B	C	D	E	F	G
include end-points	1	2	4	2	4	2	4
do not include end-points	1	0	2	0	2	0	2

Figure 4.9 No. of intersecting points in different cases of scan-line crossing

Possible solutions are even-odd filling and nonzero-winding-number filling methods. Nevertheless, these two methods require much information such as whether the lines are moving upward or downward [Gonczarow 89]. To avoid finding and storing the moving direction, a technique is used to adjust the straight lines to make the filling process easier and faster. The technique is used for simplifying the process of filling the outline and to remove the ambiguity. The technique is described below:

> *For each straight line, if the lower point lies exactly on a grid line, this point will be displaced to a position slightly higher than the grid line.*

The ambiguity is removed using this method. The result of this technique is summarized as follows:

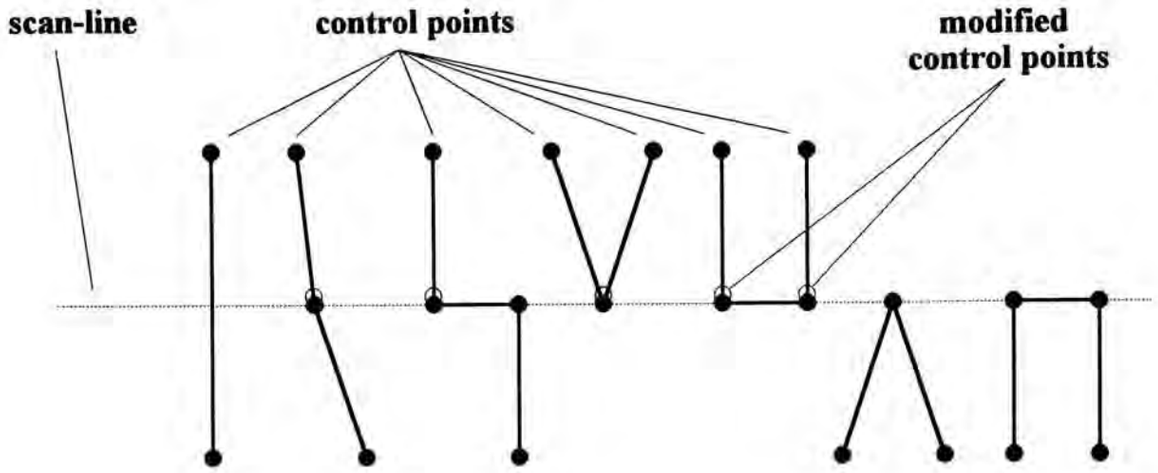
[i] case D in Figure 4.10 will be counted as having zero intersecting point.

[ii] case A and B will be counted as having a single intersecting point.

[iii] case E and F will be counted as having two intersecting points.

[iv] case C will be counted as having three intersecting points.

[v] case G will be counted as having four intersecting points.



Case	A	B	C	D	E	F	G
No. of Intersecting Points	1	1	3	0	2	2	4

Figure 4.10 No. of intersecting points in different cases of scan-line crossing (using modified control points)

When all intersecting points have been found, we need to determine which pixels should be turned ON. Obviously, the pixels between two intersecting points should be turned ON. For example, Pixel 2 in Figure 4.11 should be turned ON. But, for the pixels on which an intersecting point lies, decision is not easy to make. For examples, Pixels 1, 3, 5 and 6 are in this case. Some methods are described below to handle these difficulties.

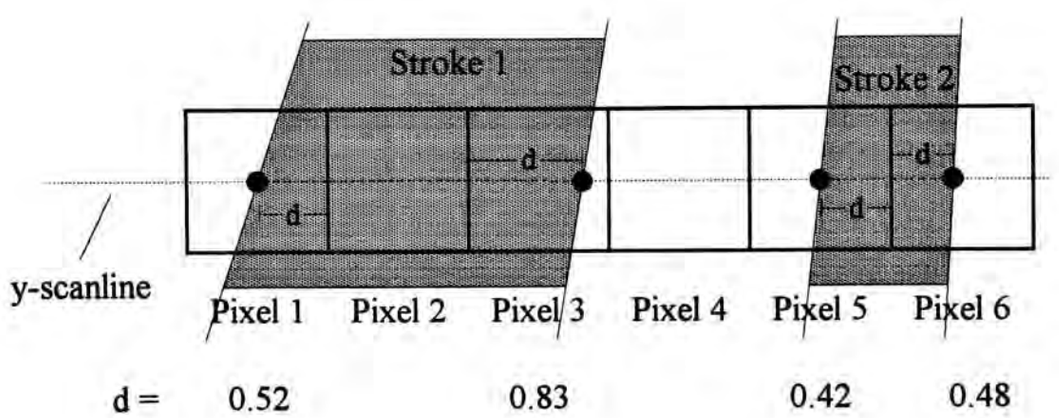


Figure 4.11 Two strokes across a y-scanline

Method 1: Rounding

If the distance of the intersecting point and the edge of this pixel (which lies in the stroke) is larger than or equal to half of a pixel's width, turn this pixel ON.

Result : In Figure 4.11, Pixels 1, 2 and 3 will be turned ON for stroke 1 and no pixel will be turned ON for stroke 2.

Method 2 : Truncation

If the intersecting point of the left edge (bottom edge for x-scanline) and the y-scanline lies on this pixel, turn this pixel ON. If the intersecting point of the right edge (top edge for x-scanline) and the y-scanline lies on this pixel, turn this pixel OFF.

Result : In Figure 4.11, Pixels 1 and 2 will be turned ON for stroke 1 and Pixel 5 will be turned ON for stroke 2.

Method 3 : Truncation plus one

If the intersecting point of the left edge (bottom edge for x-scanline) and the y-scanline lies on this pixel, turn this pixel OFF. If the intersecting point of the right edge (top edge for x-scanline) and the y-scanline lies on this pixel, turn this pixel ON.

Result : In Figure 4.11, Pixels 2 and 3 will be turned ON for stroke 1 and Pixel 6 will be turned ON for stroke 2.

All the three methods described above have pros and cons. Like method 1, it is simple but error is large. For the example in Figure 4.11, the width of stroke 1 on this scan-line is 2.35 pixel, but 3 pixels will be turned ON. The width of stroke 2 is 0.9 pixel, but no pixel will be turned ON. Discontinuity of this stroke will occur, and this phenomenon is called dropout. Methods 2 and 3 are similar. 2 pixels are turned on in stroke 1 and 1 pixel in stroke 2. The results are better than method 1 for the examples in Figure 4.11. Some rasterizers use Method 2 to prevent dropout. It forces the left-most (bottom-most) pixel to be ON to prevent dropouts which often occur on thin strokes in small characters.

The Rasterization Algorithm

The steps to convert an outline font into a bitmap is, thus, summarized as follows:

[1] Read the data of the required character from an outline font library.

[2] Transform the coordinates of the control points into the scale of the required size.

[3] Approximate all Bezier curves of the outline by straight lines.

[4] Now the outline of character has straight lines only. Apply the second technique to all straight lines to remove the ambiguity of filling process and find all intersecting points of straight lines and Y-scanlines.

[5] For each scanline,

- [5 a] Set all pixels OFF on this scanline.
- [5 b] Sort the intersecting points on this scanline.
- [5 c] Pair up the consecutive points for this scanline, turn ON all pixels between these two points along the scanline.

[6] Repeat steps 4 and 5 with the X-scanlines.

Chapter 5

Direct Rasterization with Gray Scale

Some problems were uncovered when applying the filtering method to generate gray scale Chinese font. A new approach, building a gray scale rasterizer which can generate gray scale Chinese characters directly from their outline counterparts is investigated (Figure 5.1).

Figure 5.1 Generation of gray scale font using gray scale rasterizer

Rasterization with Gray Scale

To generate gray scale character bitmaps from their outline descriptions, the rasterization algorithm described in Chapter 4 is modified. Instead of determination of whether 'ON' or 'OFF' of each pixel, the gray value of each pixel should be found. Notice that gray value is used to determine the darkness of a pixel. For example, to display black text on white background colour, the maximum gray value means it is a black pixel while the minimum value means it is a white one. Pixels with gray values between the maximum and the minimum appear in various gray colours.

Similar to the rasterization algorithm for generating black-and-white character bitmap, the intersecting points of the character outline and scanlines should be found in order to determine the gray value of each pixel. Obviously, the colour of pixels between two intersecting points should be set to black, that is the gray value of these pixels should be set to be the maximum gray value (i.e. No. of gray level - 1). A gray pixel will be generated when part of this pixel covered by a character stroke. For example, Pixel 3 in Figure 5.2 should be set to the maximum gray values while the gray values of Pixel 2 and Pixel 4 will lie between maximum and minimum gray values.

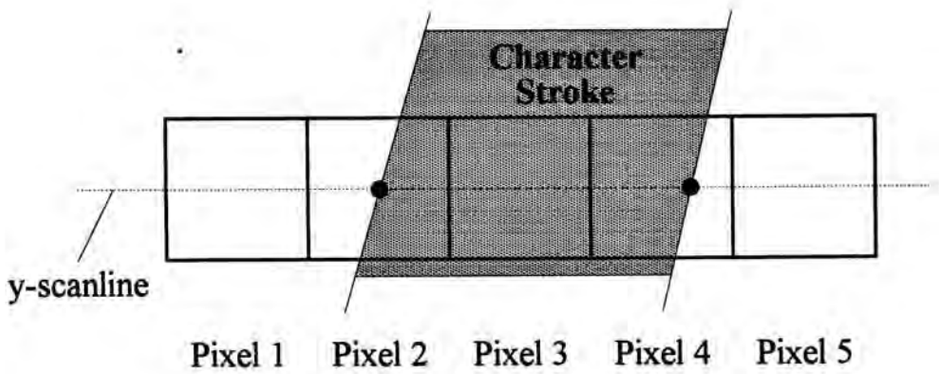
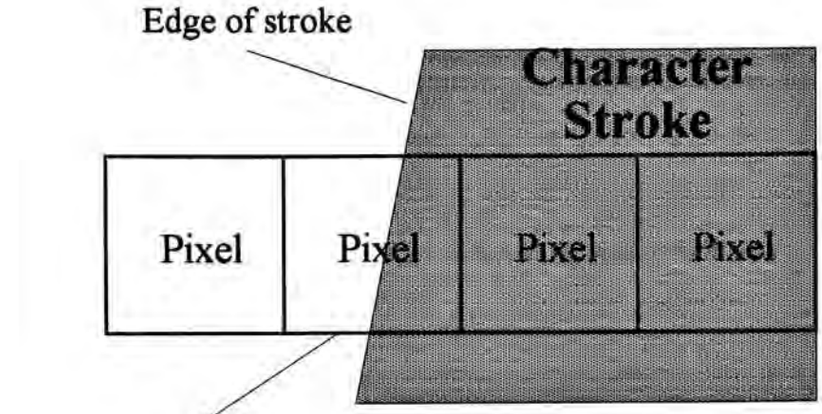


Figure 5.2 A character stroke cross y-scanline

Concerning the pixels along the edge of a character stoke, which are called boundary-pixels, the determination of gray value is non-trivial. A brief idea is proposed and illustrated in Figure 5.3. We calculate the ratio of area covered by the stroke to the area of the whole pixel. Using this ratio, the gray value of this pixel can be determined. For example, suppose the ratio equals 0.3. If there are 16 gray levels, the gray value of this pixel is 0.3 x 15, which is 5 after rounding.



$$\text{Gray value of this pixel} = \frac{\text{Area covered by this stroke}}{\text{Area of whole pixel}} \times (\text{Number of gray level} - 1)$$

Figure 5.3 Determination of the gray value of a pixel

Intuitively, it seems that the ratio of the area covered by a stroke in a boundary-pixel may be simply approximated by the distance between the edge of the pixel and the intersecting point, i.e. d in Figure 5.4. d is used because it is easily computable in the rasterization process.

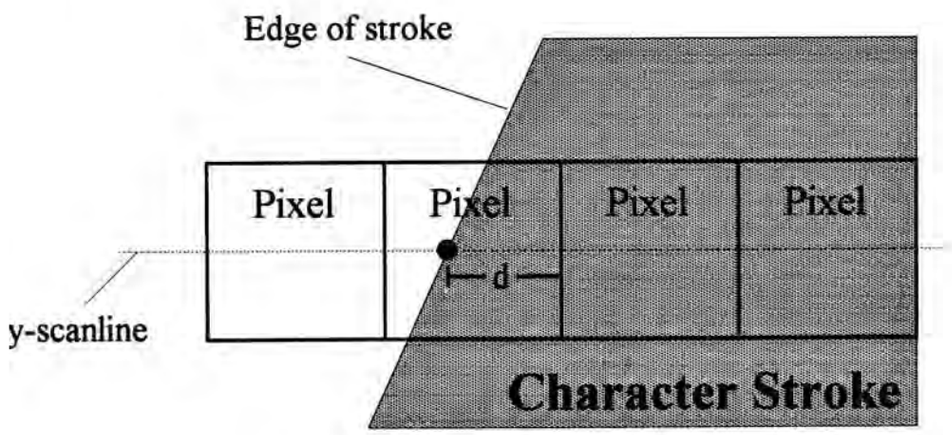
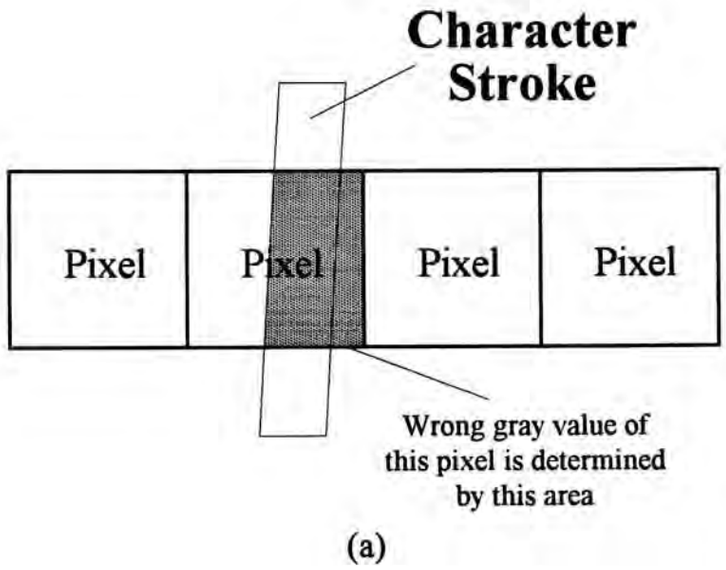


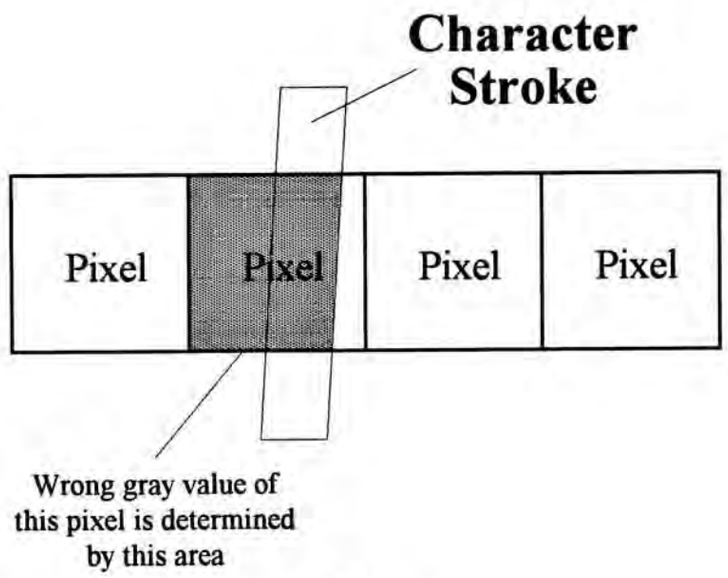
Figure 5.4 Approximate the ratio of area covered on a boundary-pixel

While analysing the character bitmaps generated by the above method, inappropriate determination of the gray values of some pixels are found. The first one occurs with the thin strokes, whose left and right edges lie on same pixel. Using the above method to calculate the gray value will give wrong result in either case as demonstrated in Figure 5.5(a) or (b). These errors occur because we

make a wrong assumption that the stroke covers the whole of right side of this pixel.



(a)



(b)

Figure 5.5 Wrong determination of the gray value of a pixel

Another case introduces wrong determination of gray value on the less inclined edges of strokes (Figure 5.6). If we use the above simple method, only Pixel 3 will be set to a gray pixel. Pixels 1 and 2 are considered outside of the character stroke. Thus, they are assigned the minimum gray value (i.e. the colour of these pixels are the same as background colour). Pixels 4, 5 and 6 will set to maximum gray value. Actually, all pixels 2 to 5 are covered by the character

stroke partially. We should expect different gray colours to be given to these pixels in order to achieve the purpose of anti-aliasing. But only one gray pixel is generated between the white background and black character stroke. As a result, staircasing still occurs on the edge of this stroke.

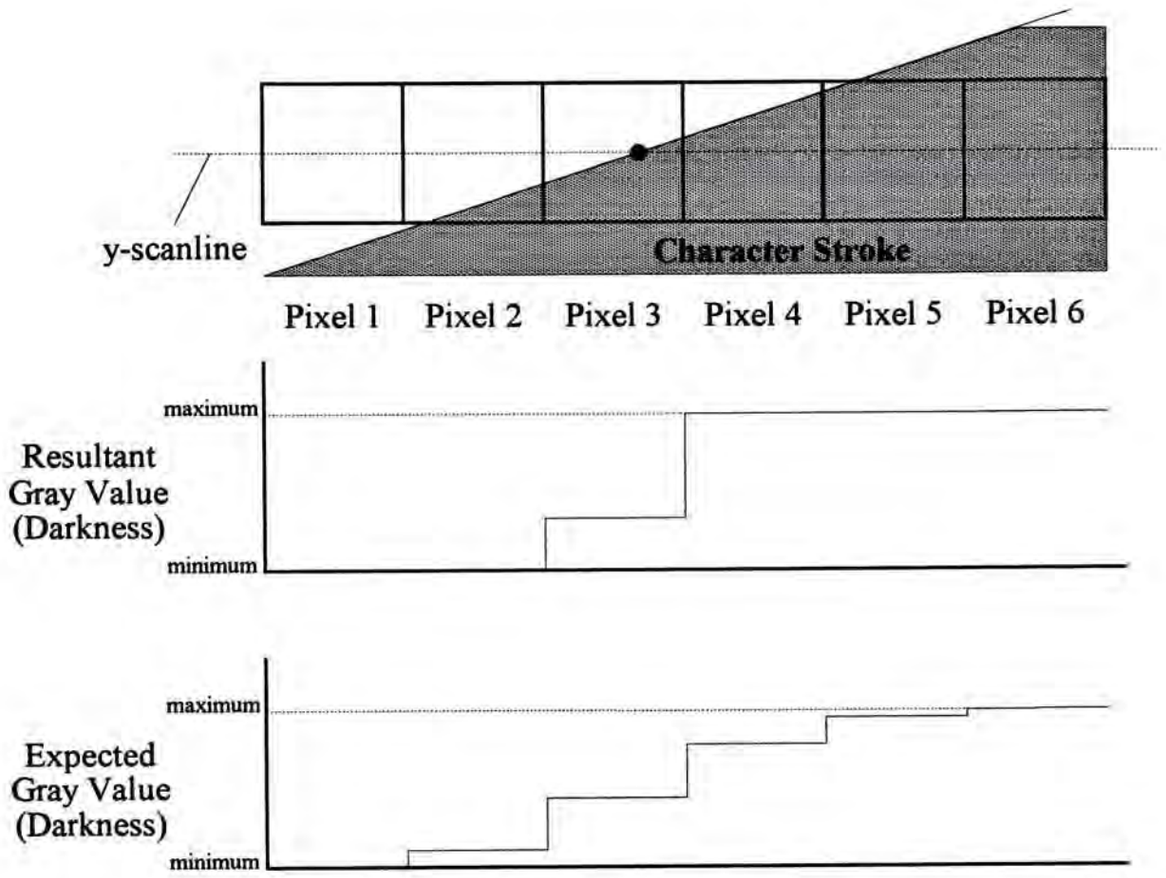


Figure 5.6 Wrong determination of gray value of pixels on less inclined stroke

Determination of Gray Value of Boundary-pixel

There are many cases how a stroke slants. Its edge may cover different portions of different pixels. For the sake of understandability, we only consider the left edge of a stroke. Here we assume a pixel is square in shape with sides of 1 unit length. Consider pixels A, B and C in Figure 5.7,

where the bottom edge of Pixel A intersects with the edge of the stroke;
the y-scanline on Pixel B intersects with the edge of the stroke;
the top edge of Pixel C intersects with the edge of the stroke.

Then, these cases are grouped into three categories and they will be described below.

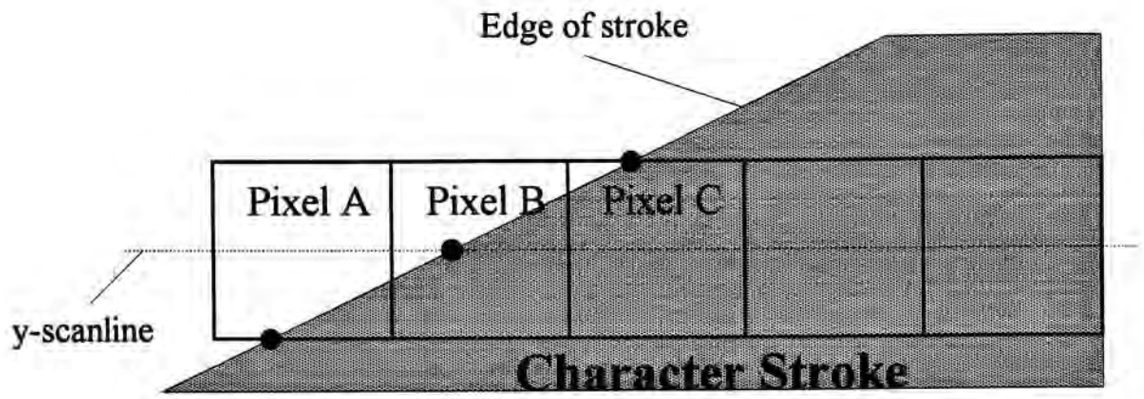


Figure 5.7 Definition of Pixels A, B and C

Category 1 : Pixels A, B and C are the same pixel

The edge of the character stroke has intersecting points with a scan line, and the top edge as well as the bottom edge of the pixel. To avoid the problem mentioned above, we use d' to calculate the gray value instead of d (Figure 5.8).

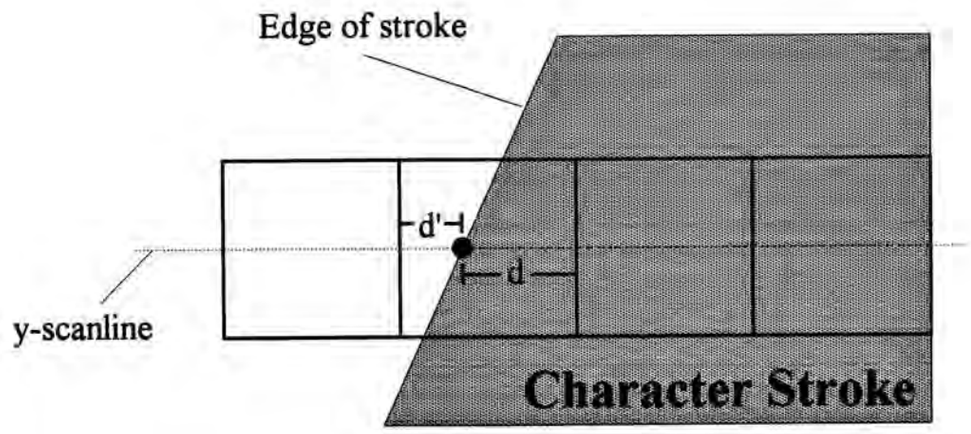


Figure 5.8 Pixels A, B and C are the same pixel

The algorithm to determine the gray value of pixels in category 1 as follow:

(All pixel is set to minimum gray value, i.e. 0, before rasterization)
For each scanline

If any part of a pixel is covered by a stroke, set the gray value of a pixel to maximum gray value(i.e. 15 if 16 gray level is used). In Figure 5.9, the gray value of the pixels 1, 2, 3, 5 and 6 are set to maximum gray value. Find d' such that d' is the distance between the edge of this pixel and the edge of character stroke intersecting the scanline (Figure 5.8). The new gray value of this pixel is set to be the old gray value of this pixel minus the product of d' and the maximum gray value.

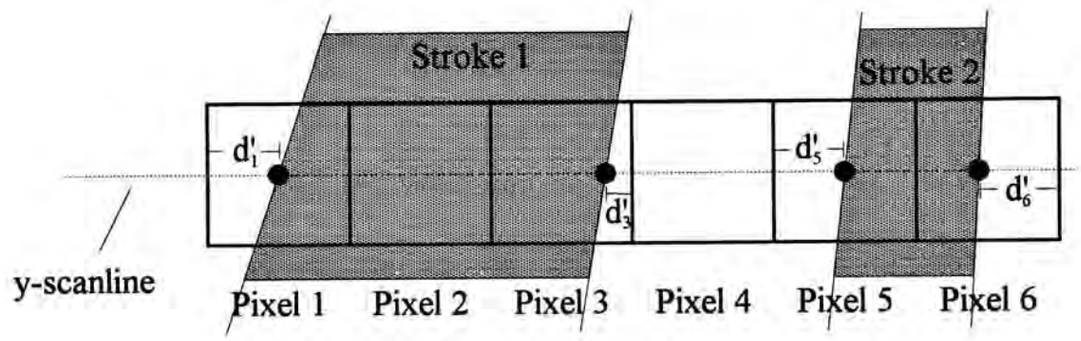


Figure 5.9 Two strokes across a scanline

Using this method, the problem mentioned above is solved. It is because the left side area that is not covered by this stroke will be eliminated when the left edge is processed, and the right side area will be eliminated in a similar way. Thus, the correct gray value can be found (Figure 5.10).

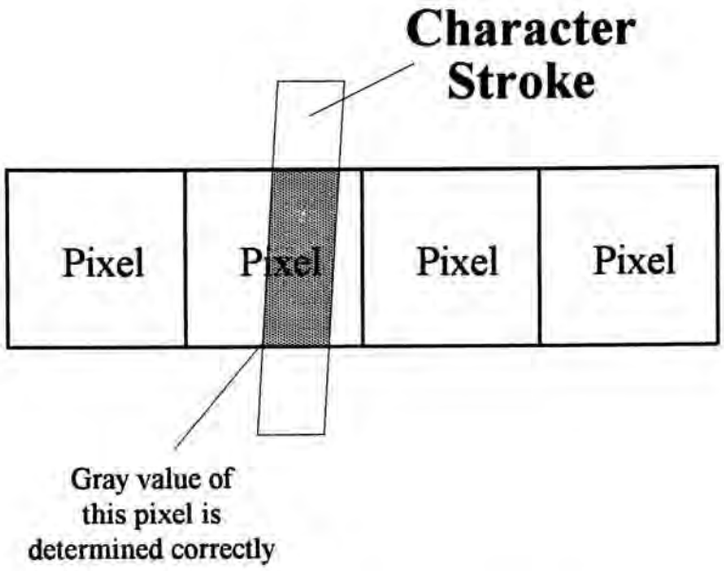


Figure 5.10 The gray value is determined correctly

Category 2 : Pixels A, B and C are neighbour pixels

In Figure 5.11, it is easy to find the area of triangle B by distance d' and the slope of the stroke's edge. Triangles A, B and C are similar. So, the area of Triangle A and C are also found. Thus, we can find the area ratios covered by this stroke on Pixels A and C, which means that the gray values of Pixels A and C can be found. But we cannot find the gray value of Pixel B. We set it to *unknown*.

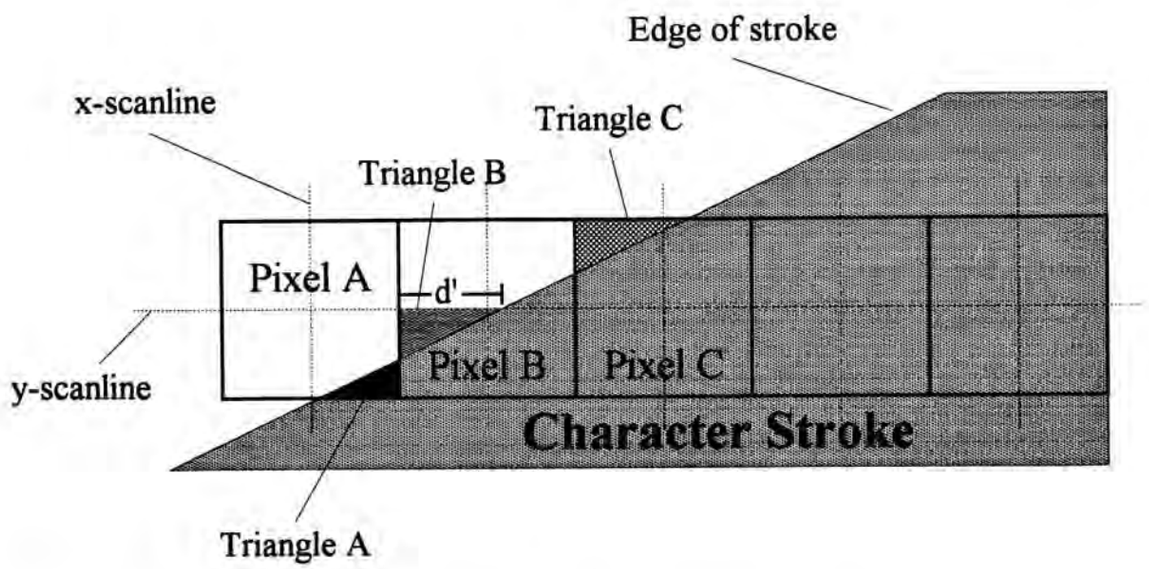


Figure 5.11 Pixels A, B and C are neighbour pixels

Category 3 : Pixels A, B and C are neither the same pixel nor neighbour pixels

Figure 5.12 is an example of such a case. Again, the area of triangle B can be found, and also the areas of triangles A and C. The gray values of Pixels A and C can be determined, but still gray value of Pixel B is left *unknown*. The pixel(s) (may be more than one) between Pixels A and B are also set to *unknown*, as well as those between Pixels B and C.

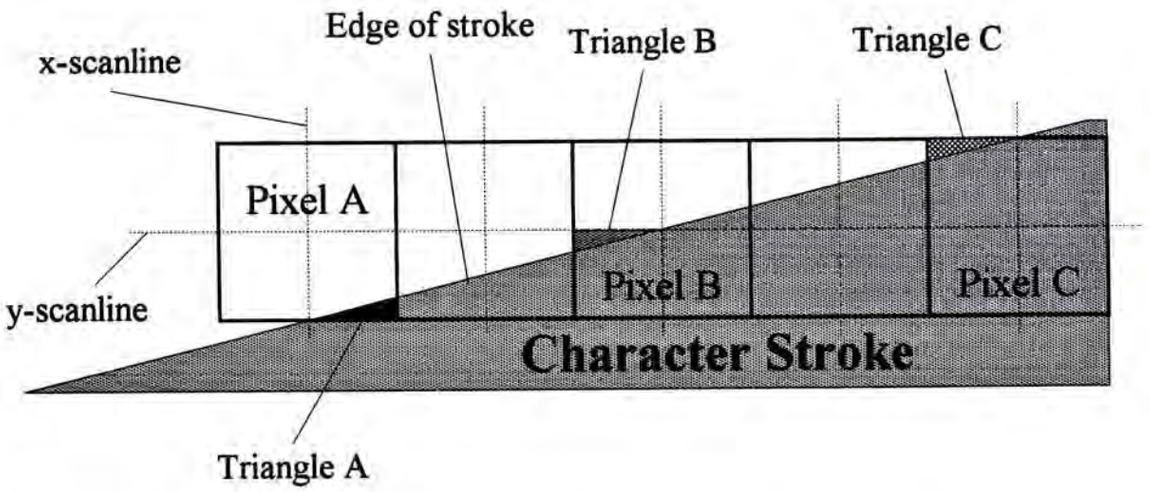


Figure 5.12 Pixels A, B and C are neither the same pixel nor neighbour pixels

In categories 2 and 3, gray values of certain pixels are undefined when considering y-scanlines. However we can see in Figure 5.12 that, when we consider x-scanlines, the pixels that set to *unknown* are all covered as the condition of category 1, that is, their gray values can be computed.

The above three categories cover all possible cases of how an edge of stroke intersecting a scanline. The way to handle the right edge of the stroke and intersections with x-scanlines is similar. Thus, the gray value of all pixels of gray scale character bitmap can be accurately determined accordingly.

Preliminary Results

A problem is observed concerning the gray scale font characters generated by this coarse gray scale rasterizer. If the font size is less than 40x40, an impression of irregularity of strokes is found in some character images (Figure

5.13). Two vertical or horizontal strokes will give different impressions to human readers although they have the same width in the character outline.

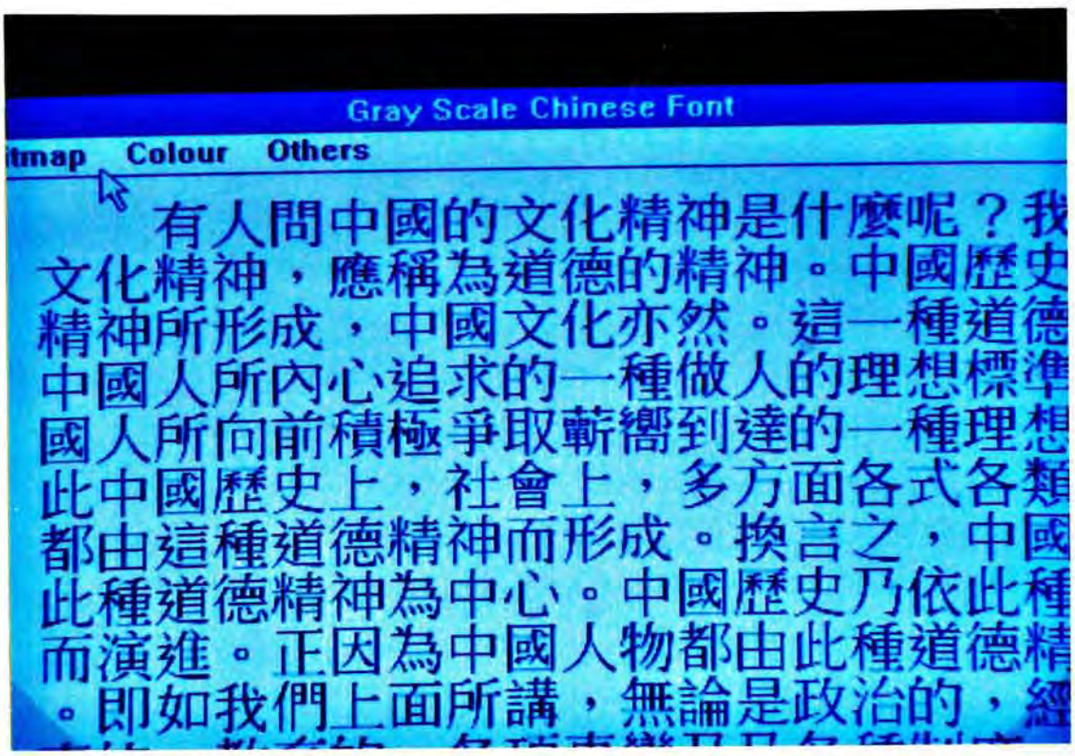


Figure 5.13 Characters generated by the coarse gray scale rasterizer (pixel size 24x24)

In Figure 5.14, we can see how the outline of part of a character could be placed on a coarse rasterization grid. Both of the two vertical strokes are 1.6 pixel wide, but the width of the left stroke covers to 0.7 of one pixel and 0.9 of the other, while the width of the right stroke covers 3 pixels with area ratio 0.1, 1.0 and 0.5 respectively. If 16 gray levels are used, the gray value of the pixels on the left stroke will be (11, 14) and that of the right one is (2, 15, 8). Therefore the impressions of the two strokes will be different, that is, the regularity of the character is destroyed. Hinting methods are, therefore, obvious solutions to overcome this problem.

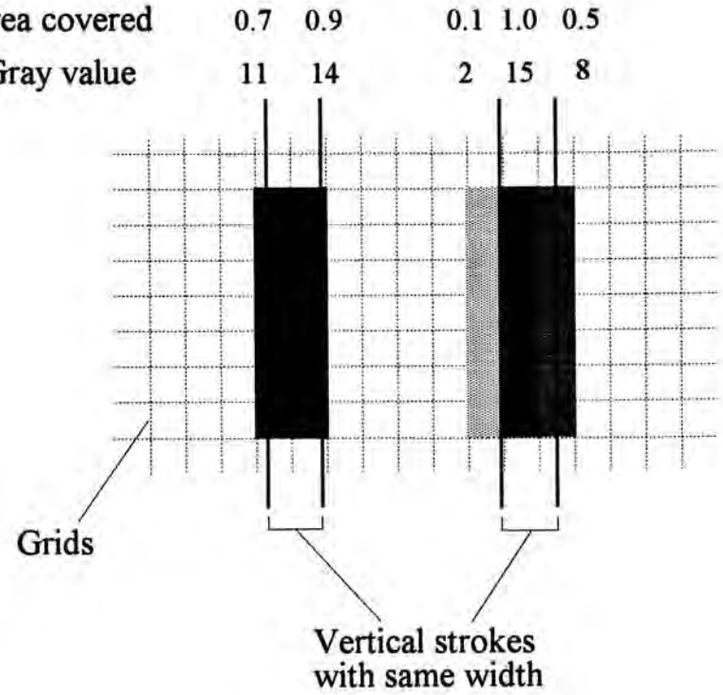


Figure 5.14 Different impression for strokes with same width

Hinting

Rasterization with Hinting

When bitmaps are produced from mathematical descriptions of fonts' outlines by rasterization, aesthetic quality is hard to achieve at low device resolution and/or small font sizes. Manually created bitmaps for characters can be hand-tuned to yield the best possible legibility and "look" of the characters whereas a naive rasterization of outline fonts can yield very unpredictable and hard-to-read characters depending on how the outlines match up with the relatively coarse rasterization grid [Andler 89].

A similar problem occurs in the character bitmap generated by the gray scale rasterizer as mentioned in the previous section. Here we introduced certain modifications, called "hinting", to the font outline. Generally speaking, hints are rules or algorithms that describe how to modify the shape of the outline characters as the size and resolution change. Such hinting modification improves the quality of fonts, especially the ones of small sizes [Deach 92]. Hints are input together with the original font outline to the rasterizer, in which the hints are applied on the font outline to produce adjusted font outline. Then bitmaps are resulted from the

adjusted font outline (Figure 5.15). One of the hinting method, called "Grid-fitting", is used to solve the problem mentioned above.

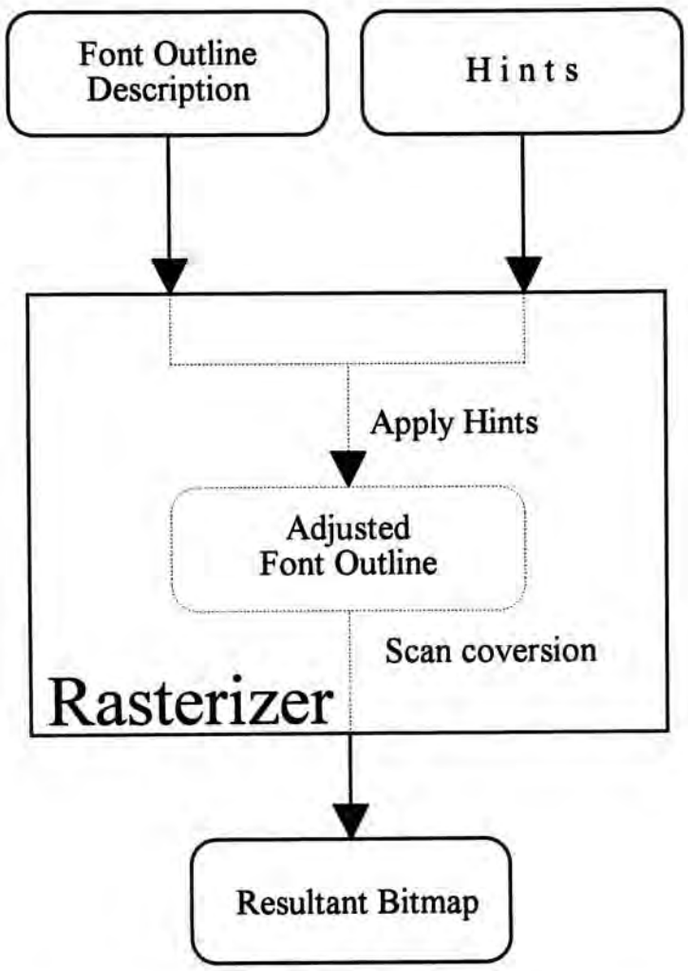
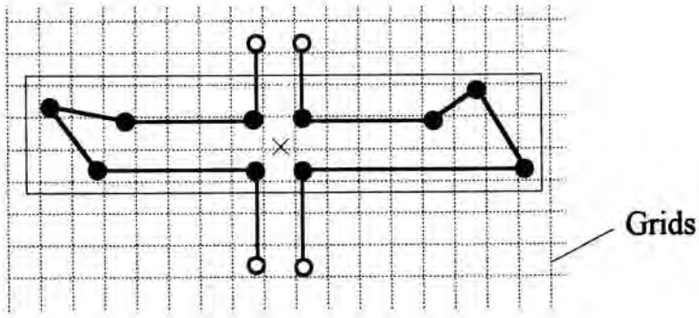


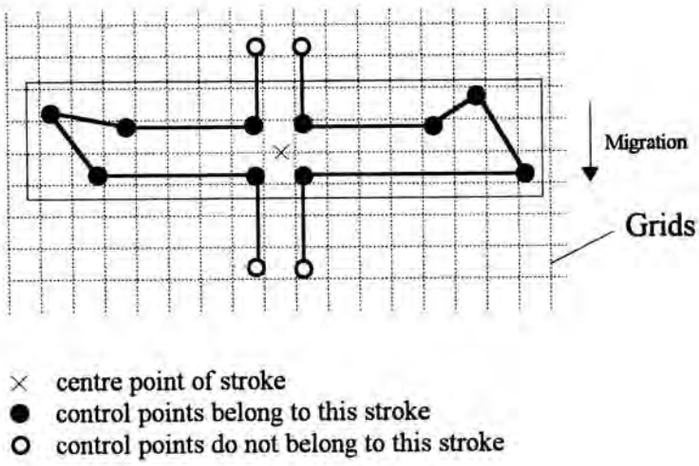
Figure 5.15 Rasterization with hinting

Strokes Migration

To overcome this problem of different impressions for strokes with same width, the outline has to be fitted to the rasterization grid. This can be done by adding some more information to the description of the character to perform migration of stroke position (Figure 5.16). We call this added information "Grid-fitting Hints". Such hints include the centre point of stroke and the control points associated with this stroke. We apply it to both vertical and horizontal strokes. After migration of strokes position, all vertical/horizontal strokes with the same width will be converted to pixel columns/rows with same pattern of gray values. Thus, the regularity of stems in a character can be preserved.



(a) Before migration of horizontal stroke



(b) After migration of horizontal stroke

Figure 5.16 Migration of a horizontal stroke

The stroke migration process moves the center point of stroke to the grid line or the middle of two grid lines, depending on the width of this stroke. For a stroke with width less than a pixel, moving the center point of this stroke to the grid line is not suitable, as the stroke will be made up of two rows of light gray pixels, making it look fuzzy. If the center point of the stroke is moved to the middle of two grid lines, the resultant character image will become sharper because darker and thinner strokes are generated. Furthermore, using smaller number of rows of pixels to represent a stroke makes a character image clearer because of reduction of chance of mixing the strokes in that character.

Hints Finding

To perform stroke migration, we need to know the information of vertical and horizontal strokes of a character. An automatic method of hints extraction from outline of a character has been investigated. The method tries to extract vertical/horizontal strokes by pairing up vertical/horizontal edges of a character, and then produces hinting information for these strokes.

Certain assumptions are made about the fonts when hints are generated:

1. The horizontal and vertical strokes consist of straight lines only.
2. A horizontal stroke must have at least one upper straight line and at least one lower straight line.
3. A vertical stroke must have at least one left straight line and at least one right straight line.
4. All strokes are supposed to be uniform in width, or with only very small differences.

To recognize all the horizontal strokes in a character, all the horizontal (or nearly horizontal) straight lines are found first. These lines are sorted by their order of appearance in the character, say from top to the bottom. They are to be paired up to form strokes according to the following rules:

1. Lines which are not yet paired with any other lines.
2. Lines with distance which is equal to the width (with tolerance) of the horizontal strokes.
3. The x-ranges of the lines overlap.

It is common in Chinese characters that strokes are always overlapping or crossing each other. This makes the strokes be broken down into several parts (as in Figure 5.17). Therefore we should check for collinear stroke parts and join them as one.

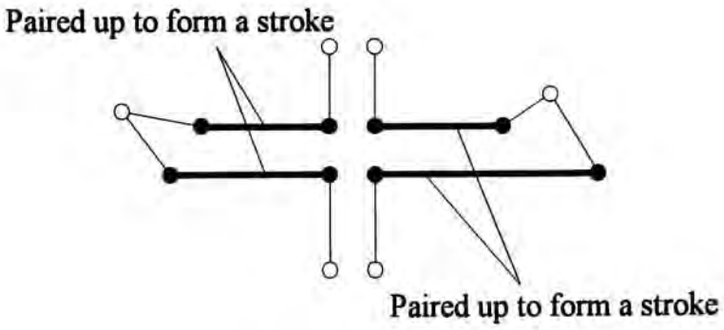


Figure 5.17 A stroke is divided into two parts

However certain characters contain discrete collinear strokes, like "晶" (Figure 5.18). Although we should not consider these collinear strokes as one, problems do not occur when applying hinting on these characters. Migration can still be done on these collinear strokes together as one.

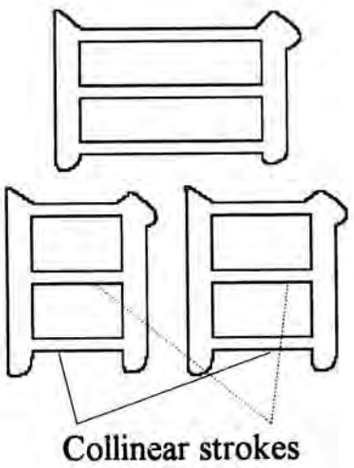
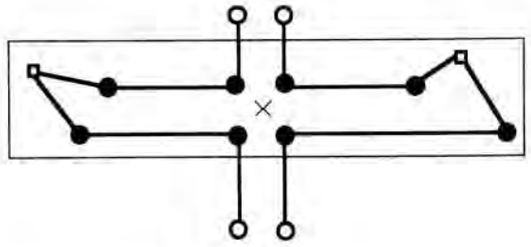


Figure 5.18 A character contains discrete collinear strokes

This method cannot identify the control points of serifs, which belong to this stroke. Curves and slanted strokes cannot be identified also. So the stroke migration process only moves some of control points of vertical and horizontal strokes of a character (Figure 5.19).



× centre point of stroke
● control points belonging to this stroke
▫ control points belong to this stroke but will not recognized
O control points not belonging to this stroke

Figure 5.19 Controls points that cannot be recognized

Although curves, serifs and slanted stems cannot be identified by above method, there are some other methods to handle these characteristics. For instance, serifs can been recognized by pattern matching method [Karow 89]. Curve and slanted stems can been detected by analysing the curve angles and the extrema [Andler 89]. Unfortunately, rasterization is a real time process. It should finish its process in reasonable time. But, the algorithms for stroke recognition are usually very time consuming.

Besides, hints should be applied to a character according to its characteristics. So, developing a universal hinting method for all different typefaces of Chinese font is quite impossible. It is difficult even for a single typeface. The hinting information is better supplied by font designer. Several formats of outline font such as Adobe Type 1, TrueType, Bitstream and Nimbus-Q provide methods to the font designers to describe hinting information [Deach 92].

Rasterization would be very slow if identifying hinting information is included in the process. Therefore we suggest that the finding of hints should be pre-processed, and separated from rasterization.

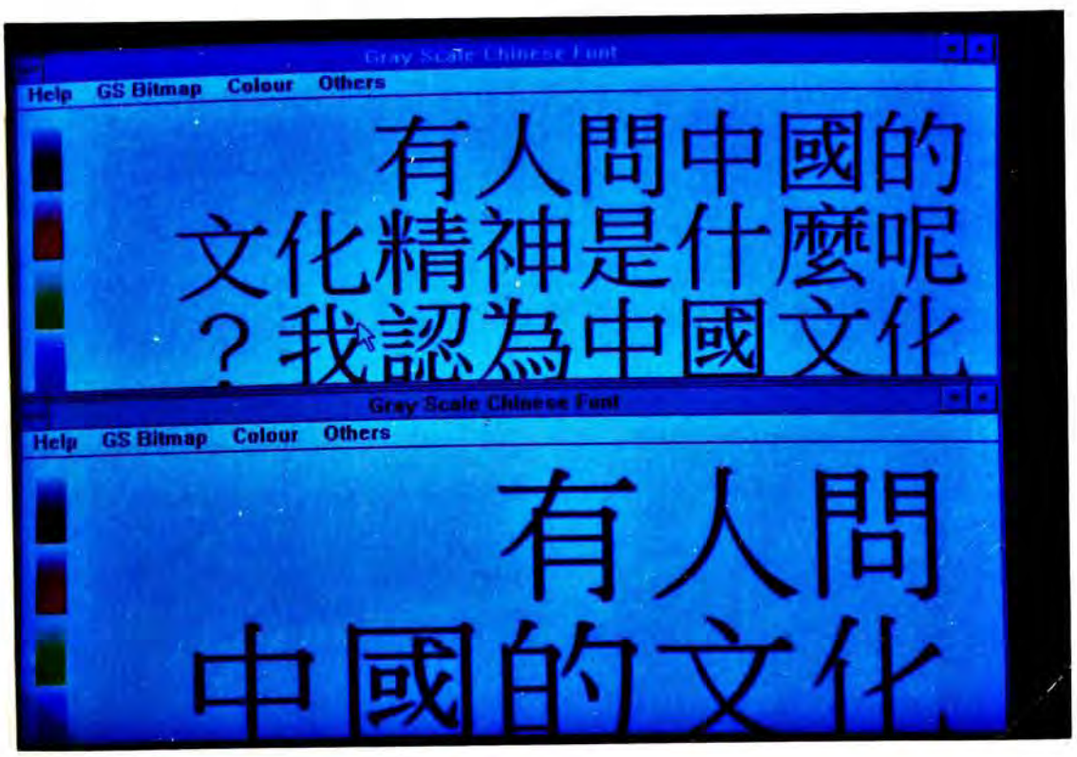
A program is written for finding hints of a character outline. It allows a user to mark the control points of a stroke of a character interactively. Then the information need for stroke migration process will be stored. For each character, the number of vertical and horizontal strokes, their positions, location of their center lines and index of control points will be stored. This information can be retrieved and passed to the rasterizer for the stroke migration process to take place.

Chapter 6

Results and Conclusion

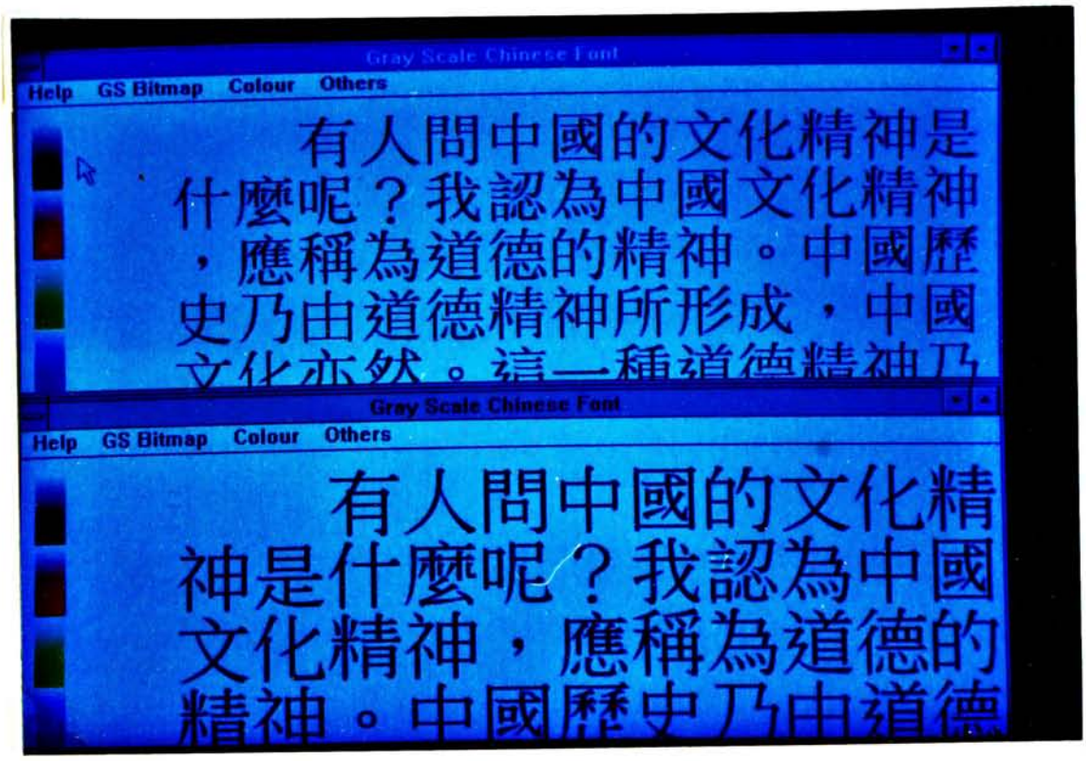
A gray scale rasterizer has been built to implement the method described in Chapters 4 and 5. The rasterization process is performed on a 386-DX33 machine, with a colour CRT display monitor having 640x480 display resolution to present the gray scale characters generated with 16 gray levels. Source data are obtained from the TrueType format outline font library in Microsoft Chinese Window 3.0.

The resultant character images are shown in Figures 6.1(a) to (h). The results will be discussed below in terms of quality and generation speeds.

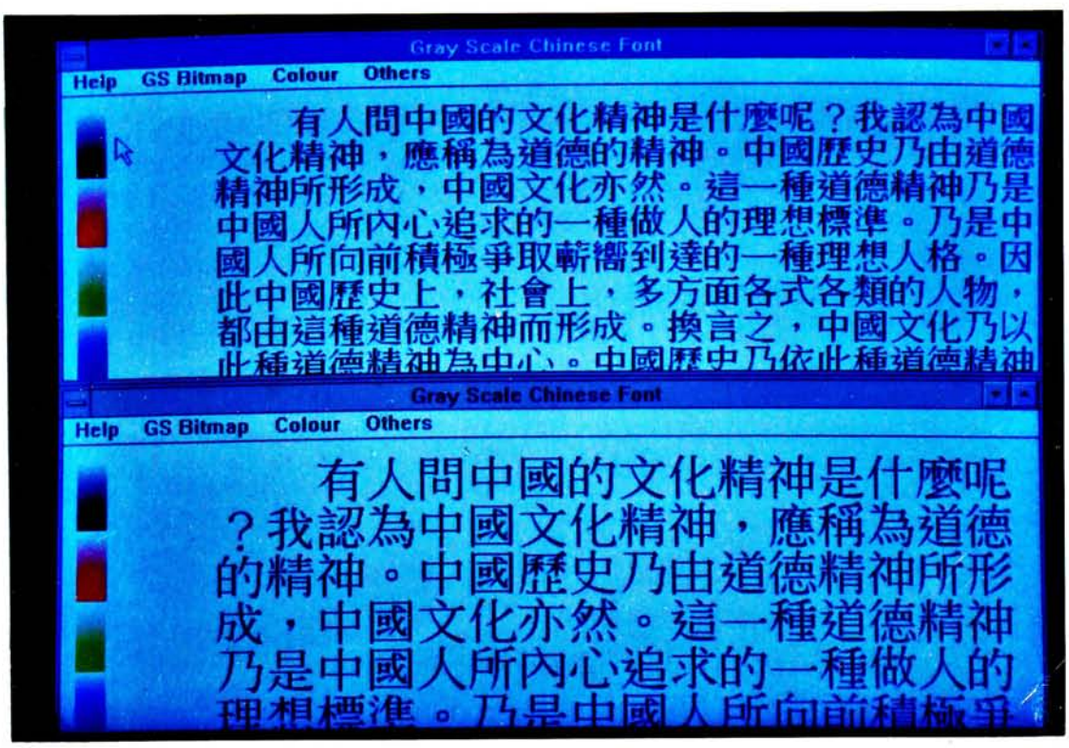


(a) Size 64x64 (upper) and 100x100 (lower)

Figure 6.1 Resultant gray scale characters of different sizes



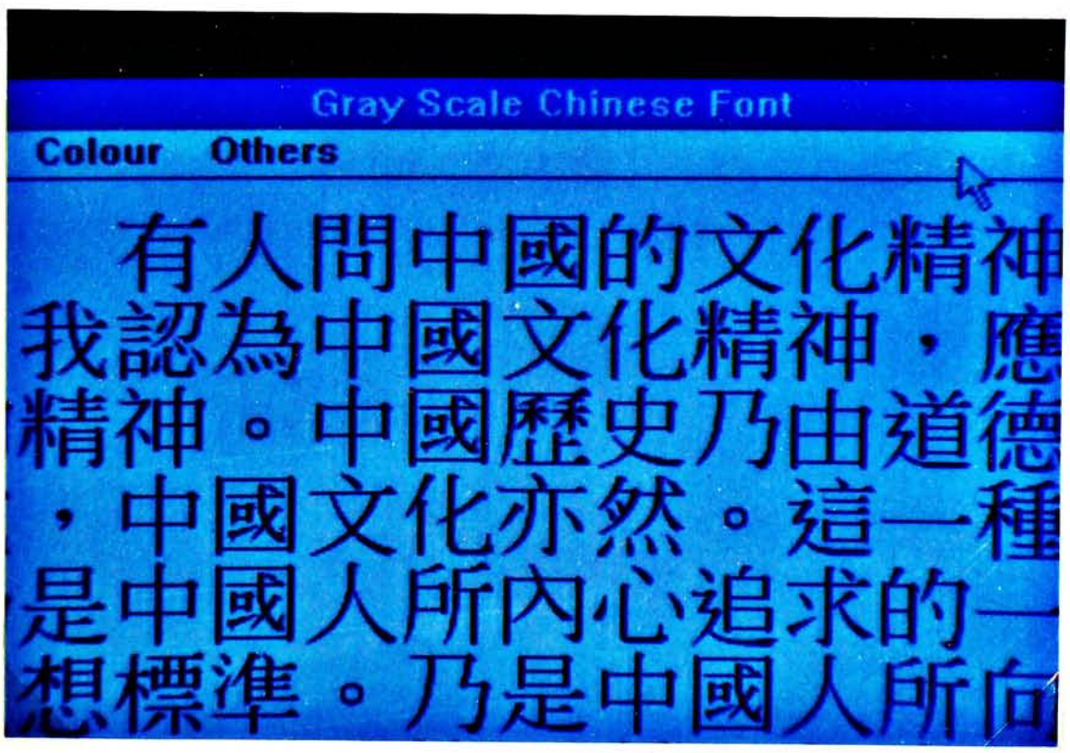
(b) Size 40x40 (upper) and 48x48 (lower)



(c) Size 24x24 (upper) and 32x32 (lower)

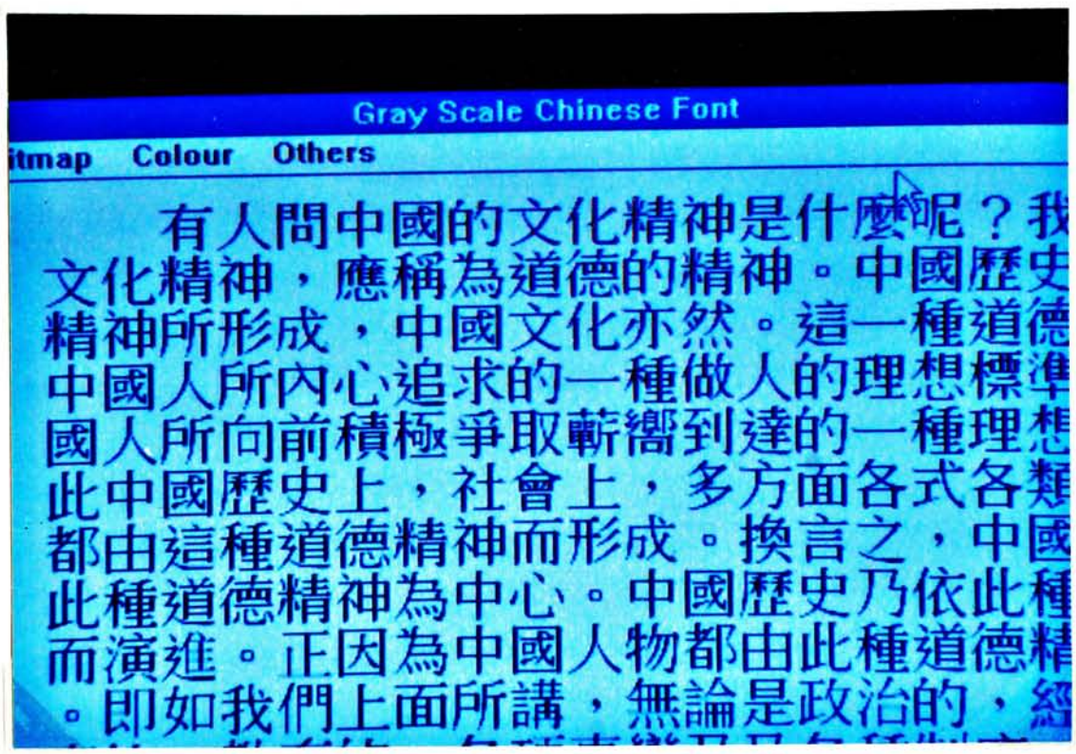
Figure 6.1 Resultant gray scale characters of different sizes (continue)

(d) Size 16x16 (upper) and 20x20 (lower)

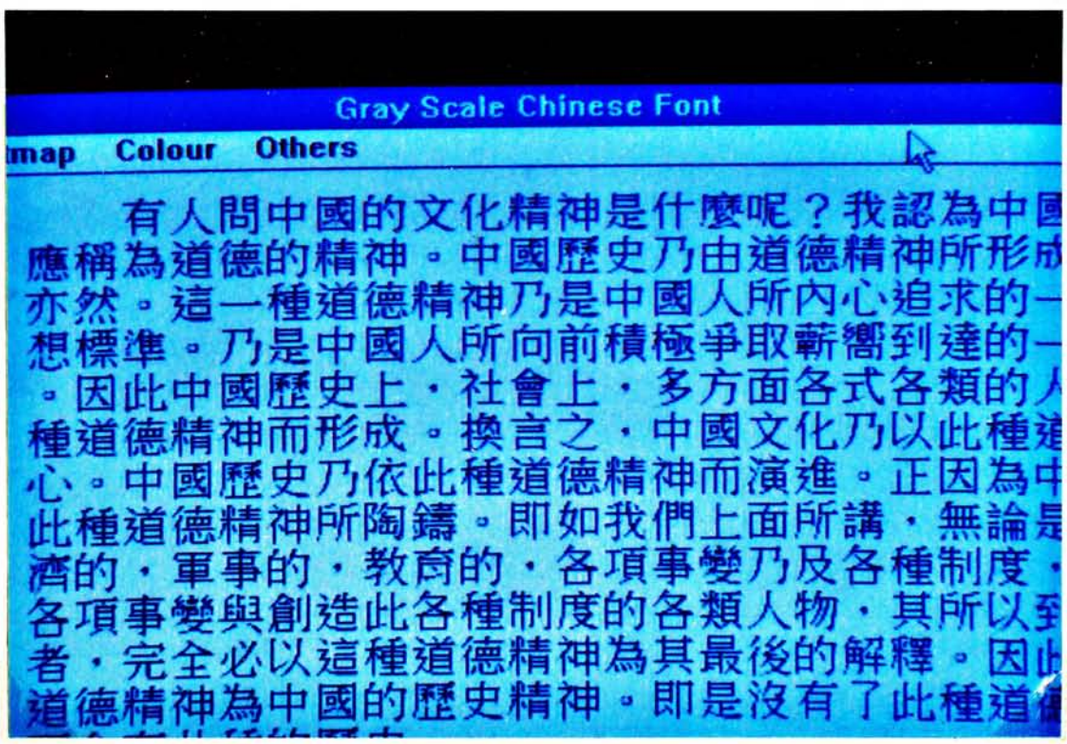


(e) Size 32x32 (close up)

Figure 6.1 Resultant gray scale characters of different sizes (continue)

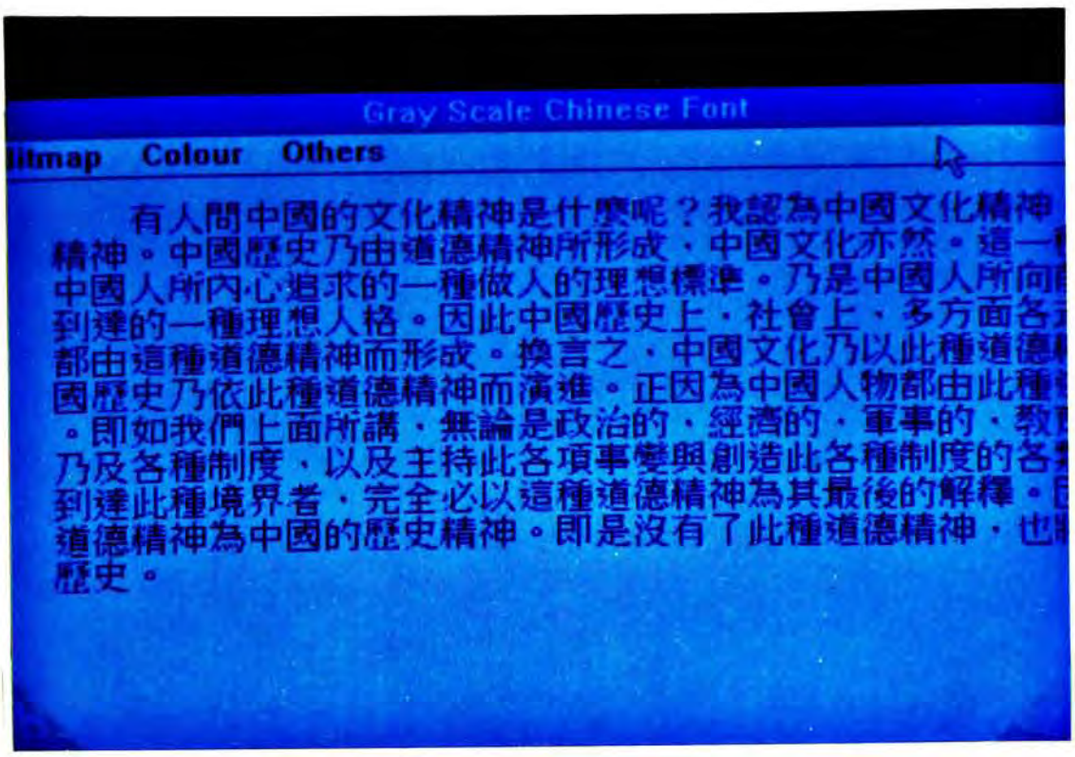


(f) Size 24x24 (close up)



(g) Size 20x20 (close up)

Figure 6.1 Resultant gray scale characters of different sizes (continue)



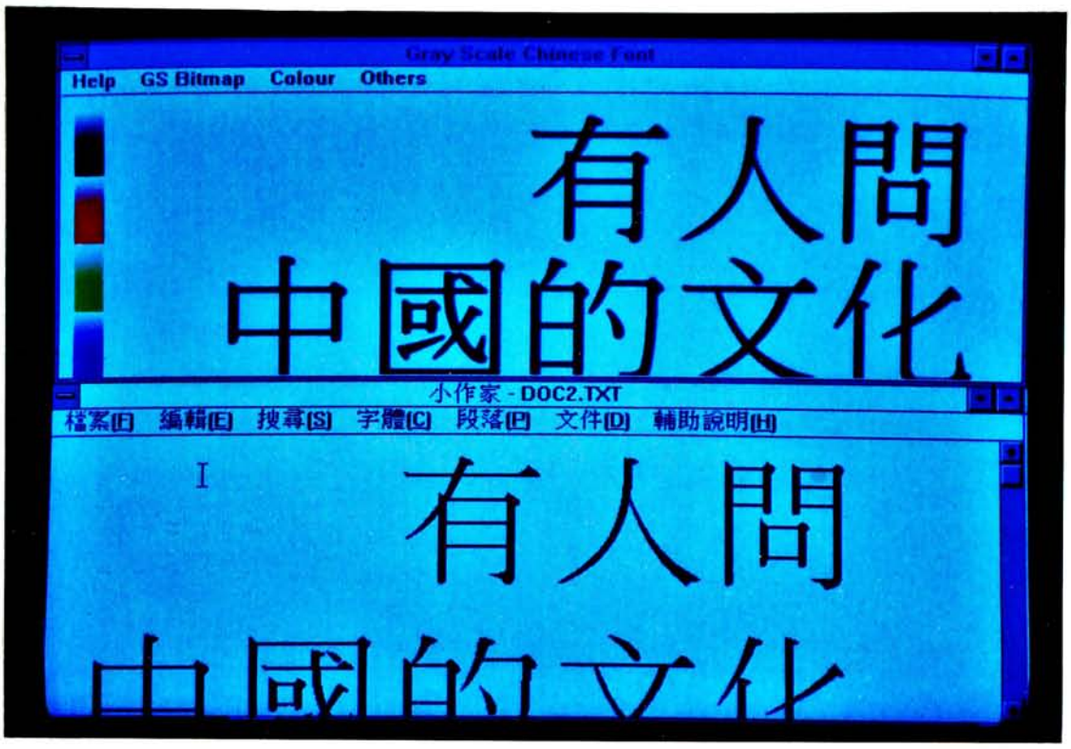
(h) Size 16x16 (close up)

Figure 6.1 Resultant gray scale characters of different sizes (continue)

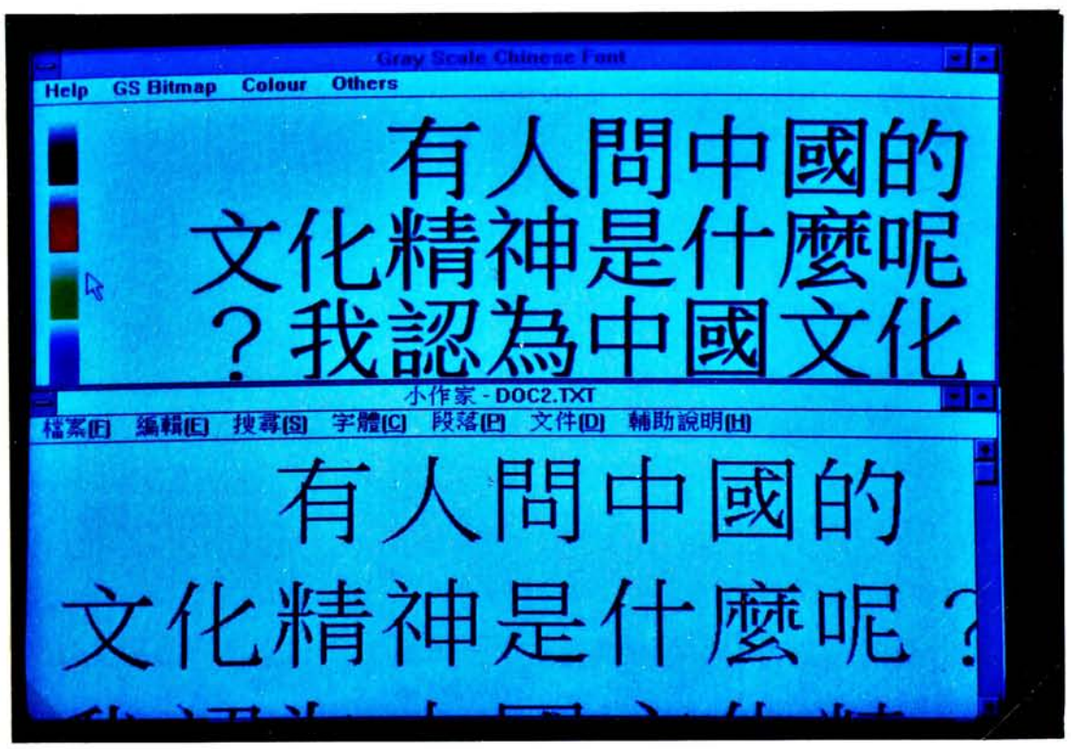
Quality

Comparison with Black-and-White Character

In each of Figures 6.2(a) to (h), gray scale Chinese characters are displayed together with black-and-white bitmap characters. Obviously, the edges of character strokes are smoothened if gray scale technique is applied.

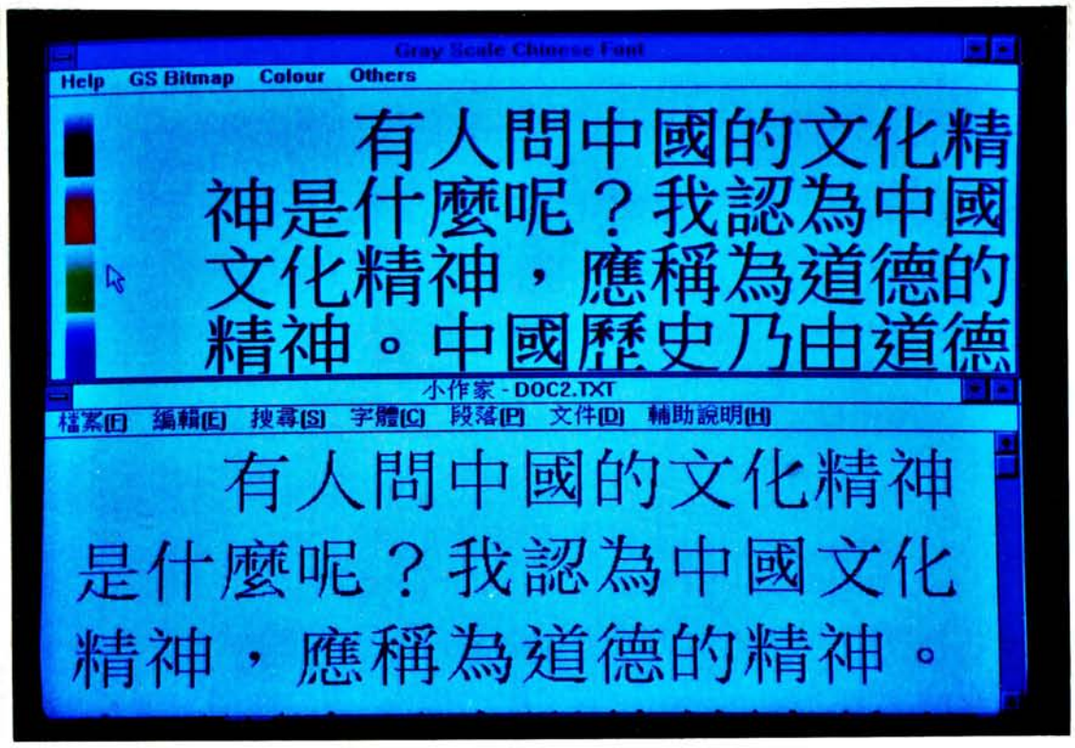


(a) Size 100x100

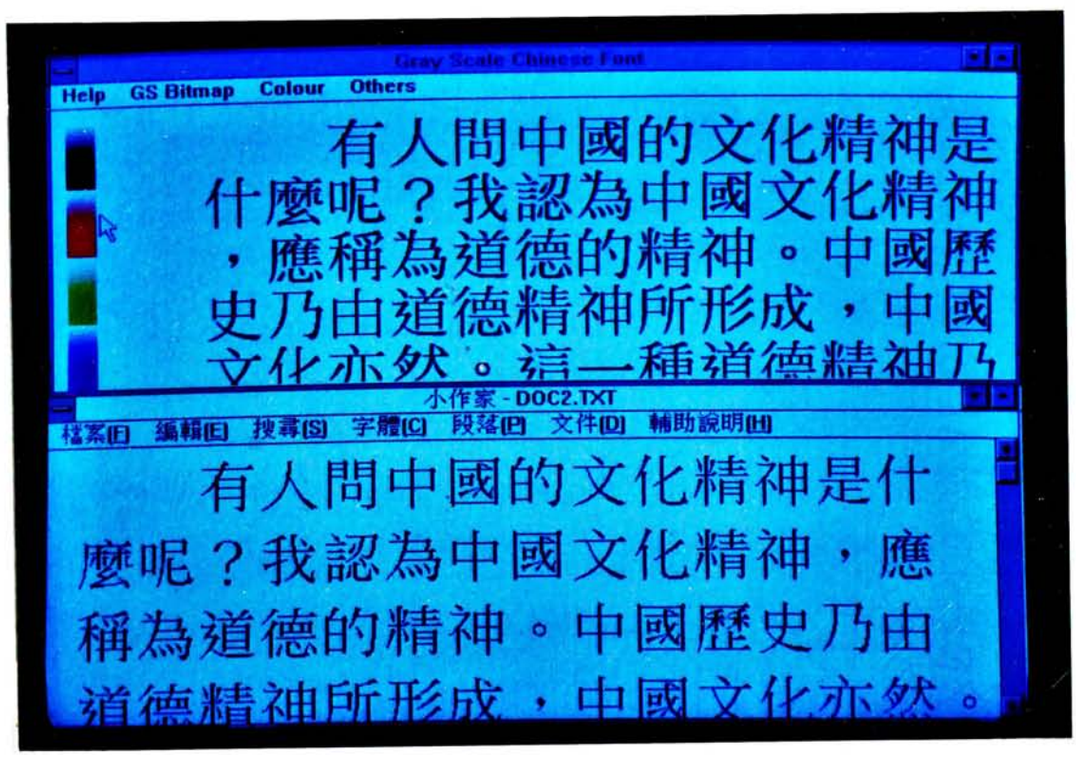


(b) Size 64x64

Figure 6.2 Comparison of the resultant gray scale characters (upper) and black-and-white characters (lower) of different sizes

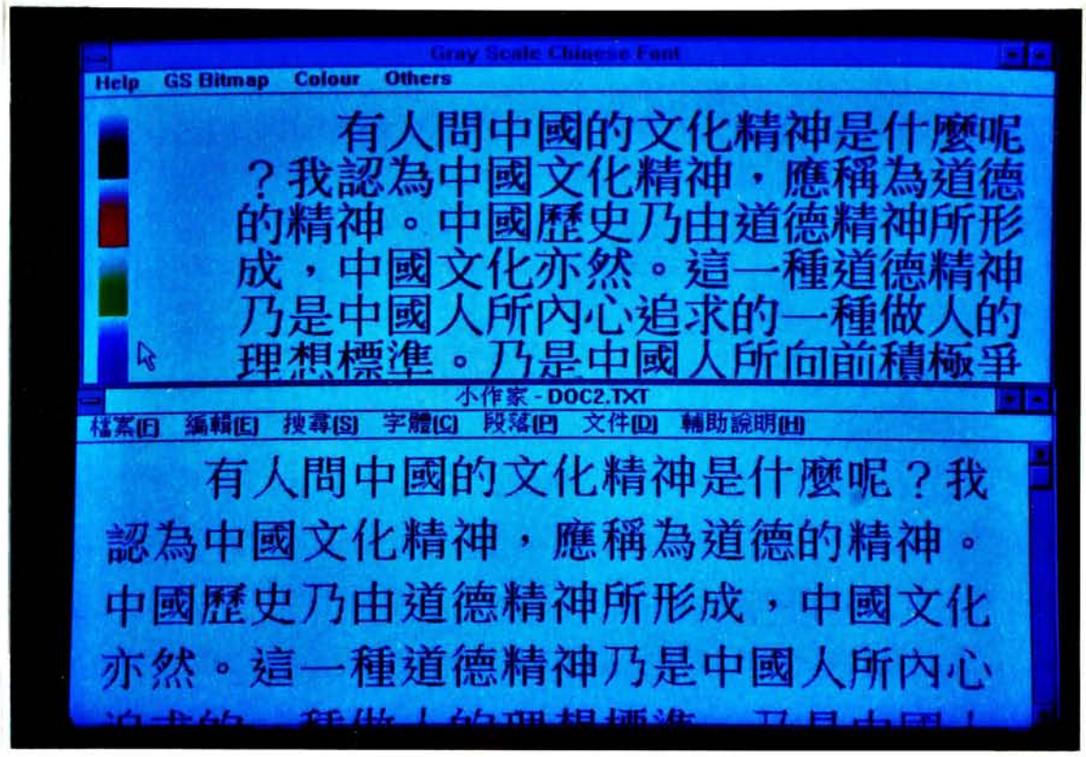


(c) Size 48x48

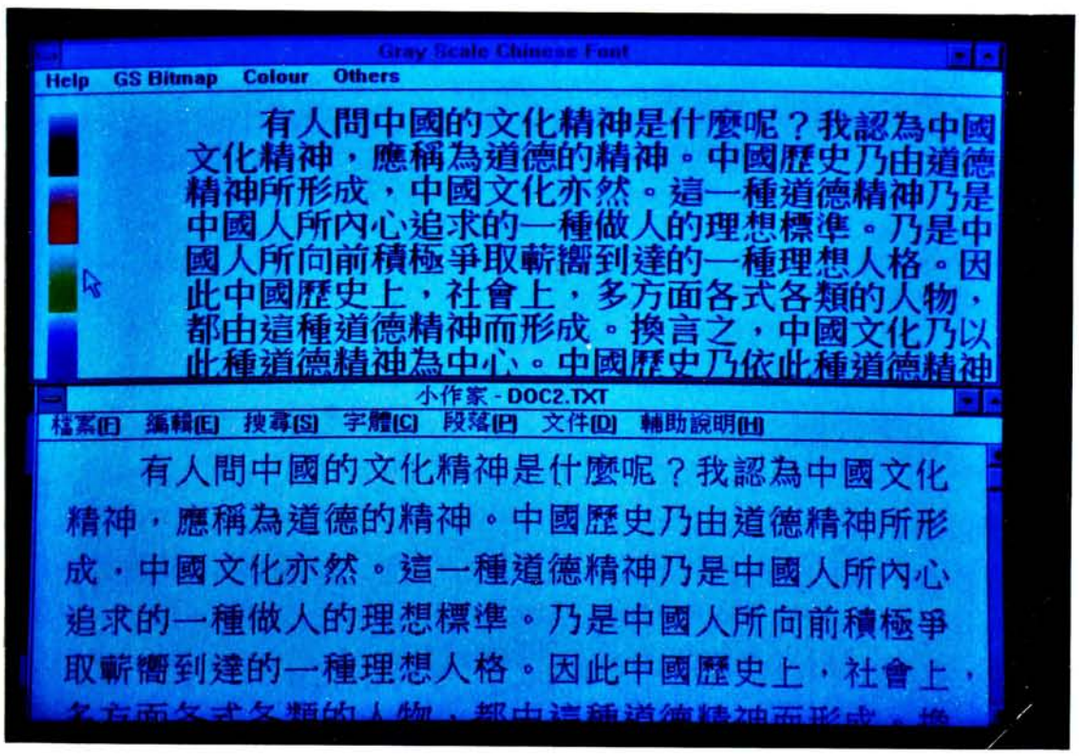


(d) Size 40x40

Figure 6.2 Comparison of the resultant gray scale characters (upper) and black-and-white characters (lower) of different sizes (continue)

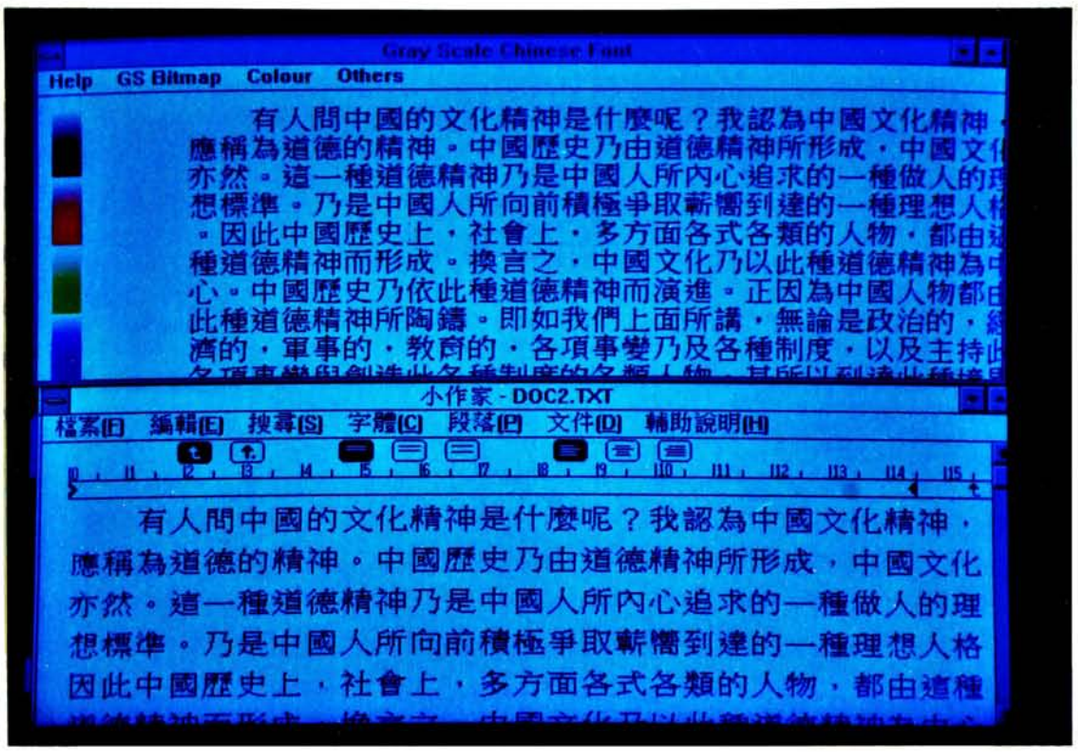


(e) Size 32x32

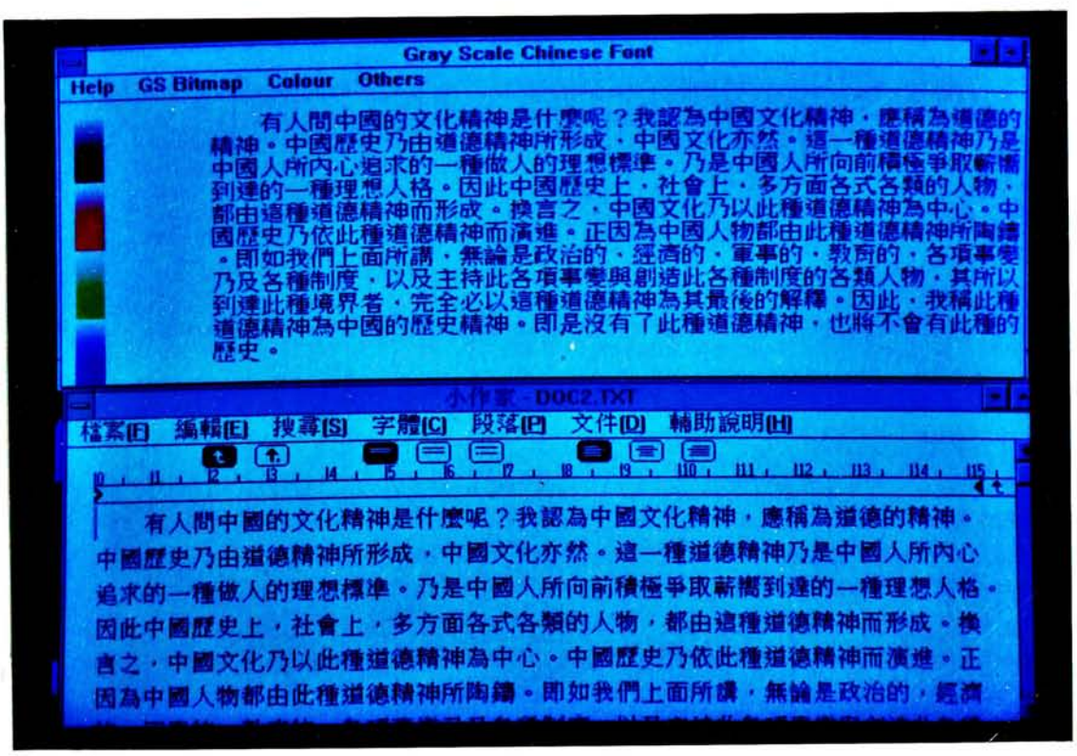


(f) Size 24x24

Figure 6.2 Comparison of the resultant gray scale characters (upper) and black-and-white characters (lower) of different sizes (continue)



(g) Size 20x20



(h) Size 16x16

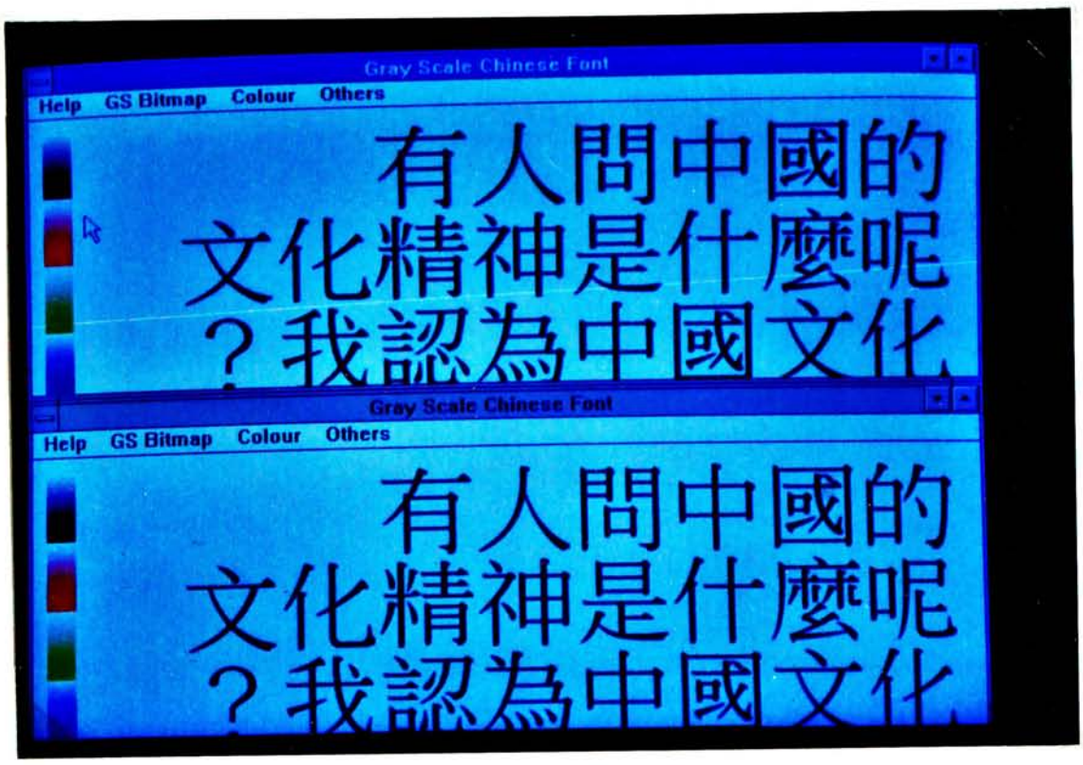
Figure 6.2 Comparison of the resultant gray scale characters (upper) and black-and-white characters (lower) of different sizes (continue)

Hinted Against Unhinted

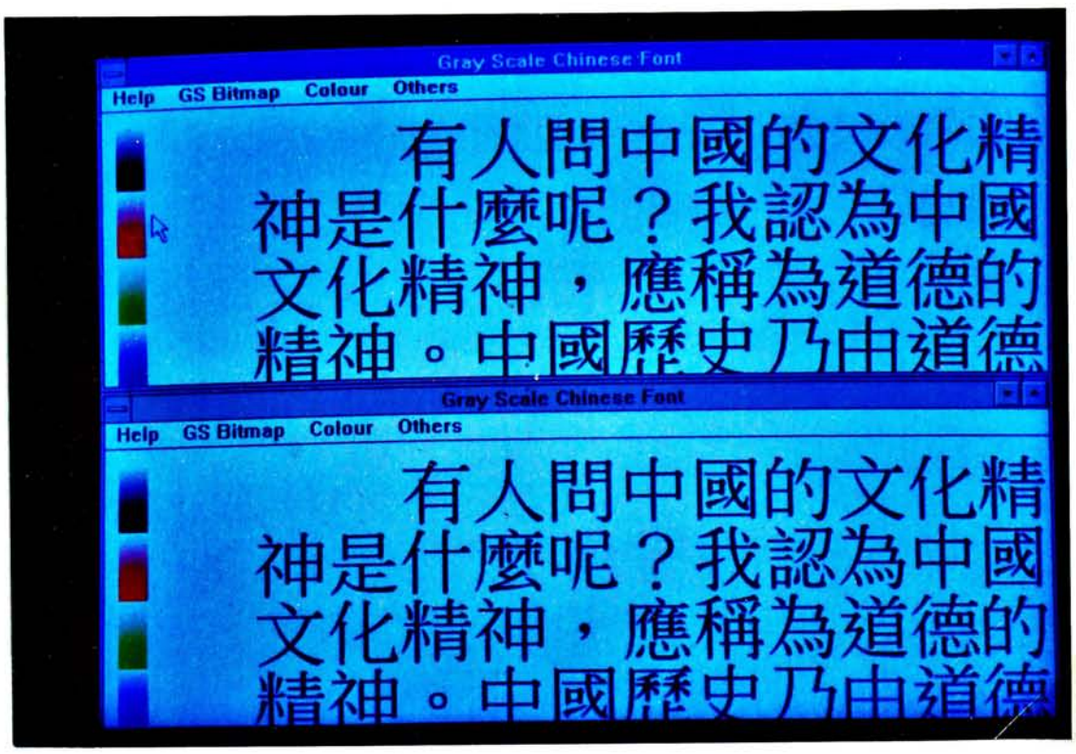
Gray scaled characters with and without pre-processed hinting information (stroke migration) are compared in Figures 6.3(a) to (h). For characters with size 40x40 or above, font quality does not seem improved after stroke migration is applied, while fonts quality of sizes 24x24 and 32x32 significantly improved. However, results for fonts of sizes 16x16 and 20x20 are unfavourable. The thin strokes problem cannot be resolved. To tackle this problem, some other hinting method like "drop-out control" for black-and-white rasterizer may be tried.

(a) Size 100x100

Figure 6.3 Comparison of the resultant gray scale characters of different sizes, with (upper) and without (lower) stroke migration

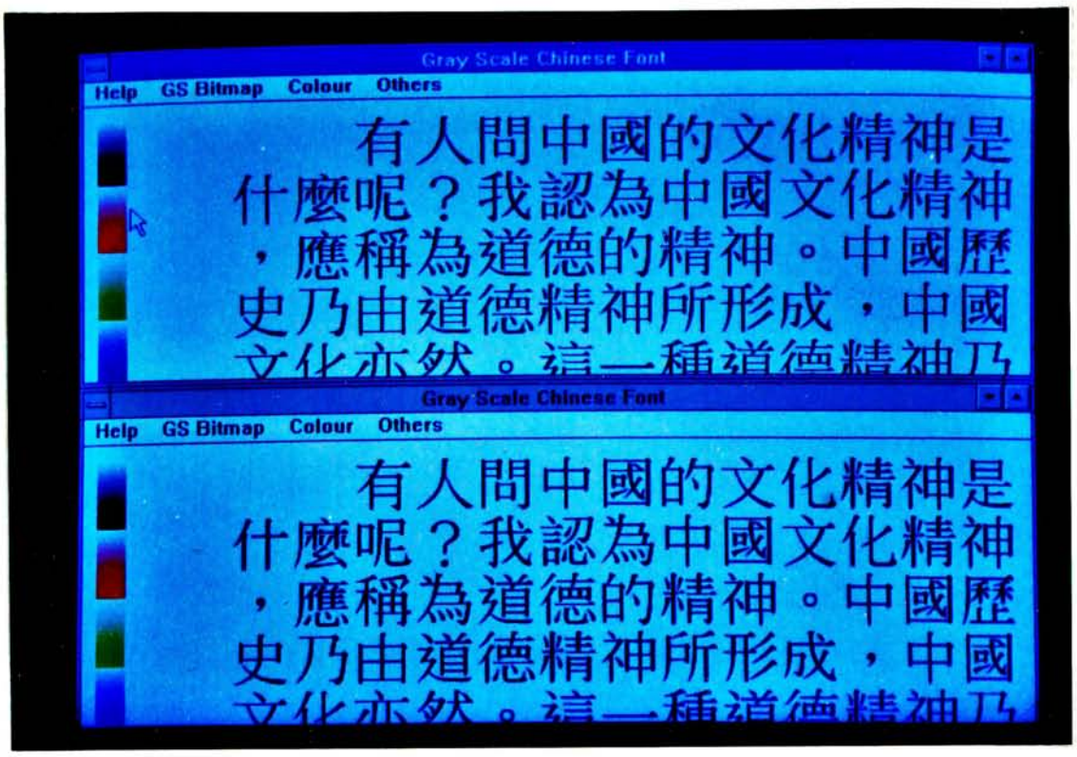


(b) Size 64x64

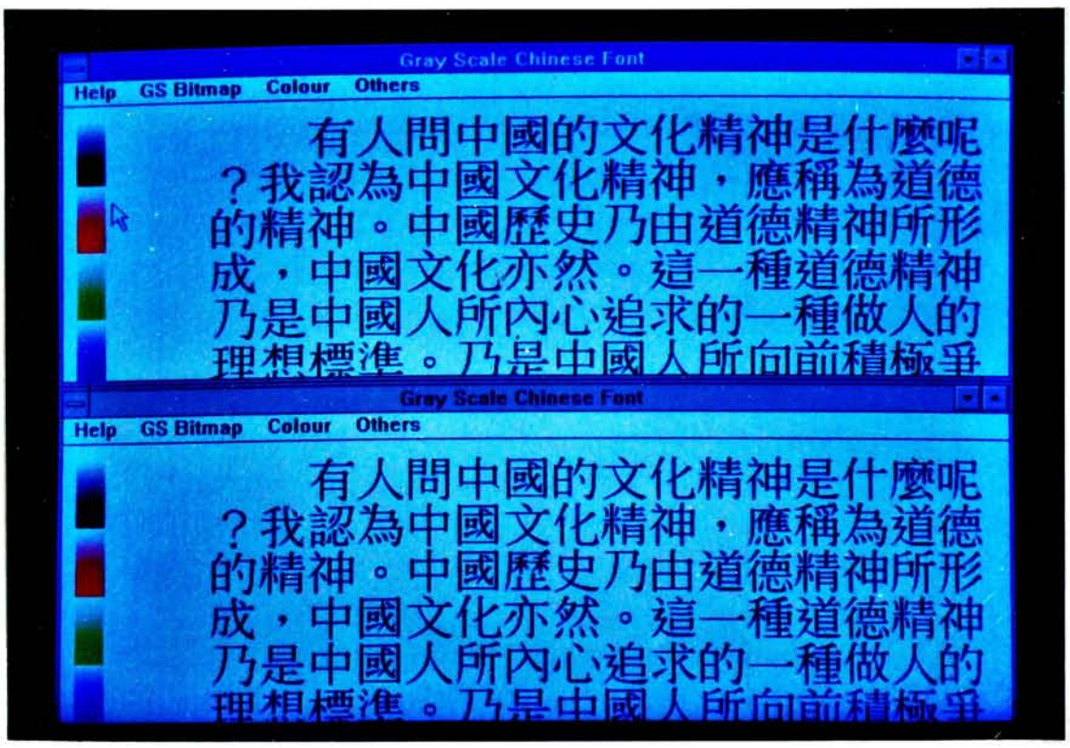


(c) Size 48x48

Figure 6.3 Comparison of the resultant gray scale characters of different sizes, with (upper) and without (lower) stroke migration (continue)

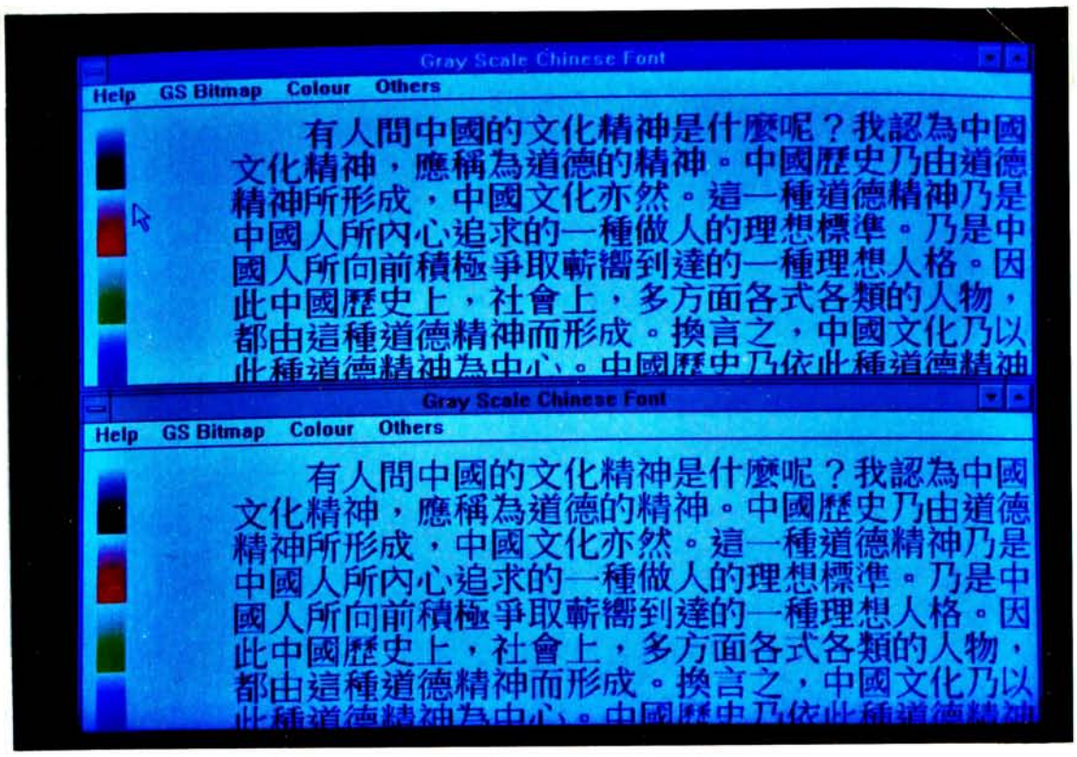


(d) Size 40x40

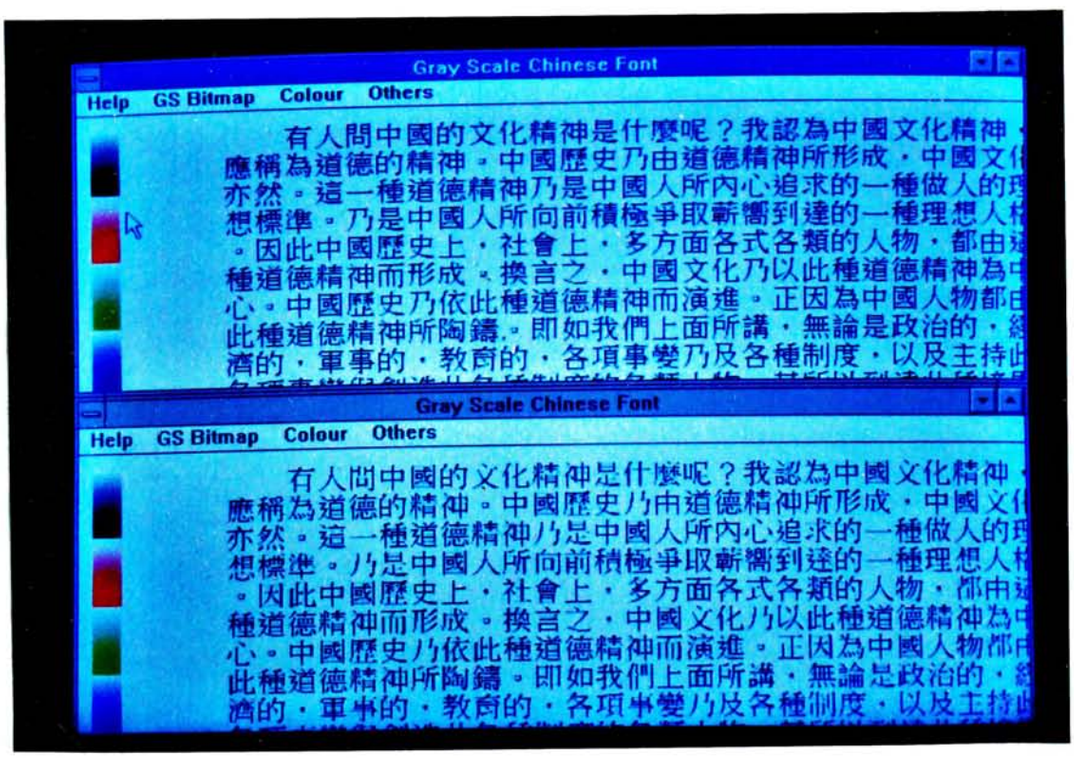


(e) Size 32x32

Figure 6.3 Comparison of the resultant gray scale characters of different sizes, with (upper) and without (lower) stroke migration (continue)

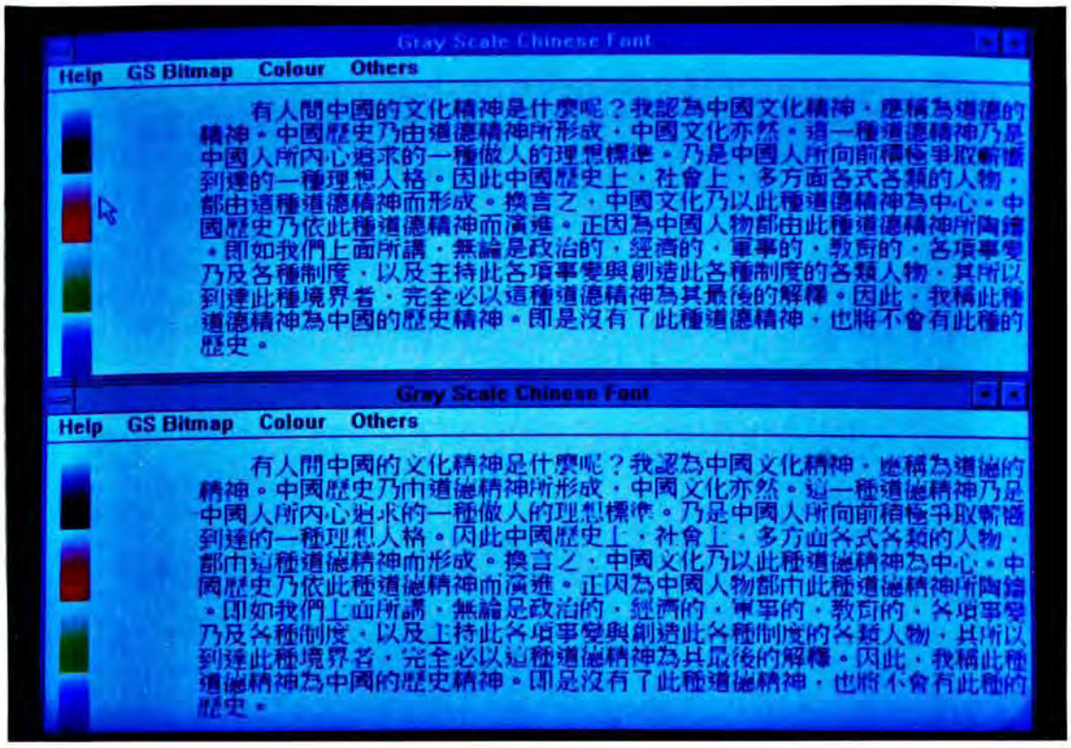


(f) Size 24x24



(g) Size 20x20

Figure 6.3 Comparison of the resultant gray scale characters of different sizes, with (upper) and without (lower) stroke migration (continue)



(h) Size 16x16

Figure 6.3 Comparison of the resultant gray scale characters of different sizes, with (upper) and without (lower) stroke migration (continue)

Generation Speeds

To measure the generation speeds of our gray scale rasterizer (including stroke migration), some experiments are performed. The first one is to measure the time needed to generate characters with different number of strokes. The same character outline will be passed to the gray scale rasterizer 20 times to reduce error. Also, experiments for bitmaps of different sizes are also performed. The results are shown in Table 6.1.

Character (No. of strokes)	No. of curves to form the character outline		Generation time of 20 characters in various sizes of character bitmaps (seconds)					
	St. Lines	Q-Bezier	20x20	24x24	32x32	48x48	64x64	100x100
一 (1)	4	2	2	2	2	3	3	4
大 (3)	8	14	4	5	7	8	10	15
汝 (6)	7	44	8	9	11	15	18	20
高 (10)	39	25	9	11	13	16	19	28
勢 (13)	46	65	15	16	18	23	28	40
篆 (15)	26	59	15	15	18	23	31	44
擦 (17)	39	70	17	18	21	27	34	49
蟹 (19)	73	64	18	21	22	30	35	50
驅 (21)	73	58	18	20	22	27	35	49
鸚 (27)	73	92	22	24	27	34	41	60

Table 6.1 Generation time of 20 characters with various bitmaps sizes (character set 1)

The second set of testing data contains the 10 most frequently used characters. Their accumulated frequency of occurrence is about 11.4% [Bei 85]. Like the first test, each character is generated 20 times repeatedly to obtain the result. Again, different sizes of target characters are tried and the results are shown in Table 6.2.

Character (No. of strokes)	No. of curves to form the character outline		Generation time of 20 characters in various sizes of character bitmaps (seconds)					
	St. Lines	Q-Bezier	20x20	24x24	32x32	48x48	64x64	100x100
的 (8)	22	27	8	9	10	13	17	24
一 (1)	4	2	2	2	2	3	3	4
是 (9)	28	20	8	8	10	13	16	24
在 (6)	23	19	6	7	7	11	13	18
了 (2)	7	12	3	3	4	5	5	9
不 (4)	7	17	4	5	6	8	10	14
和 (8)	24	23	7	7	9	12	15	22
有 (6)	25	18	6	7	9	11	15	21
大 (3)	8	14	4	5	7	8	10	15
這 (11)	32	43	11	12	15	18	23	32

Table 6.2 Generation time of 20 characters with various bitmaps sizes (character set 2)

The last test is to measure the generation time of an article with 331 characters. The content of article is shown below. The result is in Table 6.3.

有人問中國的文化精神是什麼呢？我認爲中國文化精神，應稱爲道德的精神。中國歷史乃由道德精神所形成，中國文化亦然。這一種道德精神乃是中國人所內心追求的一種做人的理想標準。乃是中國人所向前積極爭取蘄嚮到達的一種理想人格。因此中國歷史上，社會上，多方面各式各類的人物，都由這種道德精神而形成。換言之，中國文化乃以此種道德精神爲中心。中國歷史乃依此種道德精神而演進。正因爲中國人物都由此種道德精神所陶鑄。即如我們上面所講，無論是政治的，經濟的，軍事的，教育的，各項事變乃及各種制度，以及主持此各項事變與創造此各種制度的各類人物，其所以到達此種境界者，完全必以這種道德精神爲其最後的解釋。因此，我稱此種道德精神爲中國的歷史精神。即是沒有了此種道德精神，也將不會有此種的歷史。

Size of character bitmap	Generation time of all character in the article (in seconds)	Average generation time of a gray scale character (in seconds)
20x20	151	0.456
24x24	186	0.562
32x32	215	0.650
48x48	265	0.801
64x64	314	0.949
100x100	463	1.399

Table 6.3 Generation time of all character in an article with 331 characters using the gray scale rasterizer

From the results of the above tests, the generation speed of using gray scale rasterizer improves significantly with reference to the filtering method. But, real time performance is still impossible.

Discussion and Comments

There are many difficulties during the implementation of this gray scale rasterizer. The main one is the data source. It is hard to access a Chinese outline font library. The one we used comes from Microsoft Chinese Windows 3.0. The file structure is complicated. We spent lots of time to access the outline font description of a specified character. Another difficulty is that we cannot find an existing Chinese system which can support gray scale font. Thus, we should develop some tools to display gray scale font to evaluate the preliminary results and try to locate the problem induced. Moreover, there is no standard way to evaluate the quality of the generated character image. Perhaps, it is the best to leave the evaluation of character image quality to fontographers.

Although the result obtained at this stage is quite satisfactory, there are still rooms for some further improvement in speed and quality. To reduce speed, when we calculate the gray value of the boundary-pixels, floating point arithmetic takes place. To avoid this time-consuming calculation, a table lookup method can be use. As gray value can be found by

(i) the intersecting point of the scanline and the edge of the stroke, and

(ii) the slope of the edge of the stroke,

we can build a table with (i) and (ii) as index and the resultant gray value as the table content. Then, we can use this table to determine the gray values of boundary-pixels quickly.

On the other hand, while analyzing the result of filtering method, we found that good result is obtained when W=1.5d or 2d. W is the width of the filters and d is the distance of centers of two adjacent filters (refer to Figure 3.3). It means that gray value of a pixel will be affected by its neighbours. In our gray scale rasterizer, gray value is only determined by the information of one scanline. It may be modified to use the information of some neighbouring scanlines. How the gray value of a pixel affected by its neighbours may be a direction for future investigation on this topic.

Practical Font System

Actually, in an Chinese system in practice, font cache is needed (Figure 6.4).

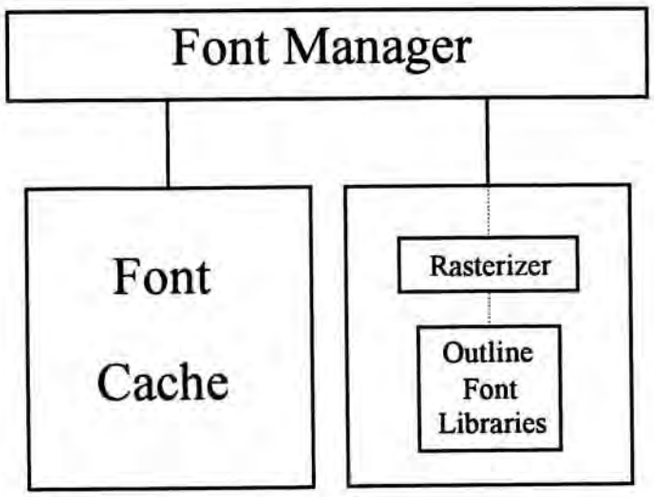


Figure 6.4 A font system with a font cache

A font cache is a memory area used to store character images. When the required character is found in font cache, the character bitmap is retrieved and sent to where it is required. Then, there will be no need to call the rasterizer to generate the required character bitmap and therefore time is saved. The percentage which required character is found in font cache is called "hit ratio". And, the overall performance of a font system with font cache is calculated by

$$\text{Overall Performance} = D_T + h \times C_T + (1-h) \times NC_T$$

where h : hit ratio

D_T : time needed to determine whether the required character is in font cache or not

C_T : operation time when the required character is in font cache

NC_T : operation time when the required character does not in font cache, i.e. time needed for rasterization

According to some researches about the occurrence frequency of Chinese characters, the most commonly used 500 characters make up to 77.4% in our daily use [Bei 85]. Thus, a 500-character font cache will yield a hit ratio 77.4%

which is reasonably high enough to improve much of the performance of the whole font system. For a system of fonts with size 20x20 and 16 gray levels, each character image needs 200 bytes to store. We can approximate the storage required for such a system as 98 Kbytes. And, from the result of the above tests, we can assume $D_T, C_T >> NC_T$. Then the overall performance can be found simply by

$$\begin{aligned} \text{Overall Performance} &= (1-h) \times NC_T \\ &= 0.22581 \times NC_T \end{aligned}$$

Thus, the generation time of this font system will become one fourth of a font system without font cache.

Hence, we have confident to say that using gray scale font in real life applications is quite possible when font cache is used.

Conclusion

A new approach to generate gray scale Chinese character is introduced in this thesis. A gray scale rasterizer which generate gray scale characters directly form characters' outline descriptions is built. It provides a possible way for real-time generation of gray scale characters.

The application of stroke migration can solve the problem of different impression of strokes with the same width in medium-size characters.

There are rooms for further improvement concerning both generation speed and quality. To reduce generation speed, a table lookup method can be used to replace the used method to determine the gray values of boundary-pixels. And, more hinting techniques can apply to the gray scale rasterizer to improve the quality of the resultant character images.

Although gray scale font are more expensive to display, it is worth to be used to provide high quality text display output when display devices having colour ability is being popular. The generation speed is the main barrier in the application of gray scale Chinese font. The idea of this new approach to generate

gray scale Chinese font presented here is only the first attempt. We hope that after some modification, gray scale Chinese font can be used in some system in the foreseeable future.

Bibliography

[Abe *et al.* 91]

Abe, H., Yamamoto, Y. and Ohno, Y., "High Quality Gray-scale Kanji Font Generation Using Automatic Stroke Displacement" in *Raster Imaging and Digital Typography II - Proceedings of the International Conference RIDT'91*, Cambridge University Press, October 1991, pp. 147-155.

[Andler 89]

Andler, Sten F., "Automatic Generation of Gridfitting Hints for Rasterization of Outline Fonts or Graphics" in *Raster Imaging and Digital Typography - Proceedings of the International Conference RIDT'89*, Cambridge University Press, October 1989, pp. 221-234.

[Bei *et al.* 88]

漢字頻度統計——速成識讀優選表，貝貴琴、張學濤匯編，電子工業出版社，1988年。

[Bigelow 85]

Bigelow, Charles, "Font Design for Personal Workstations", *Byte Magazine*, January 1985, pp. 255-270.

[Campbell and Green 65]

Campbell, Fergus W. and Green, Daniel G., "Monocular Versus Binocular Visual Acuity", *Nature*, Volume 208, Number 5006, October 1965, pp. 191-192.

[Campbell and Gregory 60]

Campbell, Fergus W. and Gregory, A. H., "Effect of Size of Pupil on Visual Acuity", *Nature*, Volume 187, Number 4743, September 1960, pp. 1121-1123.

[Crow 78]

Crow, Franklin C., "The Use of Grayscale for Improved Raster Display of Vectors and Characters", *Computer Graphics*, Volume 12, Number 3, August 1978, pp. 1-5.

[Davson 90]
Davson, Hugh, *Physiology of the Eye*, Fifth Edition, Macmillan Press, 1990.

[Deach 92]
Deach, Stephen, "Outline Font Hints and Rasterization: A Technology Primer", *The Seybold Report on Desktop Publishing*, Volume 6, Number 7, March 1992, pp. 21-32.

[Foley *et al.* 90]
Foley, J. D., van Dam, A., Feiner, S. K. and Hughes J. F., *Computer Graphics : Principles and Practice*, Second Edition, Addison-Wesley, 1990.

[Fu 89]
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，陳原主編，上海教育出版社，1989年。

[Fuchs and Knuth 85]
Fuchs, David R. and Knuth, Donald E., "Optimal Prepaging and Font Caching", *ACM Transactions on Programming Languages and System*, Volume 7, Number 1, January 1985, pp. 62-79.

[Gonczarow 89]
Gonczarowski, Jakob, "Fast generation of unfilled and filled outline characters", *Raster Imaging and Digital Typography - Proceedings of the International Conference RIDT'89*, Cambridge University Press, October 1989, pp. 97-110.

[Hersch 87]
Hersch, Roger D., "Character Generation Under Grid Constraints", *Computer Graphics*, Volume 21, Number 4, July 1987, pp. 243-252.

[Hersch 89]
Hersch, Roger D., "Introduction to font rasterization", *Raster Imaging and Digital Typography - Proceedings of the International Conference RIDT'89*, Cambridge University Press, October 1989, pp. 1-13.

[Kajiya and Ullner 81]
Kajiya J. and Ullner, M., "Filtering High Quality Text for Display", *ACM Computer Graphics*, Volume 15, Number 3, August 1981, pp. 7-15.

[Karow 89]
Karow, Peter, "Automatic hinting for intelligent font scaling", *Raster Imaging and Digital Typography - Proceedings of the International Conference RIDT'89*, Cambridge University Press, October 1989, pp. 232-241.

[Naiman 91]
Naiman, Avi C., "The Use of Grayscale for Improved Character Presentation", Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, March, 1991.

[Naiman and Fournier 87]
Naiman, Avi and Fournier, Alain, "Rectangular Convolution for Fast Filtering of Characters", *Computer Graphics*, Vol. 21, No. 4, July 1987, pp. 233-242.

[Microsoft 92]
TrueType Font Files, Version 1.00, Microsoft Corporation, October 1992.

[Moon and Cheang 91]
Moon, Y. S. and Cheang, S. M., "Screen Fonts for Chinese Windowing Systems", *Proceedings of the International Conference of Computer Processing of Chinese and Oriental Languages*, August 1991, pp. 63-68.

[Moon and Poon 92]
Moon, Y. S. and Poon, C. C., "Experiments of Gray Scale Chinese Fonts", *Proceedings of the International Conference on Computer Processing of Chinese and Oriental Languages*, December 1992.

[Moon and Shin 90]
Moon, Y. S. and Shin, T. Y., "Chinese Fonts and their Digitization", *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, September 1990, pp. 235-248.

[Moon *et al.* 92]
Moon, Y. S., Poon, C. C. and Leung, K. F., "Preliminary Studies of Gray Scale Chinese Fonts", *Proceedings of Computer Processing of Asian Languages 2*, March 1992, pp. 113-122.

[Rubinstein 88]
Rubinstein, Richard, *Digital Typography : An Introduction to Type and Composition for Computer System Design*, Addison-Wesley, 1988.

[Schmandt 80]
Schmandt, Chris, "Soft Typography", *Information Processing 1980*, North-Holland, Amsterdam, 1980, pp. 1027-1031.

[Tung 81]
Tung, Yun Mei, "LCCD - A Language for Chinese Character Design", *Software -- Practice and Experience*, Volume 11, June 1980, pp. 1273-1292.

[Warnock 80]
Warnock, John. E., "The Display of Characters Using Gray Level Sample Arrays", *ACM Computer Graphics*, Volume 14, Number 3, July 1980.