

**LCCD,
A LANGUAGE FOR CHINESE CHARACTER DESIGN**

by

Tung Yun Mei

Research sponsored by
National Science Foundation

DEPARTMENT OF COMPUTER SCIENCE
Stanford University

LCCD, A Language for Chinese Character Design

by

Tung Yun Mei

Computer Science Department
Stanford University
Stanford, Ca.

and

Institute of Computing Technology
Academia Sinica
Peking, China

1. Introduction

The Chinese characters for daily usage on computers do not demand an especially goodlooking shape. The most important thing is readability: they have to be recognizable with ease; therefore we could construct each character shape with line segments. But for some applications, for example in computerized typesetting, or even just in printing a rather formal document, it is necessary to use the characters of several fonts with different shapes and sizes, and they should be good looking and neat. Therefore it is important to make a computer produce Chinese characters that fulfill aesthetic requirements.

The character shape generated by a computer is actually a dot matrix of 0's and 1's, with ink to be placed in the positions that are 1 while the 0 positions are

The preparation of this report was supported in part by NSF grant IST-7921977.

to be left blank. If the printing equipment to be used is a high resolution machine, i.e., if there are sufficiently many pixels per square unit, then the discrete dot matrix becomes a continuous plane figure in our eyes. Thus the problem is to generate a satisfactory dot matrix.

There seem to be three ways to attack this problem.

Manual approach: Design the dot matrices one character at a time by using pen and paper manually, then encode them and input to the computer. Even though this would be feasible, it undoubtedly demands a tremendous quantity of work, and the work would have to be repeated for each font and each output device of different resolution.

Optics approach: By using special optical scanning equipment, discretize each character shape and read it into the computer; the resulting shape is displayed on the screen for possible changes by an operator until it looks all right. This approach is an efficient one and it has been used successfully. But it must be repeated for different machines and styles; and a lot of editing is necessary even when the optical equipment is of the highest quality, since strokes will discretize to different widths when the character image is perturbed slightly.

Graphics approach: Treat each Chinese character as a special kind of plane figure that is to be drawn by the computer using a simulated brush. In this case the problem of character design becomes a computer graphics problem.

To the best of the author's knowledge, Donald Knuth was the first to use the graphics approach for alphabet design [1][2]. This approach is a means of character shape creation, while the optics approach is just copying. Knuth has created many excellent fonts with his system METAFONT, and it has a potentially infinite ability in creation of different styles of type.

In the summer of 1979, the author developed an experimental system for Chinese character design [3] at the AI Laboratory of Stanford University. After that, he studied METAFONT and tried to use it for the same purpose. Based on those experiences, LCCD was developed. Many concepts of LCCD came from METAFONT, and its implementation took advantages from METAFONT in some aspects too. But the kernel part, i.e., the method of stroke generation, is completely different.

LCCD is a language specially intended for high quality Chinese character design, but of course it can be used for any kind of characters.

What is the difference between a general font compiler and a Chinese character font compiler? A general font compiler is used for a rather small group of symbols; for example, the goal of METAFONT is a group of 128 characters. Character

shapes are also rather simple — letters, digits, punctuation, and mathematical notation. Basically, every character shape is designed independently, although there are subroutines for common parts of letters. The emphasis of design and use of this kind of font compiler is how to generate the strokes as nicely as possible.

The problems that a Chinese character font compiler faces are quite different. The first one is quantity. Probably there are 50,000 to 60,000 Chinese characters all together, and among them 6,000 to 8,000 are in daily use. Therefore the computer system must do its best to reduce the working quantities that are necessary for character shape design. Secondly, Chinese characters usually have a rather complex shape, the most complex characters containing thirty or more strokes. Chinese characters are plane figures with internal structure; otherwise people would not be able to learn and acquire such a difficult literal, and it would not have such a long history (more than two thousand years!). In fact, the internal structure is rather complex: A character shape is usually subdivided into several subcharacter shapes, and each subcharacter might be subdivided again till reaching basic strokes. The subcharacters that can be used to compose different characters are called radicals. The position of radicals is somewhat like the letters in English, except that Chinese characters have a two-dimensional structure. When the same radical appears as a part of different characters, not only the relative position, but also the size is different; besides that, it might also be rotated. Whether or not a character looks nice depends not only on the niceness of basic strokes; the structure skeleton is much more important. During two thousand years a great many Chinese calligraphers have created a lot of different styles, and many calligraphy rules have been summarized to teach people how to write Chinese characters. But those rules are artistic principles with high abstraction, and it is difficult if not impossible to induce digitalized quantitative rules from them.

So, in conclusion, the problem that a Chinese character font compiler faces is much more difficult and complex than that of a general font compiler.

LCCD is an interactive font compiler. The font designer should work in a bottom-up manner; i.e., the first step is to design the basic strokes, from them to build radicals, and then to proceed from simple to complex, using radicals for generating characters. The designer is able to monitor the generating process by watching a display screen, making adjustments till he is satisfied. The font designer does not have to be a calligrapher, but he had better know something about calligraphy in order to obtain excellent character shapes. The author isn't a qualified person, so the sample appended to this paper is restricted by his calligraphy level; however, the sample does appear to look better than any of his handwriting, so there is good reason to expect that a more qualified person will be able to achieve truly excellent results.

It is important to note that we should not evaluate a Chinese character font compiler solely from the calligraphic standpoint. The traditional Chinese calligraphy uses "four treasures of the studio" as writing tools. By means of the special writing tools and the writing method determined by those tools, we can achieve special effects, and some of these effects cannot be obtained with computer output equipment. On the other hand, some potentially desirable things that a font compiler can do with ease may be extremely hard for human beings using brushes or pens; so the job of generating characters by machine is not the same as traditional calligraphy. But a font designer should know something about the traditions, because the aesthetic concepts that have formed over such a long period of time are reflected in the modern forms.

2. Drawing procedure

A Chinese character is treated as a plane figure, and LCCD is used to formalize the drawing procedure. In general how can a drawing procedure be formalized? Try to imagine that you are teaching a little girl how to draw a picture of a panda, and you do it in the following way:

First, a group of points are determined on paper (of course under your help, because she doesn't have any experience). Then you give her a pen, and tell her to draw a curve to connect some points; thereby the panda is outlined. The last step is to paint some region in color. The results maybe look like this:



Fig. 2-1 A group of points.



Fig. 2-2 An outline drawing.



Fig. 2-3 The panda.

In the same way we design character shapes with LCCD: we choose some points on rectangular grids by giving their coordinates; we connect some of them by smooth curves which will be called paths; we choose a certain pen or eraser

(there are seven kinds); and we move the pen or eraser along the path. Another possibility is to fill or erase a region that lies within a closed curve.

The drawing procedure of a picture can be defined as a subroutine, and used to construct other new pictures. In calling a subroutine, its figure can be magnified, contracted, translated and rotated as desired. Therefore, it is very easy to produce a team of pandas from a prototype panda.

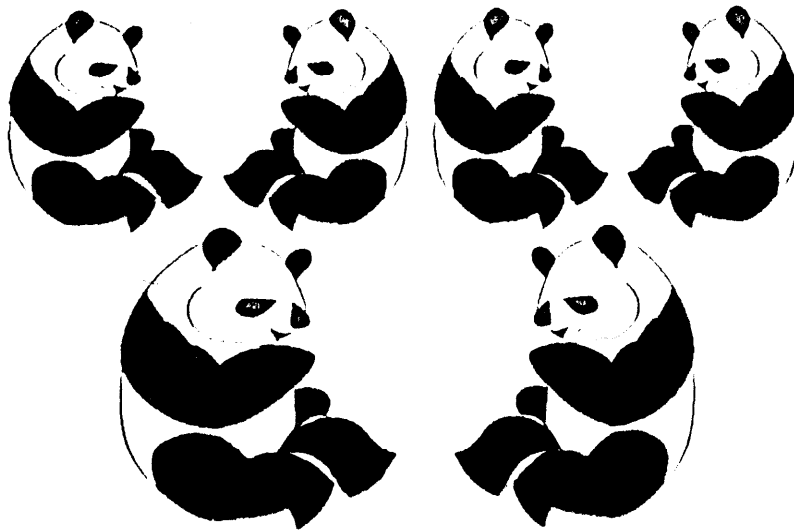


Fig. 2-4 A team of pandas generated from a single design.

3. Variables and expressions

In order to draw pictures, we have to be able to define grid points on a plane, i.e., to specify points with integer numbers as coordinates. The LCCD system allows up to 128 different points to be defined; they are specified by $z_0, z_1, \dots, z_{127}$, respectively, and called z variables. Their abscissas and ordinates are specified respectively by $x_0, x_1, \dots, x_{127}$ and $y_0, y_1, \dots, y_{127}$. These x, y are called variables.

Expressions and z expressions are evaluation rules; an expression evaluates a real number as value, and a z expression evaluates a pair of real numbers, where this pair is sometimes treated as a complex value. The values of expressions and z expressions can be assigned to variables and z variables, respectively. If no value has been assigned to a variable, its default value is zero.

The following is the syntax of LCCD expressions in BNF notation.

```

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<digit string> ::= <digit> | <digit string><digit>
<letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v
<index> ::= <digit string>
<variable> ::= x<index> | y<index>          0 ≤ index ≤ 127
<constant> ::= <digit string> | <digit string>.<digit string>
<elementary function> ::= sqrt(<expression>) |
                        cos(<expression>) |
                        sin(<expression>) |
                        round(<expression>) |
                        max(<expression>, <expression>) |
                        min(<expression>, <expression>) |
                        seg(<expression>, <expression>, <expression>) |
                        px(<z expression>) |
                        py(<z expression>) |
                        abs(<z expression>) |
                        arg(<z expression>)
<primary> ::= <variable> | <constant> | nrand | <elementary function> | (<expression>)
<term> ::= <primary> | <term> * <primary> | <term> / <primary>
<right part> ::= <term> | +<term> | -<term> |
                <right part> + <term> | <right part> - <term>
<expression> ::= <right part> | <variable> ← <expression>

```

Here variable x_j is the real part of the z variable z_j , for $0 \leq j \leq 127$, and variable y_j is the corresponding imaginary part. A constant is, of course, treated as a decimal number in the ordinary way.

In the following we shall use e to denote an expression, and ze for a z expression.

$\text{sqrt}(e)$ denotes the square root of the value of e ;

$\cos(e)$ denotes the cosine of the value of e in degrees, $-360^\circ < e \leq 360^\circ$;

$\sin(e)$ denotes the sine of the value of e in degrees, $-360^\circ < e \leq 360^\circ$;

$\text{round}(e)$ denotes the value of e rounded to the nearest integer, so that,
e.g., $\text{round}(1.5) = 2.0$, $\text{round}(-1.5) = -1.0$, $\text{round}(3.14) = 3.0$;

$\text{max}(e_1, e_2)$ denotes the maximum of the values e_1 and e_2 , so that, e.g.,
 $\text{max}(3.5, 2.8) = 3.5$;

$\text{min}(e_1, e_2)$ denotes the minimum of the values e_1 and e_2 , so that, e.g.,
 $\text{min}(3.5, 2.8) = 2.8$;

$\text{seg}(e_1, e_2, e_3) = e_2 + e_1(e_3 - e_2)$, i.e., e_1 of the way from e_2 to e_3 ;

$\text{px}(ze)$ denotes the real part of a complex value ze : $\text{px}(x + i y) = x$;

$\text{py}(ze)$ denotes the imaginary part of a complex value ze : $\text{py}(x + i y) = y$;

$\text{abs}(ze)$ denotes the norm of a complex value ze , i.e., $\text{abs}(x + i y) = \sqrt{x^2 + y^2}$;

$\text{arg}(ze)$ denotes the argument (in degrees) of a complex value ze , so that

$$ze = \text{abs}(ze)e^{i \text{arg}(ze)^\circ};$$

nrand denotes a random real number having the normal distribution, with mean 0 and standard deviation 1;

$*$, $/$, $+$, and $-$ denote multiplication, division, addition, and subtraction, respectively;

$\leftarrow$ denotes assignment.

The syntax for z expressions is similar:

$\langle z \text{ variable} \rangle ::= z(\text{index}) \quad 0 \leq \text{index} \leq 127$
 $\langle z \text{ elementary function} \rangle ::= \text{max}(\langle z \text{ expression} \rangle, \langle z \text{ expression} \rangle) \mid$
 $\quad \text{min}(\langle z \text{ expression} \rangle, \langle z \text{ expression} \rangle) \mid$
 $\quad \text{comb}(\langle \text{expression} \rangle, \langle \text{expression} \rangle) \mid$
 $\quad \text{mgz}(\langle \text{expression} \rangle, \langle \text{expression} \rangle) \mid$
 $\quad \text{seg}(\langle \text{expression} \rangle, \langle z \text{ expression} \rangle, \langle z \text{ expression} \rangle) \mid$
 $\quad \text{cross}(\langle z \text{ expression} \rangle, \langle z \text{ expression} \rangle, \langle z \text{ expression} \rangle, \langle z \text{ expression} \rangle)$
 $\langle z \text{ primary} \rangle ::= \langle z \text{ variable} \rangle \mid \langle z \text{ elementary function} \rangle \mid (\langle z \text{ expression} \rangle)$

$\langle z \text{ term} \rangle ::= \langle z \text{ primary} \rangle \mid \langle z \text{ term} \rangle * \langle \text{primary} \rangle \mid \langle z \text{ term} \rangle / \langle \text{primary} \rangle$
 $\langle z \text{ right part} \rangle ::= \langle z \text{ term} \rangle \mid + \langle z \text{ term} \rangle \mid - \langle z \text{ term} \rangle \mid$
 $\quad \langle z \text{ right part} \rangle + \langle z \text{ term} \rangle \mid \langle z \text{ right part} \rangle - \langle z \text{ term} \rangle$
 $\langle z \text{ expression} \rangle ::= \langle z \text{ right part} \rangle \mid \langle z \text{ variable} \rangle \leftarrow \langle z \text{ expression} \rangle$

Suppose that $ze = e_1 + i e_2$ and $zé = \acute{e}_1 + i \acute{e}_2$; then

$$\begin{aligned}\max(ze, z\acute{e}) &= \max(e_1, \acute{e}_1) + i \max(e_2, \acute{e}_2), \\ \min(ze, z\acute{e}) &= \min(e_1, \acute{e}_1) + i \min(e_2, \acute{e}_2).\end{aligned}$$

Furthermore $\text{comb}(e_1, e_2) = e_1 + i e_2$, i.e.,

$$\begin{aligned}\text{comb}(\text{px}(ze), \text{py}(ze)) &= ze, \\ \text{px}(\text{comb}(e_1, e_2)) &= e_1, \\ \text{py}(\text{comb}(e_1, e_2)) &= e_2;\end{aligned}$$

and $\text{mgz}(e_1, e_2) = e_1 \cos(e_2) + i e_1 \sin(e_2)$. In this last formula e_1 has to be nonnegative, and e_2 is the argument in degrees. Therefore, mgz is used to transform complex from norm-argument notation to rectangular notation; abs and arg do the transform inversely, i.e.,

$$\begin{aligned}\text{mgz}(\text{abs}(ze), \text{arg}(ze)) &= ze, \\ \text{abs}(\text{mgz}(e_1, e_2)) &= e_1, \\ \text{arg}(\text{mgz}(e_1, e_2)) &= e_2.\end{aligned}$$

Finally $\text{seg}(e, ze, z\acute{e}) = ze + e(z\acute{e} - ze)$, which is the point obtained by starting at ze and going e of the distance from ze to $zé$; and $\text{cross}(ze_1, ze_2, ze_3, ze_4)$ is the intersection point of segments $\overline{ze_1 ze_2}$ and $\overline{ze_3 ze_4}$, if it exists, otherwise $\text{cross}(ze_1, ze_2, ze_3, ze_4) = (0, 0)$.

For the multiplication and division in a $\langle z \text{ term} \rangle$, the multiplier and the divisor must be real numbers; but addition, subtraction, and assignment are vector operations.

4. Curves, pens and erasers

As in METAFONT, a path consists of cubic spline curves, and a pen or eraser moves along the path to draw a picture.

In the language of complex variables, suppose that there are two points z_0 and z_1 , each one associated with a direction. The direction at z_0 makes an angle θ and the direction at z_1 makes an angle ϕ with respect to the straight line from z_0 to z_1 . The direction vectors can be normalized as

$$\begin{aligned}\delta_0 &= e^{i\theta}(z_1 - z_0), \\ \delta_1 &= e^{-i\phi}(z_1 - z_0);\end{aligned}$$

and the cubic spline curve can be defined by the formula

$$z(t) = z_0 + (3t^2 - 2t^3)(z_1 - z_0) + rt(1-t)^2\delta_0 - st^2(1-t)\delta_1, \text{ for } 0 \leq t \leq 1.$$

Fig. 4-1 Curves leaving point 0 at various multiples of 10° from the horizontal, and entering point 1 at an angle of 30° .

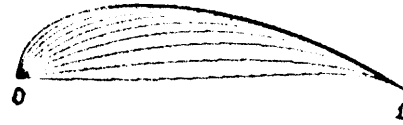
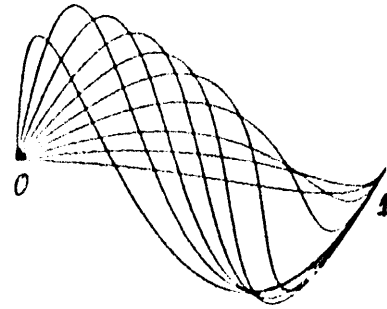


Fig. 4-2 Curves leaving point 0 at various multiples of 10° from the horizontal, and entering point 1 at an angle of -60° .



Notice that

$$\begin{aligned}z(0) &= z_0, & z'(0) &= r\delta_0, \\ z(1) &= z_1, & z'(1) &= s\delta_1.\end{aligned}$$

The numbers r and s are positive and called "velocities" at z_0 and z_1 ; they are evaluated by the formulas

$$r = \left| \frac{2 \sin \phi}{(1 + |\cos \psi|) \sin \psi} \right|, \quad s = \left| \frac{2 \sin \theta}{(1 + |\cos \psi|) \sin \psi} \right|$$

where

$$\psi = \frac{(\theta + \phi)}{2},$$

provided that $\psi \neq 0$; otherwise $r = s = 2$.

These velocity formulas have been chosen so that excellent approximations to circles and ellipses are obtained in the case $\theta = \phi$ and $\theta + \phi = \pi/2$. Furthermore, if θ and ϕ are nonnegative, the curve from z_0 to z_1 will lie entirely between or on the lines $z_0 + t\delta_0$ and $z_0 + t(z_1 - z_0)$ and entirely between or on the lines $z_1 - t\delta_1$ and $z_1 - t(z_1 - z_0)$, for $t \geq 0$. (See Figure 4-3.)

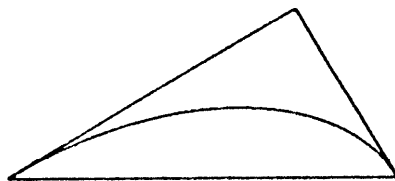


Fig. 4-3 LCCD curves are bounded by triangles.

Seven pens and seven erasers can be used in LCCD:

```

⟨w variable⟩ ::= w0 | w1 | w2 | w3 | w4 | w5 | w6 | w7 |
                w8 | w9 | w10 | w11 | w12 | w13 | w14 | w15
⟨pen or eraser⟩ ::= cpen(⟨w variable⟩) |
                    hpen(⟨w variable⟩, ⟨w variable⟩) |
                    vpen(⟨w variable⟩, ⟨w variable⟩) |
                    spen(⟨w variable⟩, ⟨w variable⟩, ⟨expression⟩) |
                    teardrop(⟨w variable⟩, ⟨expression⟩, ⟨expression⟩) |
                    lteardrop(⟨w variable⟩, ⟨expression⟩, ⟨expression⟩) |
                    rteardrop(⟨w variable⟩, ⟨expression⟩, ⟨expression⟩) |
                    cpen(⟨w variable⟩)# |
                    hpen(⟨w variable⟩, ⟨w variable⟩)# |
                    vpen(⟨w variable⟩, ⟨w variable⟩)# |
                    spen(⟨w variable⟩, ⟨w variable⟩, ⟨expression⟩)# |
                    teardrop(⟨w variable⟩, ⟨expression⟩, ⟨expression⟩)# |
                    lteardrop(⟨w variable⟩, ⟨expression⟩, ⟨expression⟩)# |
                    rteardrop(⟨w variable⟩, ⟨expression⟩, ⟨expression⟩)#

```

The sixteen w variables are system variables that can be assigned values by the command

⟨w variable⟩ ⟨expression⟩;

If no value is assigned to a w variable, it is 0 by default.

If a “#” appears at the ending of a pen definition, the pen becomes an eraser. Pens and erasers have the same shape, the difference is color: a pen is black and used to fill, while an eraser is white and used to cover up black marks.

$\text{cpen}(w_i)$, "circular pen", is a circle with w_i as radius.

$\text{hpen}(w_i, w_j)$, "horizontal pen", is an ellipse of width $2w_i$ and height $2w_j$ at the extremes of a path; it is a horizontal segment of width $2w_i$ in the middle of a path.

$\text{vpen}(w_i, w_j)$, "vertical pen", is an ellipse of width $2w_i$ and height $2w_j$ at the extremes of a path, but it is a vertical segment of height $2w_j$ in the middle of a path.

$\text{spen}(w_i, w_j, \theta)$, "special pen", is an ellipse of width $2w_i$, height $2w_j$, and rotated an angle θ counterclockwise.

$\text{teardrop}(w_i, e_1, e_2)$, "tear drop".

$\text{lteardrop}(w_i, e_1, e_2)$, "left tear drop".

$\text{rteardrop}(w_i, e_1, e_2)$, "right tear drop".

} (see below)

The three teardrops have a tear drop shape at the extremes of a path, but they behave like a $\text{cpen}(w_i)$ in the middle of the path. A tear drop consists of a circle and an isosceles triangle whose two sides are tangent to the circle. The vector that starts from the circle center and ends at the other vertex of the isosceles triangle is called the tear drop's direction. If the length of this vector is l , then $w_i:l = e_1:e_2$.

The three tear drops have different directions, depending on the curve being drawn: lteardrop and rteardrop decide their direction such that one of the sides of the isosceles triangle is parallel to the path direction at circle center, while the direction of a teardrop is the same as the path direction at the circle's center.

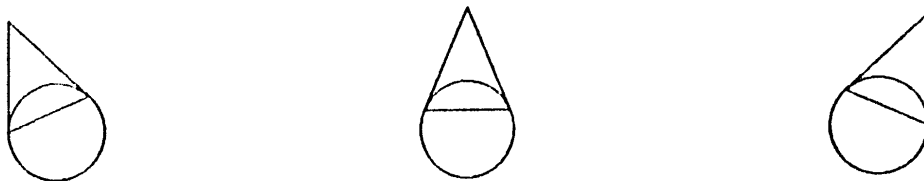


Fig. 4-4 lteardrop , teardrop , and $\text{rteardrop}(50, 1, 2.5)$, when the path direction is 90° from horizontal.

5. Draw commands

The basic means to generate strokes are 'draw' commands, described by the following syntax:

```
<prime path> ::= <point> | <prime path> .. <point>
<path> ::= <prime path> | <hiding head><prime path><hiding end>
<hiding head> ::= <empty> | (<point> .. )
<hiding end> ::= <empty> | ( .. <point> )
<point> ::= <width><index><direction>       $0 \leq index \leq 63$ 
<width> ::= <empty> | |<expression>| |<expression>#|
<direction> ::= <empty> | |<expression>, <expression>|
<draw command> ::= draw <path> | fill <path> | clean <path>
```

Three kinds of draw commands are possible:

`draw <path>` makes the current pen move along the path and generate the stroke as explained in Chapter 4. If an eraser was used instead of a pen, the effect is to erase a stroke.

`fill <path>` fills the region contoured by the specified path, which must be a closed curve.

`clean <path>` erases the region contoured by the specified path, which must be a closed curve.

In all three cases the path is a smooth curve determined by a series of points, and the index of each point must be less than or equal to 63. For each point, the direction and relative width may be given or may be defaulted. But the width information has no effect in the cases of fill and clean.

If the direction information of some point z_i has been given as $[p, q]$, then the path curve direction at this point is a vector that starts at z_i and goes p units horizontally, q units vertically. In the case that direction information is defaulted, the path curve direction will be decided in the same way as in METAFONT:

Suppose the prime path points are $z_1, \dots, z_n$, the hiding head is z_0 , and hiding end is z_{n+1} . (LCCD will set $z_0 = z_1$ if no hidden point is given at the beginning, and $z_{n+1} = z_n$ if no hidden point is given at the end; thus, each point of the curve has a predecessor and a successor.)

For any three points z_{k-1} , z_k and z_{k+1} , where $1 \leq k \leq n$, if the direction information of z_k has not been explicitly given, then if $z_{k-1} = z_k$, it is the direction from z_k to z_{k+1} ; if $z_k = z_{k+1}$, it is the direction from z_{k-1} to z_k ; otherwise it is the direction of the circle through z_{k-1} , z_k , and z_{k+1} , unless these

points are collinear. In the collinear case, it is the direction of a straight line from z_{k-1} to z_k to z_{k+1} .

Thus, hidden points are used just to help decide the direction. Notice that when $n = 1$ the direction doesn't make any sense, except for pens of tear drop shape.

The width information of the hidden head is set to "1#" when it is defaulted; if any of the other widths are defaulted, the width information will be set the same as the preceding one.

Width information is used to determine the relative size of pen or eraser at the point, to control the thickness of stroke. Actually, when a draw command is effected, the pen size varies according to a cubic spline function while the pen or eraser is moving along the path curve.

Suppose for points $z_0, z_1, \dots, z_{n+1}$, the relative widths have been determined as $s_0, s_1, \dots, s_{n+1}$. First the derivatives $(s'_1, s'_2, \dots, s'_n)$, which express the rates of change in pen relative size as the curve passes points $(z_1, z_2, \dots, z_n)$, are computed as follows:

$$\text{Let } \Delta s_j = s_{j+1} - s_j, \quad \Delta z_j = z_{j+1} - z_j.$$

1°. If a "#" appears in the width information of point z_j , then let $s'_j = 0$, (i.e., the width is stable at z_j).

2°. Otherwise if $\Delta z_{j-1} = 0$, then $s'_j = \Delta s_j$.

3°. Otherwise if $\Delta z_j = 0$, then $s'_j = \Delta s_{j-1}$.

4°. Otherwise $s'_j = \left(\frac{\Delta s_{j-1}}{|\Delta z_{j-1}|^2} + \frac{\Delta s_j}{|\Delta z_j|^2} \right) / \left(\frac{1}{|\Delta z_{j-1}|^2} + \frac{1}{|\Delta z_j|^2} \right)$.

Then, for each pair of points z_j and z_{j+1} , there is a relative width function

$$s_j(t) = s_j + (3t^2 - 2t^3)\Delta s_j + t(1-t)^2 s'_j - t^2(1-t)s'_{j+1};$$

$$0 \leq t \leq 1, \quad j = 1, \dots, n-1.$$

Notice that

$$\begin{aligned} s_j(0) &= s_j, & s'_j(0) &= s'_j; \\ s_j(1) &= s_{j+1}, & s'_j(1) &= s'_{j+1}. \end{aligned}$$

If the current pen is $\text{cpen}(w_k)$, then in fact the pen to be used is a continuous one, i.e., $\text{cpen}(w_k s_j(t))$, for t from 0 to 1, when the path curve goes from z_j to

z_{j+1} . In the other cases the parameters vary as shown in the following table:

current pen type	continuous pen used
$\text{hpen}(w_i, w_k)$	$\text{hpen}(w_i s_j(t), w_k)$
$\text{vpen}(w_i, w_k)$	$\text{vpen}(w_i, w_k s_j(t))$
$\text{spen}(w_i, w_k, \theta)$	$\text{spen}(w_i s_j(t), w_k s_j(t), \theta)$
$\text{teardrop}(w_i, p, q)$	$\text{teardrop}(w_i s_j(t), p, q)$
$\text{lteardrop}(w_i, p, q)$	$\text{lteardrop}(w_i s_j(t), p, q)$
$\text{rteardrop}(w_i, p, q)$	$\text{rteardrop}(w_i s_j(t), p, q)$

6. Example of the eight diagrams surrounding the cosmological scheme

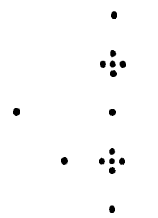
Now we are able to look at a drawing example in detail: a “BaGua TaiJiTu.” This picture makes a good example because it is not so complex that it is too hard to explain, neither is it so simple that there is nothing interesting to discuss. Although it had a mysterious meaning in ancient Chinese philosophy, it is actually a notation of binary and octal numbers, invented several thousand years ago: one thousand years for the cosmological scheme symbol, and three thousand years for the eight diagrams.

We shall use subroutines to specify its drawing procedure; the texts between “ and ” are comments to explain the drawing procedure.

subroutine tjt:

 x0+y0+100;

 complex z1+z0+mgz(50,0),
 z2+z0+mgz(50,90),
 z3+z0+mgz(50,180),
 z4+z0+mgz(50,270),
 z10+seg(0.5,z0,z2),
 z11+seg(0.5,z0,z4),
 z8+z10+mgz(25,0),
 z9+z11+mgz(25,180),
 z12+z10+mgz(5,0),
 z13+z10+mgz(5,90),
 z14+z10+mgz(5,180),
 z15+z10+mgz(5,270),
 z16+z11+mgz(5,0),
 z17+z11+mgz(5,90),
 z18+z11+mgz(5,180),
 z19+z11+mgz(5,270);



“A group of points is defined, as shown at the right.” ;

fill 4[-1,0]..3[0,1]..2[1,0]..8[0,-1]..0[-1,0]..9[0,-1]..4[1,0];

“This command fills a region as shown in Fig. 6-1.” ;

draw |0|4[1,0]..1[0,1]..2[-1,0];

“Draw a half circle; we obtain Fig. 6-2, which looks strangely like a SAFEWAY trademark in mirror image!” ;

```
fill 16[0,1]..17[-1,0]..18[0,-1]..19[1,0]..16[0,1];
```

"A small circle is filled, now Fig. 6-3 is obtained.";

```
clean 12[0,1]..13[-1,0]..14[0,-1]..15[1,0]..12[0,1].
```

"A small circle is erased; we obtain Fig. 6-4, the so-called Tai Ji Tu (diagram of cosmological scheme). It denotes the unification of two contrary forces or spirits, e.g., positive vs. negative, male vs. female, active vs. passive, brightness vs. darkness, etc. It played a very important role in ancient Chinese philosophy. The dark part is called Yin, which represents female; the white part is called Yang, which represents male.";



Fig. 6-1



Fig. 6-2

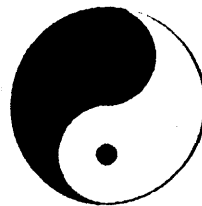


Fig. 6-3

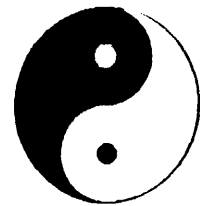


Fig. 6-4

subroutine zero:

```
y0+y1+40;
x10+60*sin(17)/cos(17);
x0+100+x10;
x1+100-x10;
draw 0..1.
```



"This subroutine defines a bar as shown. It was called Yang Yao, but actually it represents a binary zero.";

subroutine one:

```
y0+y1+y2+y3+40;  
x10+60*sin(3)/cos(3) ;  
x11+60*sin(17)/cos(17);  
x0+100+x10; x2+100-x10;  
x1+100+x11; x3+100-x11;  
draw 0..1; draw 2..3.
```



"This subroutine defines two short bars as shown. It was called Yin Yao, but actually it represents a binary one.";

subroutine g0:

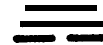
```
call zero;  
call zero(0, 1.15, 1.15, 0, 0);  
call zero(0, 1.25, 1.25, 0, 0).
```



"So called Qian Gua, or octal zero. Enlargement by factors of 1.15 and 1.25 with respect to the center point (100,100) causes the strokes to be of unequal length and positioned above each other." ;

subroutine g1:

```
call zero;  
call zero(0, 1.15, 1.15, 0, 0);  
call one(0, 1.25, 1.25, 0, 0).
```



"So called Dui Gua, or octal one." ;

subroutine g2 :

```
call zero;  
call one(0, 1.15, 1.15, 0, 0);  
call zero(0, 1.25, 1.25, 0, 0).
```



"So called Li Gua, or octal two." ;

subroutine g3:

```
call zero;  
call one(0, 1.15, 1.15, 0, 0);  
call one(0, 1.25, 1.25, 0, 0).
```

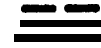


"So called Zhen Gua, or octal three." ;

```

subroutine g4 :
  call one;
  call zero(0, 1.15, 1.15, 0, 0);
  call zero(0, 1.25, 1.25, 0, 0).
  "So called Xun Gua, or octal four." ;

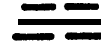
```



```

subroutine g5:
  call one;
  call zero(0, 1.15, 1.15, 0, 0);
  call one(0, 1.25, 1.25, 0, 0).
  "So called Kan Gua, or octal five." ;

```



```

subroutine g6:
  call one;
  call one(0, 1.15, 1.15, 0, 0);
  call zero(0, 1.25, 1.25, 0, 0).
  "So called Gen Gua, or octal six." ;

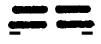
```



```

subroutine g7 :
  call one;
  call one(0, 1.15, 1.15, 0, 0);
  call one(0, 1.25, 1.25, 0, 0).
  "So called Kun Gua, or octal seven." ;

```



"The eight kinds of Guas are called the eight diagrams, first recorded in YiJing (the Book of Changes).";

```

subroutine bgtj:
  call tjt;
  call g0 ( 0 , 1.01 , 1.01 , 0 , 0 ) ;
  call g1 ( 45 , 1.01 , 1.01 , 0 , 0 ) ;
  call g2 ( 90 , 1.01 , 1.01 , 0 , 0 ) ;
  call g3 ( 135 , 1.01 , 1.01 , 0 , 0 ) ;
  call g4 ( -45 , 1.01 , 1.01 , 0 , 0 ) ;
  call g5 ( -90 , 1.01 , 1.01 , 0 , 0 ) ;
  call g6 ( -135 , 1.01 , 1.01 , 0 , 0 ) ;
  call g7 ( 180 , 1.01 , 1.01 , 0 , 0 ) .

```

"This subroutine defines the shape of eight diagrams combined with the diagram of the cosmological scheme."

After having these 12 subroutine definitions, it is very easy to draw the final diagram just by executing the statements

```
w0 2; cpen(w0); call bgtj .
```

The result will be Fig. 6-5.

Fig. 6-5 The eight diagrams surrounding the cosmological scheme.



7. Discreteness

Naturally we cannot draw perfectly with a varying size pen, since a continuous curve must be rounded to a discrete raster. To get the best possible effects, the rounding time (binding time) should be postponed as much as possible.

In LCCD, the problem of drawing is converted to a sequence of tasks: to find the envelope of the pen shape, to discretize this envelope curve, and then to fill or erase the region bounded by the discrete grid points. Compared to another possible method, in which a discrete pen shape moves along a discrete path curve, this method has just one step for rounding, therefore it is the more precise one.

The first problem is to find the envelope of a class of curves that depends on one parameter. Suppose that the equation

$$\phi(x, y, t) = 0 \quad (1)$$

defines a class of curves, where ϕ is a continuous differentiable function of its all variables such that

$$\phi_x^2 + \phi_y^2 \neq 0 \quad (2)$$

for all (x, y, t) satisfying (1). Then, according to well known principles of differential geometry, the envelope of this class (if it exists) is given by the equations

$$\begin{aligned} \phi(x, y, t) &= 0, \\ \phi_t(x, y, t) &= 0. \end{aligned} \quad (3)$$

In other words, for every point (x, y) on the envelope, there is a t , such that $\phi = 0$ and $\phi_t = 0$ are both satisfied for the values (x, y, t) .

Now the case of $\text{spen}(w, h, \theta)$ shall be considered, supposing that the pen size does not vary. The ellipse equation is

$$ax^2 + bxy + cy^2 - 1 = 0, \quad (4)$$

where

$$\begin{aligned} a &= \frac{\cos^2 \theta}{w^2} + \frac{\sin^2 \theta}{h^2}, \\ b &= \left(\frac{1}{w^2} - \frac{1}{h^2} \right) \sin 2\theta, \\ c &= \frac{\sin^2 \theta}{w^2} + \frac{\cos^2 \theta}{h^2}. \end{aligned} \quad (5)$$

The problem is to find the envelope for the class of such ellipses with center motion along the curve $(X(t), Y(t))$. Thus the center point (x, y) of ellipse (4) has to be

replaced by $(x - X(t), y - Y(t))$; then a class of ellipses is obtained. Notice that we have $b^2 \neq 4ac$, from which it can be verified that condition (2) always holds. Therefore finding the envelope is equivalent to finding (x, y) satisfying (3).

Formulas (4) and (5) can be rewritten as

$$\frac{1}{w^2}(x \cos \theta + y \sin \theta)^2 + \frac{1}{h^2}(-x \sin \theta + y \cos \theta)^2 = 1,$$

so the problem reduces to solving the system of equations

$$\begin{aligned} & \frac{1}{w^2}((x - X(t)) \cos \theta + (y - Y(t)) \sin \theta)^2 \\ & \quad + \frac{1}{h^2}(-(x - X(t)) \sin \theta + (y - Y(t)) \cos \theta)^2 = 1, \\ & \frac{1}{w^2}((x - X(t)) \cos \theta + (y - Y(t)) \sin \theta)(-X'(t) \cos \theta - Y'(t) \sin \theta) \\ & \quad + \frac{1}{h^2}(-(x - X(t)) \sin \theta + (y - Y(t)) \cos \theta)(X'(t) \sin \theta - Y'(t) \cos \theta) = 0. \end{aligned} \tag{6}$$

Suppose that

$$\begin{aligned} x - X(t) &= pw \cos \theta - qh \sin \theta, \\ y - Y(t) &= pw \sin \theta + qh \cos \theta. \end{aligned} \tag{7}$$

Substituting into (6), we obtain

$$\begin{aligned} & p^2 + q^2 = 1, \\ & -\frac{p}{w}(X'(t) \cos \theta + Y'(t) \sin \theta) + \frac{q}{h}(X'(t) \sin \theta - Y'(t) \cos \theta) = 0. \end{aligned}$$

Let

$$\begin{aligned} \alpha &= \frac{1}{w}(X'(t) \cos \theta + Y'(t) \sin \theta), \\ \beta &= \frac{1}{h}(X'(t) \sin \theta - Y'(t) \cos \theta). \end{aligned} \tag{8}$$

Then we have $p^2 + q^2 = 1$ and $\alpha p = \beta q$, and there are two solutions,

$$\begin{aligned} p &= \frac{\pm \beta}{\sqrt{\alpha^2 + \beta^2}}, \\ q &= \frac{\pm \alpha}{\sqrt{\alpha^2 + \beta^2}}. \end{aligned} \tag{9}$$

Substituting this in (7) leads to

$$\begin{aligned} x &= X(t) \pm \frac{1}{\sqrt{\alpha^2 + \beta^2}} (\beta w \cos \theta - \alpha h \sin \theta), \\ y &= Y(t) \pm \frac{1}{\sqrt{\alpha^2 + \beta^2}} (\beta w \sin \theta + \alpha h \cos \theta). \end{aligned} \quad (10)$$

It is not difficult to deduce from (8) that

$$\begin{aligned} \beta w \cos \theta - \alpha h \sin \theta &= -wh(\tfrac{1}{2}bX'(t) + cY'(t)), \\ \beta w \sin \theta + \alpha h \cos \theta &= wh(aX'(t) + \tfrac{1}{2}bY'(t)), \\ \alpha^2 + \beta^2 &= aX'^2(t) + bX'(t)Y'(t) + cY'^2(t); \end{aligned}$$

so we obtain the following equations for the two envelopes:

$$\begin{aligned} x &= X(t) \mp \frac{wh(\tfrac{1}{2}bX'(t) + cY'(t))}{\sqrt{aX'^2(t) + bX'(t)Y'(t) + cY'^2(t)}}, \\ y &= Y(t) \pm \frac{wh(aX'(t) + \tfrac{1}{2}bY'(t))}{\sqrt{aX'^2(t) + bX'(t)Y'(t) + cY'^2(t)}}. \end{aligned} \quad (11)$$

For cpen with a varying size, let the radius be $w(t)$ while its center moves along the path curve $(X(t), Y(t))$. We can find its envelope in a similar fashion, obtaining

$$\begin{aligned} x &= X(t) - w(t) \frac{X'(t)w'(t) \pm Y'(t)\sqrt{X'^2(t) + Y'^2(t) - w'^2(t)}}{X'^2(t) + Y'^2(t)}, \\ y &= Y(t) - w(t) \frac{Y'(t)w'(t) \mp X'(t)\sqrt{X'^2(t) + Y'^2(t) - w'^2(t)}}{X'^2(t) + Y'^2(t)}. \end{aligned} \quad (12)$$

Here the condition

$$X'^2(t) + Y'^2(t) \geq w'^2(t) \quad (13)$$

has to be satisfied, i.e., the velocity of the path curve has to be greater than or equal to the velocity of radius variation. Unfortunately this condition cannot be guaranteed to hold in all cases.

The envelope of an spen in the case of variable pen size seems much more difficult, and the author believes that it is not an elementary function. Furthermore its existence must depend on some condition about the velocity of the path and the change of the width, because cpen is one of its special examples.

So, when the pen size is changeable, LCCD does not use the true envelope; it simply uses equation (11) with w replaced by $ws(t)$, where $s(t)$ is the relative width function described in Chapter 5. A cpen(w) and a teardrop pen of width w are treated as an spen($w, w, 0$).

The formulas used for hpen and vpen are even simpler; the envelope of an hpen(w, h) is given by

$$\begin{aligned} x &= X(t) \pm ws(t), \\ y &= Y(t), \end{aligned} \tag{14}$$

and the analogous formulas for vpen(w, h) are

$$\begin{aligned} x &= X(t), \\ y &= Y(t) \pm hs(t). \end{aligned} \tag{15}$$

Approximate computing of curves. The continuous envelopes defined above need to be rounded to discrete grid points, and the same is true of the curves that bound the pen at the two extremities of a stroke. The algorithm used by LCCD for this purpose is called the "cross corridor rule."

Suppose that a curve

$$z(t) = x(t) + iy(t), \quad 0 \leq t \leq 1,$$

is given. We want to find a sequence of rational numbers

$$0 = t_0 < t_1 < \dots < t_n = 1$$

and a sequence of integer pairs

$$\{(p_i, q_i)\}_{i=0, 1, \dots, n},$$

such that, for some given $\epsilon > 0$, conditions 1°–4° hold.

- 1°. $|p_0 - x(t_0)| \leq \frac{1}{2}, \quad |q_0 - y(t_0)| \leq \frac{1}{2};$
 $|p_n - x(t_n)| \leq \frac{1}{2}, \quad |q_n - y(t_n)| \leq \frac{1}{2};$
- 2°. $\max(|p_i - x(t_i)|, |q_i - y(t_i)|) \leq \frac{1}{2},$
 $\min(|p_i - x(t_i)|, |q_i - y(t_i)|) \leq \epsilon, \quad i = 1, \dots, n-1;$
- 3°. $(p_i, q_i) \neq (p_{i+1}, q_{i+1}), \quad i = 0, 1, \dots, n-1;$

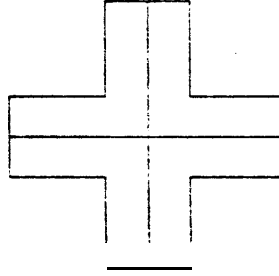
$$4^\circ. \quad \begin{aligned} |p_i - p_{i+1}| &\leq 1, \\ |q_i - q_{i+1}| &\leq 1, \end{aligned} \quad i = 0, 1, \dots, n-1.$$

Then we will use the sequence $\{(p_i, q_i)\}_{i=0,1,\dots,n}$ instead of the curve $z(t)_{0 \leq t \leq 1}$.

Condition 1° says that (p_0, q_0) and (p_n, q_n) are approximations of $(x(0), y(0))$ and $(x(1), y(1))$, respectively. We can't expect do to better than that.

Condition 2° says that (p_i, q_i) is an approximation of $(x(t_i), y(t_i))$. We can acquire a better approximation in the middle by choosing $\{t_i\}$ suitably: If the integer point (p_i, q_i) is plotted, then the curve $z(t)$ must pass through a cross region, whose center is (p_i, q_i) , width and height are 1, and corridor width is 2ϵ .

Fig. 7-1 Cross corridor



Condition 3° says that the sequence $\{t_i\}$ is sparse enough that there are not any redundant grid points in $\{(p_i, q_i)\}$.

Condition 4° says that the sequence $\{t_i\}$ is dense enough that the broken line of grid points $\{(p_i, q_i)\}$ is continuous (has no gaps).

Before solving the approximation problem, let us introduce the notation $\llbracket r, s \rrbracket$ to represent the interval $[\min(r, s), \max(r, s)]$. We shall begin by solving the following subtask:

Let $z(t)$ be a continuous function defined on the interval $\llbracket a, b \rrbracket$, and let $z_0 \in \llbracket z(a), z(b) \rrbracket$. Given $\epsilon > 0$, find $c \in \llbracket a, b \rrbracket$ such that either

$$|z_0 - z(c)| < \epsilon,$$

or

$$|s - z(c)| < \epsilon$$

for some integer q (not necessarily in $\llbracket z(a), z(b) \rrbracket$). This subtask can be solved as follows.

Algorithm of subtask: Let $c_1 = (a + b)/2$.

i) If $|z(c_1) - z_0| < \epsilon$, terminate with $c = c_1$. (Notice that this case must occur for sufficiently small intervals $\llbracket a, b \rrbracket$, because of continuity.)

ii) If the nearest integer q to $z(c_1)$ satisfies $|z(c_1) - q| < \epsilon$, terminate with $c = c_1$; again this is what we want.

iii) Otherwise we have $|z(c_0) - z_0| \geq \epsilon$ and the fractional part $\{z(c_1)\}$ lies in the interval $[\epsilon, 1 - \epsilon]$. Let $a_1 = c_1$, and if $z(c_1) > z_0$, then let

$$b_1 = \begin{cases} a, & \text{if } z(a) < z(b); \\ b, & \text{if } z(a) \geq z(b). \end{cases}$$

Notice that

$$z(c_1) > z_0 \geq \min(z(a), z(b)) = z(b_1).$$

If $z(c_1) < z_0$, then let

$$b_1 = \begin{cases} a, & \text{if } z(a) \geq z(b); \\ b, & \text{if } z(a) < z(b); \end{cases}$$

and notice that

$$z(c_1) < z_0 \leq \max(z(a), z(b)) = z(b_1).$$

Therefore, in either case, we obtain $z_0 \in \llbracket z(a_1), z(b_1) \rrbracket$, $\llbracket a_1, b_1 \rrbracket \subset \llbracket a, b \rrbracket$, and $|a_1 - b_1| = \frac{1}{2}|a - b|$. Thus, the task has been reduced to a smaller interval $\llbracket a_1, b_1 \rrbracket$. By using this procedure repeatedly, we obtain a sequence of intervals

$$\llbracket a, b \rrbracket \supset \llbracket a_1, b_1 \rrbracket \supset \cdots \supset \llbracket a_m, b_m \rrbracket \supset \cdots,$$

where

$$|a_m - b_m| = \frac{1}{2^m}|a - b|, \quad z_0 \in \llbracket z(a_m), z(b_m) \rrbracket.$$

This process must terminate when m is sufficiently great, so the algorithm will find a c fulfilling the requirements. Since the interval is cut in one half each time, this process converges rapidly. Furthermore the probability of $z(c)$ falling in a interval having an integer as center and 2ϵ as width is 2ϵ . The chance of success by the n th iteration is $1 - (1 - 2\epsilon)^n$, and this makes the convergence even faster.

In LCCD, the value of ϵ is set to 0.25, and empirical experiments show that the average number of iterations is between 1.1 and 1.7. Thus the subtask has been solved.

Notice that since $z(t)$ is uniformly continuous, there is a number δ such that the number c found the subtask algorithm satisfies

$$|c - a| \geq \delta, \quad |c - b| \geq \delta.$$

Now we describe the algorithm that is used to approximate a continuous curve. It does the approximation by interpolating, where each interpolation is reduced to the subtask described above.

Suppose we want to plot $(x(t), y(t))$ in the interval $a \leq t \leq b$, and that we are given integer pairs

$$(p_a, q_a) \neq (p_b, q_b)$$

such that

$$\begin{aligned} \max(|p_a - x(a)|, |q_a - y(a)|) &\leq \frac{1}{2}, \\ \max(|p_b - x(b)|, |q_b - y(b)|) &\leq \frac{1}{2}; \end{aligned}$$

furthermore if $a > 0$ we assume that

$$\min(|p_a - x(a)|, |q_a - y(a)|) < \epsilon,$$

and if $b < 1$ we assume that

$$\min(|p_b - x(b)|, |q_b - y(b)|) < \epsilon,$$

for some given $\epsilon \leq \frac{1}{2}$. Then the discussion can be subdivided into two cases.

Case I:

$$\begin{aligned} |p_a - p_b| &\leq 1, \\ |q_a - q_b| &\leq 1. \end{aligned} \tag{*}$$

Then segment $\overline{(p_a, q_a)(p_b, q_b)}$ is already the desired approximation of curve $z(t) = x(t) + i y(t)$ ($a \leq t \leq b$), and the treatment of interval $[a, b]$ is finished.

Note that if $x(t)$ and $y(t)$ are continuous, this case must hold whenever the endpoints a and b are sufficiently near each other that

$$|x(a) - x(b)| < 1, \quad |y(a) - y(b)| < 1.$$

For this implies that

$$\begin{aligned} |p_a - p_b| &\leq |p_a - x(a)| + |x(a) - x(b)| + |x(b) - p_b| < \frac{1}{2} + 1 + \frac{1}{2}, \\ |q_a - q_b| &\leq |q_a - y(a)| + |y(a) - y(b)| + |y(b) - q_b| < \frac{1}{2} + 1 + \frac{1}{2}. \end{aligned}$$

Relation (*) must now hold because p_a, q_a, p_b , and q_b are integers.

Case II: If (*) is false, we may suppose by symmetry that $|p_a - p_b| \leq |q_a - q_b|$ and $|q_a - q_b| > 1$. Since q_a and q_b are integers, we have $|q_a - q_b| \geq 2$.

Let

$$q_c = \left\lfloor \frac{q_a + q_b + 1}{2} \right\rfloor,$$

i.e., the greatest integer less than or equal to $\frac{1}{2}(q_a + q_b + 1)$. Then

$$\begin{aligned} \min(q_a, q_b) + 1 &= \frac{2 \min(q_a, q_b) + 2}{2} < \frac{q_a + q_b + 1}{2} \\ &= \frac{\max(q_a, q_b) + \min(q_a, q_b) + 1}{2} \leq \frac{2 \max(q_a, q_b) - 1}{2} < \max(q_a, q_b), \end{aligned}$$

so

$$\min(q_a, q_b) + 1 \leq q_c \leq \max(q_a, q_b) - 1;$$

in other words $q_c \neq q_a$ and $q_c \neq q_b$.

Also notice that $y(a) - \frac{1}{2} \leq q_a \leq y(a) + \frac{1}{2}$, and $y(b) - \frac{1}{2} \leq q_b \leq y(b) + \frac{1}{2}$; therefore

$$\begin{aligned} \min(y(a), y(b)) + \frac{1}{2} &\leq \min(q_a, q_b) + 1 \leq q_c \\ &\leq \max(q_a, q_b) - 1 \leq \max(y(a), y(b)) - \frac{1}{2}, \end{aligned}$$

and we have $q_c \in \llbracket y(a), y(b) \rrbracket$, $q_c \neq y(a)$, $q_c \neq y(b)$.

The subtask algorithm with $z_0 = q_c$ will find $c \in \llbracket a, b \rrbracket = [a, b]$ such that $|q - y(c)| < \epsilon$ for some integer q . Furthermore there is a positive number δ such that

$$a + \delta \leq c \leq b - \delta.$$

Let p_c be the integer nearest to $x(c)$. Then

$$\begin{aligned} \max(|p_c - x(c)|, |a - y(c)|) &\leq \max(\frac{\epsilon}{2}, \epsilon) = \frac{\epsilon}{2}, \\ \min(|p_c - x(c)|, |q - y(c)|) &\leq |q - y(c)| \leq \epsilon. \end{aligned}$$

Condition 2° is satisfied.

The task at interval $[a, b]$ can now be reduced into two subtasks at smaller subintervals $[a, c]$ and $[c, b]$, respectively. All of the data for the second subinterval $[c, b]$, and its corresponding p and q , will be pushed down into a stack. Then the first subinterval $[a, c]$, which is a smaller one compared to the original interval $[a, b]$, will be treated by the same procedure. Whenever an interval is subdivided into two smaller ones, we always push the second onto a stack, then treat the first. Thus, the current interval to be treated becomes smaller and smaller. If an interval $[a, b]$ to be treated is a degenerate one, i.e., $p_a = p_b$ and $q_a = q_b$, it is simply deleted; this will ensure condition 3°, and the curves being plotted should be sufficiently well behaved that no "loops" are being lost in this way. We know from the discussion in Case I that the treatment of the current interval must eventually be finished. Once an interval is finished, we pop up the stack and treat the top interval, until the stack is empty. Eventually this process must

terminate, since the sum of the length of intervals to be treated decreases by at least δ whenever an element is popped off, and the length of the current interval decreases by at least δ whenever an element is pushed on.

In the beginning, let $[a, b] = [0, 1]$, and let p_a, q_a, p_b , and q_b be the integers nearest to $x(0)$, $y(0)$, $x(1)$, and $y(1)$, respectively. Condition 1° clearly holds. Also, the curve to be treated should be non-closed and non-degenerate; therefore $(p_0, q_0) \neq (p_n, q_n)$, and all of the prerequisite conditions are satisfied.

8. Transformation in subroutine definitions and subroutine calls

Chapter 6 showed by example that a drawing process can be defined as a subroutine. The following things are permitted in an LCCD subroutine: to define a group of points, to specify the pens or erasers to be used, to draw or fill or erase, and to call other subroutines. An "adjust command" might also be put at the end of a subroutine definition.

Adjustment might be automatic: in this case, a transformation determined by the system becomes a portion of the subroutine definition, such that under this transformation, the containing box (the minimum rectangle containing the figure defined by the subroutine) is precisely the "standard box", i.e., a square with point (100, 100) as center and 180 as side length.

With such an automatic transformation, the designer of a Chinese character needn't worry about computing the size of the character shape; he or she can devote attention to the relative position and proportion relations of the various components. The automatic adjustment transformation gives every character the same size.

However, because of visual psychological factors, some Chinese characters with the same containing box actually look like they have different sizes. They must be changed a little bit, such that the different-size characters appear as if their sizes are the same.

It would be interesting to discover an automatic adjustment transformation that would account for most of these subtle visual factors, but in LCCD a simple solution has been adopted, i.e., a non-automatic adjust command.

When the system scans such an adjust command in processing a subroutine definition, the figure is displayed on the screen in a standard box, then a conversation between the operator and machine starts. The system displays the following message:

Do you think It's OK now? If so, type <cr>. For help, type ?<cr>

⊗

If you are satisfied with the position and size of the character shape on the screen, type carriage return to finish processing the subroutine definition. Five numbers $(\theta, xl, yl, \Delta x, \Delta y)$ are used by LCCD to adjust the curves drawn by a subroutine: θ is the angle of rotation; xl and yl are magnifying factors at the x direction and the y direction, respectively; Δx and Δy are the shifts at these two directions. The order of transformation is first to rotate, then to magnify, then to translate. There is no rotation for an adjustment transformation, so $\theta = 0$.

If you are not satisfied for any reason, suppose it is the first time you are

using this system, so you don't know what to do. Then you might type a question mark ? followed by carriage return; the system responds immediately:

Please show me the transform or transforms you want.

↑<integer> to shift upxard.

↓<integer> to shift downward.

←<integer> to shift left.

→<integer> to shift right.

x<constant>toenlargein the X direction.

y<constant>toenlargein the Y direction.

Transformations are cumulative (e.g., x2 cancels a previous x0.5) unless precoded by comma (e.g., ,x2 magnifies by exactly 2).

⊗

Now you know how to magnify and translate the figure; suppose you type

x1.5y0.8+20↓30

followed by a carriage return. This means to magnify the present figure 1.5 times in the *x* direction, and 0.8 times (actually contracting) in the *y* direction, then translating 20 pixels rightward and 30 pixels downward. There always exists an adjust transformation in the background (at the beginning it is the identity); the system will combine the previous one and the new one, using addition for rotating angles, multiplication for magnifying times, and addition for translation.

If what you typed is

x1.5,y0.8+20↓30

then it means the previous *y* magnification factor is to be ignored; the new *y* magnification should be 0.8, unconditionally.

The transformation affects the paths of all curves that are defined in the subroutine (i.e., the skeleton of picture), but not the pen widths. The new picture and transform parameters will be displayed on the screen, and LCCD will inquire

Do you think it's OK nox? If so, type <cr>. For help,type?<cr>

⊗

again, until you respond with a carriage return denoting satisfaction. Once you have finished processing a subroutine definition, the subroutine name and adjust transformation parameters will be written onto a file called **subrtn.def**.

When designing a group of Chinese characters, it is best to begin with the automatic adjust command to make them have the same size, then to adjust

some of them by non-automatic means until the whole set of characters looks right visually.

A new character is usually built up from previously created characters or radicals. In other words, a subroutine definition usually consists of a series of other subroutine calls. The size and relative position of various components of a character is extremely important for the appearance of character shapes, and this can be decided only by experienced eyes.

In this system, each subroutine call can have an attached transformation, i.e, first rotating, then magnifying or contracting (with point (100, 100) as center), then translation; see the example in Chapter 6. When calling a subroutine that doesn't appear within another subroutine definition, its picture is displayed on the screen just as when a non-automatic adjustment transformation is being executed; then a person-machine conversation begins. The user can change parameters, until he or she feels satisfied. When the subroutine calling is finished, the final parameters are written onto file `subrtn.def`.

Thus in order to design a picture, you first type a series of subroutine calls. In executing each of them, you determine the transformation parameters of each subpicture online. When a satisfactory picture is obtained, the final form of the subroutine calls will be recorded in file `subrtn.def`, and it is very easy to edit this file and acquire a new subroutine definition. Automatic or non-automatic adjustment can be carried on further from this new definition.

When a called subroutine appears in another subroutine definition, its parameters cannot be changed online.

9. The other features of LCCD language

Many aspects of LCCD have already been described. In this section we will list the remaining ones and consider the overall organization.

An LCCD program consists of a series of sections, each of which is terminated by a period. The whole program ends with the word "stop":

```

$$\begin{aligned}\langle \text{program} \rangle &::= \langle \text{section list} \rangle \text{ stop} \\ \langle \text{section list} \rangle &::= \langle \text{section} \rangle \mid \langle \text{section list} \rangle \langle \text{section} \rangle \\ \langle \text{section} \rangle &::= \langle \text{statement list} \rangle . \mid \langle \text{subroutine definition} \rangle . \mid \langle \text{Chinese file} \rangle .\end{aligned}$$

```

There are three kinds of sections, discussed further below. The variables, z variables, pens and erasers defined in a section are local to that section; but w variables are global.

A typical LCCD program starts out with an initial section to set desired system parameter values and control modes, then a series of subroutine definitions, followed by a section that is a Chinese file to output something; or by a section that is a statement list, to try something that will define a new subroutine.

When the last word "stop" is scanned, the user can choose whether the subroutine definitions will be saved or not. If the decision is to save them, the system will remember all of the subroutines defined in this program, and they will be preloaded the next time that LCCD is run. There is a file named `ledini.tbl` in which everything the system knows is kept. The first thing the system does is to load `ledini.tbl`, and one of the last things it does is to decide whether new subroutines need to be saved or not in this file.

One kind of section is a statement list, which is a series of statements separated by semicolons ";":

```

$$\begin{aligned}\langle \text{statement list} \rangle &::= \langle \text{statement} \rangle \mid \langle \text{statement list} \rangle ; \langle \text{statement} \rangle \\ \langle \text{statement} \rangle &::= \langle \text{simple statement} \rangle \mid \langle \text{conditional statement} \rangle \\ \langle \text{conditional statement} \rangle &::= \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement list} \rangle \text{ fi} \mid \\ &\quad \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement list} \rangle \text{ else } \langle \text{statement list} \rangle \text{ fi} \\ \langle \text{condition} \rangle &::= \langle \text{control mode} \rangle \mid \langle \text{expression} \rangle \\ \langle \text{simple statement} \rangle &::= \langle \text{empty} \rangle \mid \\ &\quad \langle \text{title} \rangle \mid \\ &\quad \langle \text{expression} \rangle \mid \\ &\quad \text{real } \langle \text{expression list} \rangle \mid \\ &\quad \text{complex } \langle \text{z expression list} \rangle \mid \\ &\quad \langle \text{pen or eraser} \rangle \mid\end{aligned}$$

```

```

        <draw command> |
        <subroutine call> |
        <system parameter> <expression> |
        <control mode> |
        no <control mode>
<title> ::= <sequence of any characters except " and carriage return>
<expression list> ::= <expression> | <expression list>, <expression>
<z expression list> ::= <z expression> | <z expression list>, <z expression>
<subroutine call> ::= call <name> <calling transform>
<name> ::= <letter or digit> | <name> <letter or digit>
<letter or digit> ::= <letter> | w | x | y | z | <digit>
<calling transform> ::= <empty> |
        (<expression>, <expression>, <expression>, <expression>, <expression>)
<system parameter> ::= <real parameter> | <integer parameter>
<real parameter> ::= <w variable> |
        trxx |
        trxy |
        trx |
        tryx |
        tryy |
        try |
        maxvr |
        minvr |
        maxvs |
        minvs |
        stheta
<integer parameter> ::= pentype |
        seed |
        fontstyle |
        dumpwindow
<control mode> ::= titletrace |
        drawtrace |
        curtrace |
        pause |
        pagewarning |
        pensystem |
        constwidth |
        blank |
        crsmode |
        tfnmode

```

Let us describe the simple statements first.

An empty statement can be used to place semicolons or blank spaces so that user will feel free when typing.

A title is used for a message to remind the user for some reason. Under control mode *titletrace*, titles will be displayed on the screen when they are scanned.

Expressions and *z* expressions are used to define points, which are local to that section. At most 128 points can be defined in a section.

A pen or eraser defines the current pen or eraser to be used, local to that section. Notice that by subroutine calling, a new level of current pen may be introduced, and the old level will be restored after finishing the call. If the user does not say anything about pens, a *cpen(w0)* will be assumed.

A draw command relates to points defined in the current section, using the current pen or eraser, as described in Chapter 5.

By subroutine calls the user is able to build characters from previously known subcharacters. As described in the previous chapter, the calling transform just decides the initial transformation, which can be changed online. In case the calling transform is empty, it means (0, 1, 1, 0, 0), which is the identity mapping.

System parameters are global to whole program. They are subdivided into two groups, i.e., integer valued and real valued parameters. They can be assigned in any section that isn't a subroutine definition.

System *w* variables (normally 0) are used to specify the size of a pen or eraser.

The system parameters *trxx*, *trxy*, *trx*, *tryx*, *tryy*, and *try* (normally 1.0, 0.0, 0.0, 0.0, 1.0, 0.0) define a transformation that can be used to convert (*x*, *y*) to (*X*, *Y*) as follows:

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} trxx & trxy \\ tryx & tryy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} trx \\ try \end{pmatrix} \quad (*)$$

The user can consider that (*) is a final transformation applied to a picture before output. But actually (*) is used for the path curves (i.e., the skeleton of the picture), not for the envelopes.

Such transformations are generally not necessary while a character is being designed, but they can be used to enlarge or shrink characters as in the examples of Appendix 4. A picture to be output has to be able to fit in a box whose four sides are *x* = 0, *x* = 1700, *y* = 0, and *y* = 589. But for display, the four sides of the box should be *x* = 0, *x* = 200, *y* = 0, and *y* = 200.

System parameters *maxvr*, *minvr*, *maxvs*, *minvs* (normally 4.0, 0.5, 4.0, 0.5) are used as velocity thresholds when computing the path spline curves, just as in METAFONT.

A control mode *pensystem* can be used to override the pen or eraser specifications that actually appear; in such a case the value of *pentype* (normally 1) specifies what kind of pen will be used, according to the following code:

The value of <i>pentype</i>	Pen or eraser to be used
1	<i>cpen</i> (w_{11})
2	<i>hpen</i> (w_{12}, w_{13})
3	<i>vpen</i> (w_{13}, w_{12})
4	<i>spen</i> ($w_{14}, w_{15}, \textit{stheta}$)

When the original specification calls for an eraser instead of a pen, the new one will be an eraser. System parameter *stheta* (normally 135.0) denotes the angle of the *spen* that is used when *pentype* = 4.

The *seed* parameter (normally set to a value based on the time of day so that it will be different every time you run LCCD) is used to start the pseudo-random number generator that produces the values of *nrand*. It can be used to guarantee that you acquire the same sequence of *nrand* values.

Parameter *fontstyle* (normally 0) is used to control font style. If *fontstyle* is 1, the two extremes of a stroke won't be plotted. The value of *fontstyle* can also be tested using an expression as a condition between the words "if" and "then": if the value of the expression equals the value of *fontstyle*, then the condition is true, otherwise it's false.

Parameter *dumpwindow* (normally 32) is the maximum number of characters displayed on each line of an error message when identifying the current program location.

A *(control mode)* statement causes the system to enter a certain state (mode); the *no(control mode)* statement is used to leave that state. In each mode, the system will do something special:

titletrace causes LCCD to print titles when they are encountered.

drawtrace causes LCCD to display the result of each draw command.

curtrace causes LCCD to print the computing efficiency of the curve plotting routine.

pause causes LCCD to show each line of a text file that is being input, just before that line is scanned.

pagewarning causes LCCD to give a warning message whenever a file page ends inside a subroutine definition or a section containing a title.

pensystem CAUSES LCCD to give up the pen or eraser to be used, and decides a new one by means of system parameter *pentype*.

constwidth causes LCCD to set the relative width function equal to 1, in spite of whatever has been specified. This is used for some particular font styles.

blank causes LCCD to generate strokes by plotting only the envelope curves instead of filling them, so that the user can observe strokes in detail.

crsmode causes LCCD to output to a file named **alp.590**, which can be used by METAFONT for output to an Alphatype machine.

tfnmode causes LCCD to output to a file named **xgp.29**, which can be used by METAFONT for output to an XGP.

When the condition in a conditional statement is a control mode, the condition is true when that mode has been turned on, otherwise it is false.

By means of conditional statements, the user is able to design subroutines that can generate characters of several different font styles.

The second type of section is a subroutine definition:

```

(subroutine definition) ::= subroutine <name>: <body>
<body> ::= <normal statement list> |
          <normal statement list>;<adjust command>
<adjust command> ::= adjust; |
                   adjust |
                   adjust ((<expression>,<expression>,<expression>,<expression>,<expression>)); |
                   adjust ((<expression>,<expression>,<expression>,<expression>,<expression>))
<normal statement list> ::= <normal statement> |
                           <normal statement list>;<normal statement>
<normal statement> ::= <simple normal statement> |
                      <conditional normal statement>
<conditional normal statement> ::=
  if<condition>then <normal statement list>fi |
  if <condition> then <normal statement list> else <normal statement list> fi
<simple normal statement> ::= <empty> |
                             "<title>" |
                             <expression> |
                             real <expression list> |
                             complex <z expression list> |
                             <pen or eraser> |
                             <draw command> |
                             <subroutine call>

```

This syntax indicates that certain statements are not allowed within subroutines; system parameters cannot be changed, nor can control modes be turned on or off. These things are not permitted in LCCD subroutines because the author thought it would be unsafe. Also recursive subroutines are not permitted.

The recommended way to design a new character is that you first try to execute a sequence of statements, then edit it to be a subroutine definition. If you want an automatic adjust transformation, you should use the command

adjust

followed by the period that ends the subroutine. If you want a non-automatic adjust transformation, you should use the command

adjust;

just before the final period. In both cases, after processing of the subroutine definition is finished, file `subrtn.def` will contain the resulting transformation

adjust(0, x_l , y_l , Δx , Δy).

The author would like to suggest that you let this adjustment replace the original one, because LCCD just accepts this kind of transformation without doing any computing. The fourth possibility

adjust(0, x_l , y_l , Δx , Δy);

is another non-automatic adjust transformation. This gives initial parameters for the online adjustments, so that the user can make refinements more quickly.

The final type of section is a “Chinese file”:

```

<Chinese file> ::= begin <typesetting command list> end | copy <expression>
<typesetting command list> ::= <typesetting command> |
    <typesetting command list>;<typesetting command>
<typesetting command> ::= pgsz(<expression>,<expression>,<expression>) |
    position(<expression>,<expression>,<expression>) |
    ccstr(<name string>) |
    chstr(<name string>) |
    <empty> |

```

```

        "<title>" |
        <pen or eraser> |
        <parameter> <expression> |
        <control mode> |
        no <control mode>
<name string> ::= <name> | <name> <name string>

```

Chinese files are used to tell the system what and how to output. LCCD can output to the XGP and/or to a disk file.

`pgsize( $e_1, e_2, e_3$ )` is used to define the page layout. The page will consist of e_1 lines; the space between two characters within a line will be e_2 pixels; and the space between two lines will be e_3 pixels. If the user does not give any `pgsize` command, the system will assume the default values

`pgsize(10, 10, 10).`

`position( $e_1, e_2, e_3$ )` is used to decide where to put the first character that follows this command. If $e_1 \neq 0$, the character is output on a new page; e_2 denotes how many lines should be fed before output; e_3 denotes how many blanks should be put preceding that character. If the user does not give any `position` command, the system assumes

`position(0, 0, 0).`

`ccstr` denotes the characters to be output on the XGP by giving the subroutine names, each of which has to be followed by a blank. In case of output to the XGP, the result also goes to a disk file named `xgpsr.dat`; the command

`copy     $n$`

will make n XGP copies.

`chstr` denotes the characters to be output to disk file `alp.590` or `xgp.29`, depending on either `crsmode` or `tfnmode` being turned on. For output to file `alp.590` or `xgp.29`, commands `pgsize` and `position` are meaningless. Thus, for output to XGP, do not use the command `chstr` in the Chinese file, and turn off `crsmode` and `tfnmode`; for output to file `alp.590`, do not use commands `pgsize`, `position`, and `ccstr` in the Chinese file, and remember to turn on `crsmode`. For output to file `xgp.29`, do not use commands `pgsize`, `position`, and `ccstr` in the Chinese file, and remember to turn on `tfnmode`.

LCCD treats input just as METRFont does. Thus, the user can input a disk file by typing

```
input <file name>;
```

but the extension of the file name must be "lcd". The user can also type anything that is supposed to be input, when the system prompts with *.

Besides the outputs that have been described, the user will discover that the disk file `errors.tmp` (in which all the messages the system sent to the user were recorded) may be helpful for debugging an LCCD program.

10. How to design Chinese character shapes

The best approach to designing Chinese character shapes is a combination of 'top down' and 'bottom up' procedures. That is, we first analyse the structure of characters (top down), and obtain a sequence of sets. Each set contains some subcharacters or so-called radicals as its elements, and these sets are arranged in a certain order such that the radicals in a set of higher order can be built from the radicals in sets that have lower orders. Thus, the set with lowest order consists of basic strokes or atoms, and the structure becomes more complex when the order is higher.

The next step is to build all of the characters (bottom up); i.e., a radical set having lower order is built first, and higher order ones are built later. The first set to be built contains the basic strokes. We need to use the various means provided by LCCD to design basic strokes as attractively as possible, so we must be especially earnest in doing this initial task. To build the high order radicals is different, however, somewhat like making a toy house with building blocks. From the top-down analysis we know the structure of characters, so the high order subroutine body consists of low order subroutine calls, and nothing else except a possible adjust command. Of course this building process does need experienced eyes.

We don't have to adjust a radical subroutine unless it is a Chinese character already. As described in Chapter 8, when we are designing character shapes individually, we don't have to worry about the size of those shapes. At that time what we need to consider is the relative position and size of various radicals that compose the character to be designed. An adjust command will automatically make the character fit into the standard box; and sometimes we need non-automatic adjust transformations for better visual effect.

The most difficult time is the beginning. Once several hundred radicals have been built, it becomes rather easy to generate a new character. Almost all of the components have probably been produced by that time, so you need mostly to consider only how to put them together.

Knuth asked the author to design a group of Chinese characters for the new edition of his book "The Art of Computer Programming". This group can be used as an example to illustrate the design process.

First we design the atoms that are used to build everything. There are 29 atoms whose subroutine definitions are given in Appendix 1. The atom shapes are given in the first five lines of Appendix 2. They are printed under control mode *blank*, and some of them have been magnified so that we can observe them in detail. Notice that the eighth should be drawn by an eraser. Following the 29 atom shapes in Appendix 2, there are 346 basic strokes and radicals, most of them

printed under *blank* mode. Although you cannot find their subroutine definitions, yet you can guess how to build them.

Appendix 3 contains the 112 Chinese characters that Knuth needed. They are printed in *no blank* mode, at first with the same containing box, so some of them appear to have a different size. After using non-automatic adjust transformations, another group that looks somewhat more uniform was obtained; these characters appear at the end of Appendix 3.

Appendix 4 shows another sample, which is a couplet that hangs in a beautiful garden in the author's hometown Kunming. This couplet was created about two hundred years ago; it is one of the best couplets ever created, and probably the longest (180 characters). Since it is a personal favorite of the author's, he decided to use it for the first experiments with LCCD.

Notice that the couplet in Appendix 4 is printed twice with different pen and system parameters. Actually so many different fonts can be generated, they never can be exhausted, even though the underlying LCCD description remains the same.

The text of this report, together with all of its illustrations and the examples in Appendices 2–4, was printed on an XGP printer, which has comparatively low resolution (200 dots per inch) and nonlinear distortion. Better equipment will soon be available for computer-generated output. Appendix 5 shows the 112-character font at the end of Appendix 3 when it has been printed on an Alphatype CRS, a high-quality machine whose resolution exceeds 3500 dots per inch.

Acknowledgments

I would like to thank Professor Donald Knuth for his support and encouragement. Without METAFONT, perhaps I would never have thought about the problem of a Chinese character font compiler, and I really learnt a lot from METAFONT. Without his hospitality, it would have been impossible to implement LCCD at Stanford. He also helped me to edit the draft of this paper into readable English.

I would also like to thank the people associated with the Stanford Artificial Intelligence Laboratory for the use of their computing facilities during the development of this system.

REFERENCES

[1] Donald E. Knuth, *Mathematical Typography*, Bulletin of the American Mathematical Society (new series), Vol.1, No.2 (1979), 337–372. Reprinted with corrections as part 1 of $\text{\TeX}$ and METAFONT: New Directions in Typesetting, American Mathematical Society and Digital Press, 1979.

[2] Donald E. Knuth, *METAFONT, a system for alphabet design*, part 3 of $\text{\TeX}$ and METAFONT: New Directions in Typesetting, American Mathematical Society and Digital Press, 1979.

[3] Dong Yun Mei, *An experimental system for generating high quality Chinese characters*, Chinese Journal of Computers, Vol.3, No.1 (1980), 33–37.

APPENDIX 1

```
subroutine xs11:
x0+y0+100;
lteardrop(w1,1,1.45);
draw o[0,1].
```

```
subroutine xs12:
x0+y0+ 100 ;
lteardrop(w1,1,1.8);
draw o[0,1].
```

```
subroutine xs13:
call xs11;
call xs11( -90, 1, 1, 0, 0 ).
```

```
subroutine xs14:
call xs11;
call xs12( -90, 1, 1, 0, 0 ).
```

```
subroutine xs15:
x0+y0+100;
rteardrop(w1,1,1.45);
draw o[0,-1].
```

```
subroutine xs16:
x0+y0+100;
rteardrop(w1,1,1.8);
draw o[0,-1].
```

```
subroutine xs17:
call xs13( 180, 1, 1, 0, 0 ).
```

```

subroutine xs18:
x0+y0+100;
lteardrop(w2,1,1.4)#;
draw o[0,-1].

```

```

subroutine xs19:
x0+y0+100;
lteardrop(w2,1,1.4);
draw o[0,-1].

```

```

subroutine xs110:
x0+y0+100;
rteardrop(w1,1,1.8);
draw o[0,1].

```

```

subroutine xs111:
call xs11( -90, 1, 1, 0, 0 );
call xs110( 90, 1, 1, 0, 0 ).

```

```

subroutine xs21:
cpen (w0) ;
x0+100; x1+123; x2+123; x3+117; x4+113; x5+105;
y0+100; y1+100; y2+102; y3+108; y4+108; y5+100;
fill 5..1[x1-x0,y1-y0]..2[x3-x2,y3-y2]..3[x3-x2,y3-y2]
. .4[x5-x4,y5-y4]..5;
draw 0. .1[x1-x0,y1-y0]..2[x3-x2,y3-y2]..3[x3-x2,y3-y2]
, .4[x5-x4,y5-y4]..5.

```

```

subroutine xs31:
cpen (w1) ;
x0+100; x1+123; x2+123; x3+117; x4'-113; x5+105;
y0+100; y1+100; y2+102; y3+108; y4+108; y5+100;
draw 0. .1[x1-x0,y1-y0]..2[x3-x2,y3-y2]..3[x3-x2,y3-y2]
. .4[x5-x4,y5-y4]..5.

```

```

subroutine xs32:
call xs31( 90, -0.5, 1, 0, 0 ).

```

```

subroutine xs41:
cpen(w1) ;
x0+100; x1+150;
y0+100; y1+150;
draw |1#|0[0,1]. . |0.3#|1[1,0].

```

```

subroutine xs42:
cpen(w1) ;
x0+100; x1+150;
y0+100; y1+150;
draw |1#|0[0,1]. . |0.3|1[1,0].

```

```

subroutine jb11:
cpen(w1) ;
x0+100; x1+100; x2+100; x3+120; x4+95; x5+105;
y0+150; y1+100; y2+50; y3+100; y4+55; y5+55;
fill 0[1,-2]..1[-1,-2]..4..2[1,0]..5..3[0,1]..0[-1,1];
draw |0.1|0[1,-2]..|0.4|1[-1,-2]..|1|4..|1|2[1,0]..|1|5
      ..|0.5|3[0,1]..|0.1|0[-1,1].

```

```

subroutine jb21:
cpen (w0) ;
x0+50; x1+150;
y0+100; y1+100;
draw 0..1.

```

```

subroutine jb31:
cpen(w1) ;
x0+100; x1+100;
y0+50; y1+150;
draw 0..1.

```

```

subroutine jb41: x20+100; y20+15; y21+15;
cpen(w1) ;
x0+150; x1+x0-x20;
y0+150; y1+50;
y10+arg(z1-z0)+y20; y11+arg(z1-z0)-y21;
draw 0[cos(y10),sin(y10)]..|0.3|1[cos(y11),sin(y11)].

```

```

subroutine jb42: x20+50; y20+45; y21+15;
cpen(w1) ;
x0+150; x1+x0-x20;
y0+150; y1+50;
y10+arg(z1-z0)+y20; y11+arg(z1-z0)-y21;
draw 0[cos(y10),sin(y10)]..|0.3|1[cos(y11),sin(y11)].

```

```

subroutine jb43: x20+50; y20+30; y21+45;
cpen(w1) ;
x0+150; x1+x0-x20;
y0+150; y1+50;
y10+arg(z1-z0)+y20; y11+arg(z1-z0)-y21;
draw 0[cos(y10),sin(y10)]..|0.3|1[cos(y11),sin(y11)].

```

```

subroutine jb44: x20+50; y20+30; y21+30;
cpen(w1) ;
x0+150; x1+x0-x20;
y0+150; y1+50;
y10+arg(z1-z0)+y20; y11+arg(z1-z0)-y21;
draw 0[cos(y10),sin(y10)]..|0.3|1[cos(y11),sin(y11)].

```

```

subroutine jb45: x20+50; y20+30; y21+15;
cpen (w1) ;
x0+150; x1+x0-x20;
y0+150; y1+50;
y10+arg(z1-z0)+y20; y11+arg(z1-z0)-y21;
draw 0[cos(y10),sin(y10)]..|0.3|1[cos(y11),sin(y11)].

```

```

subroutine jb46: x20+50; y20+30; y21+15;
cpen(w1) ;
x0+150; x1+x0-x20;
y0+150; y1+50;
y10+arg(z1-z0)+y20; y11+arg(z1-z0)-y21;
draw |0.3|0[cos(y10),sin(y10)] . . |1|1 [cos(y11),sin(y11)].

```

```

subroutine jb47: x20+50; y20+45; y21+15;
cpen(w1);
x0+150; x1+x0-x20;
y0+150; y1+50;
y10+arg(z1-z0)+y20; y11+arg(z1-z0)-y21;
draw 0[cos(y10),sin(y10)] . . 1 [cos(y11),sin(y11)] .

```

```

subroutine jb48: x20+50; y20+45; y21+15;
cpen(w1) ;
x0+150; x1+x0-x20;
y0+150; y1+50;
. y10+arg(z1-z0)+y20; y11+arg(z1-z0)-y21;
draw |0.6|0[cos(y10),sin(y10)] . . |1|1[cos(y11),sin(y11)].

```

```

subroutine jb51: x20+100; x21+80; y20+10;
cpen(w1) ;
x0+50; x1+50; x2+60; x3+x2+x21;
y0+150; y1+y0-x20-5; y2+y1-10; y3+y2;
. y10+-y20; y11+y20; y12+y21;
draw 0.. 1 [x1-x0,y1-y0] . . 2[cos(y10),sin(y10)] . . 3[cos(y11),sin(y11)] .

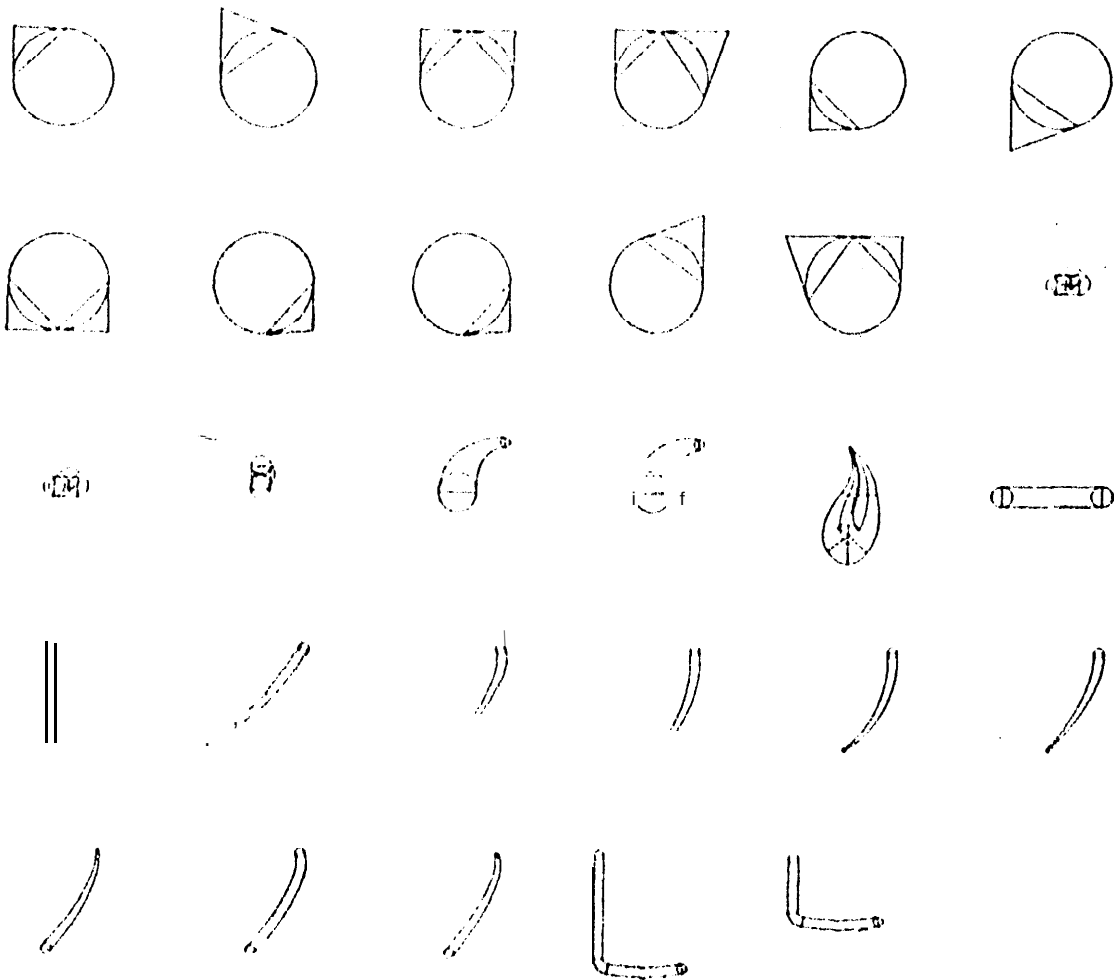
```

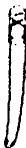
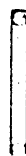
```

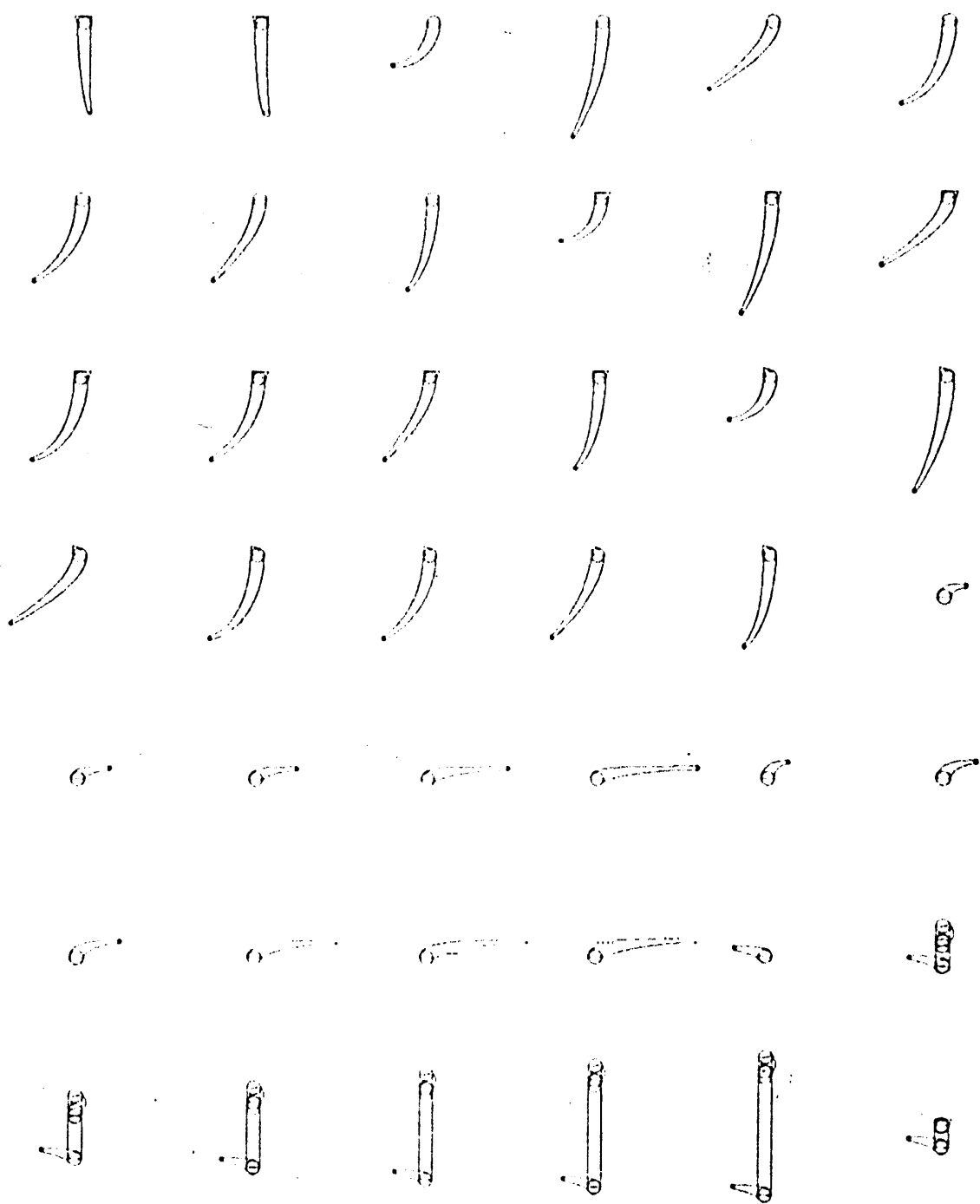
subroutine jb52: x20+50; x21+80; y20+10;
cpen(w1) ;
x0+50; x1+50; x2+60; x3+x2+x21;
y0+150; y1+y0-x20-5; y2+y1-10; y3+y2;
y10+-y20; y11+y20; y12+y21;
draw 0.. 1 [x1-x0,y1-y0] . . 2[cos(y10),sin(y10)] . . 3[cos(y11),sin(y11)] .

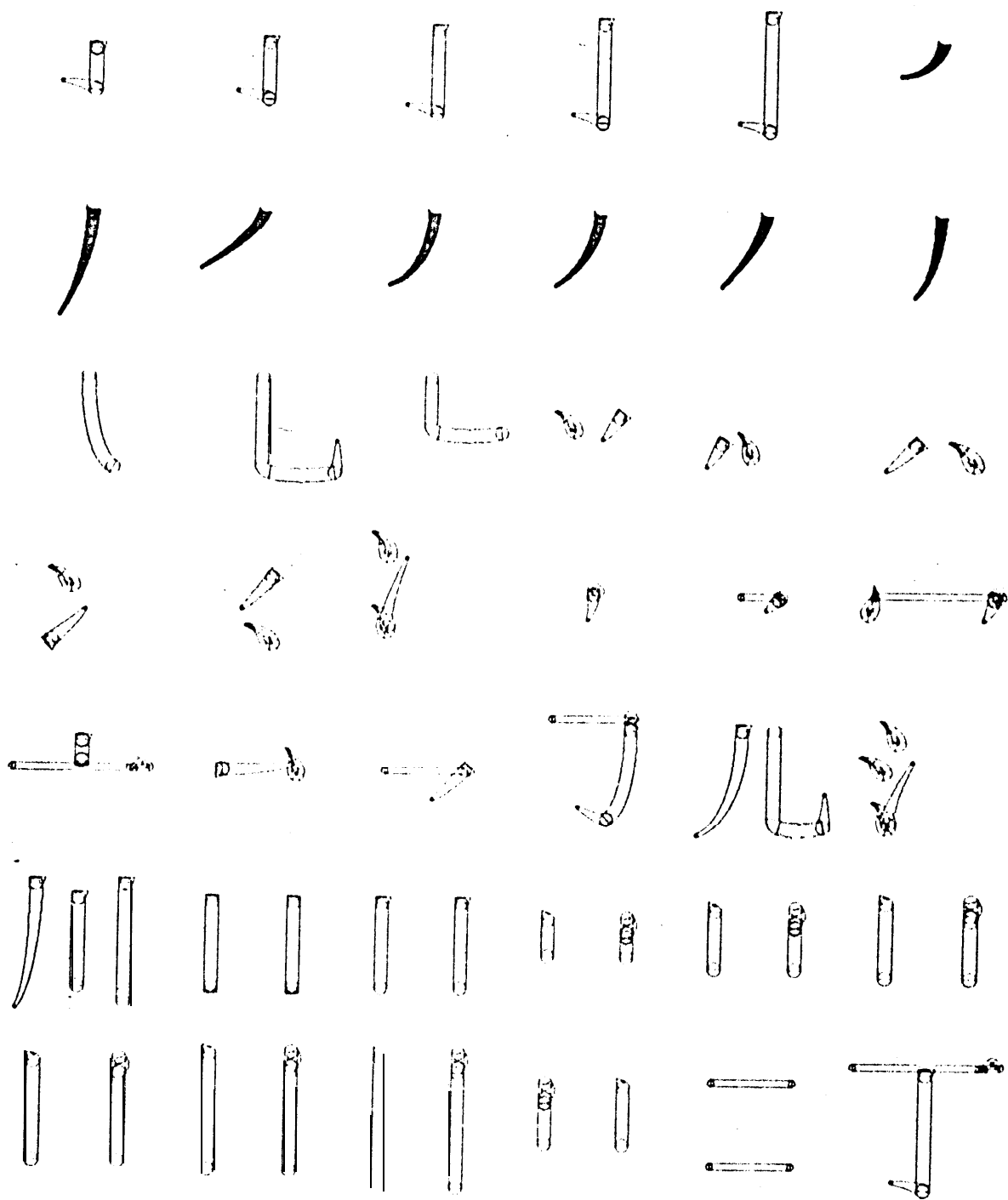
```

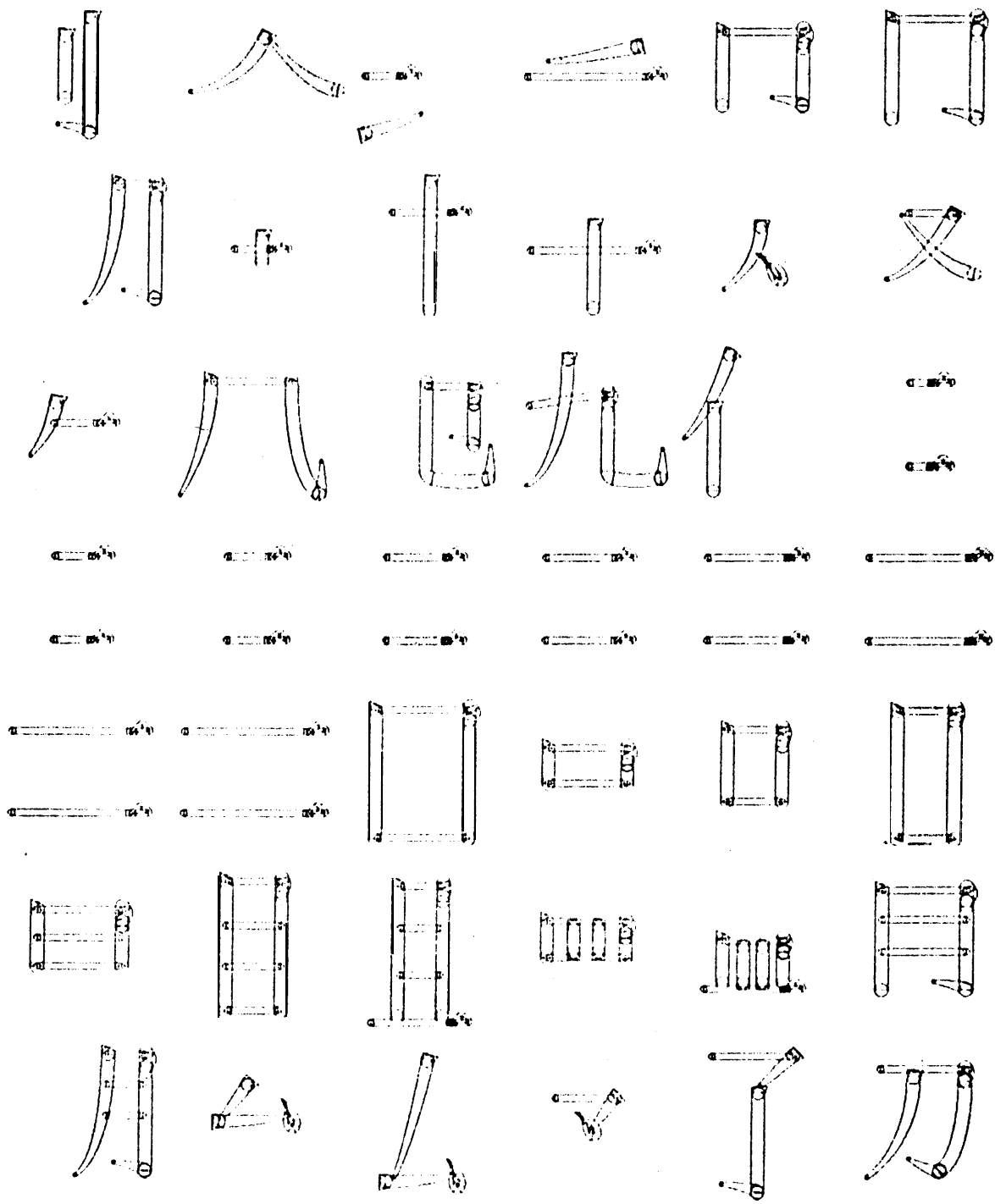
APPENDIX 2

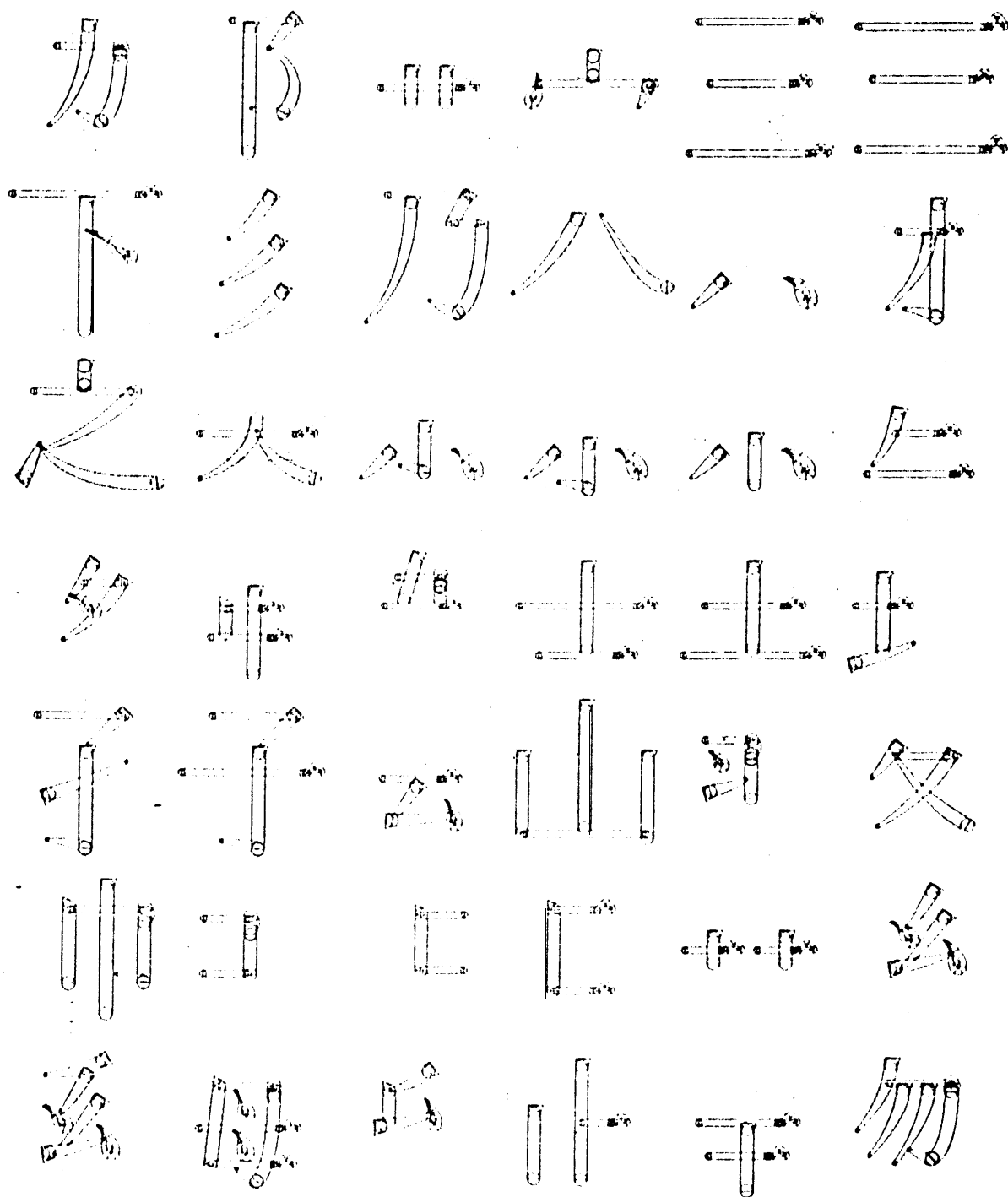


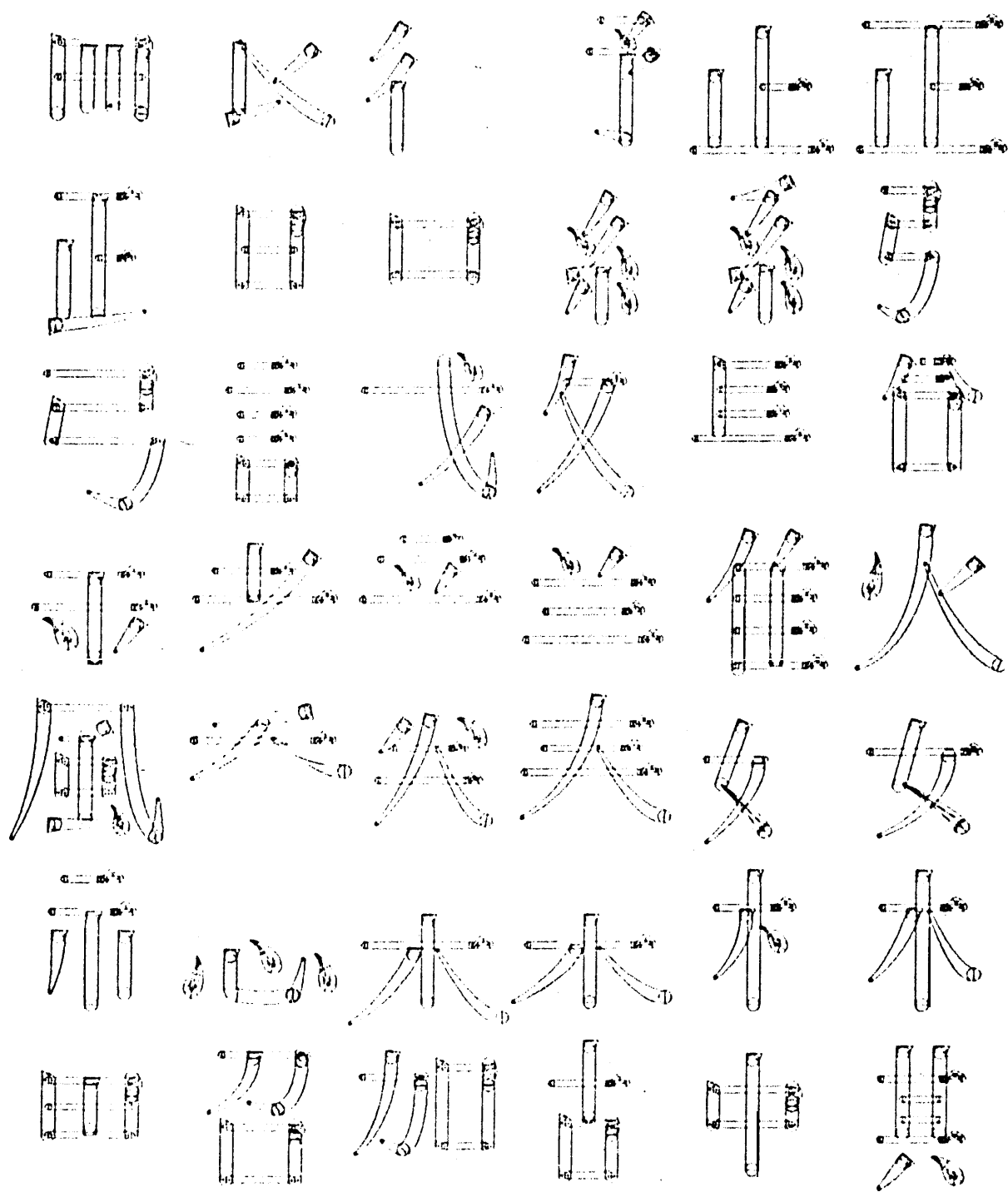






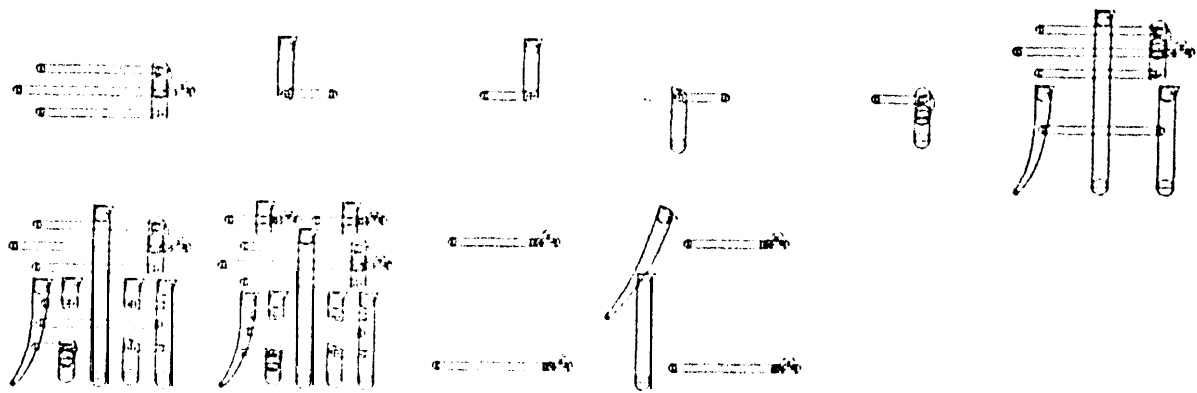








威	武	我	東	畏	雙
帝	重	谷	曲	三	腹
罪	止	罪	罪	貝	泰
禾	禾	易	暇	晶	知
帝	炎	辰	史	桀	兆
帝	金	亞	金	門	諸
華	華	才	才	華	川



澤	陳	繁	昌	鄭	川	訓	秦	九	韶	范	崇
德	高	納	黃	石	橋	善	弘	孔	祥	重	洪
谷	政	昭	孫	子	秀	俊	祖	冲	之	董	美
權	田	勇	姚	期	智	儲	楓	韞	中	憚	元
一	有	誠	張	系	國	朱	世	傑	鍾	金	芳
蓉	開	萊	胡	强	光	明	李	再	華	林	身
劉	燭	朗	耀	榮	宮	正	室	賀	三	郎	野
下	浩	平	成	杉	藤	雄	財	丁	錦	王	亞
威	義	孝	山	尙	葉	志	堅	袁	師	弓	場
敏	嗣	蕭	仁								

澤	陳	繁	昌	鄭	川	訓	秦	九	韶	范	崇
德	高	納	黃	石	橋	善	弘	孔	祥	重	洪
谷	政	昭	孫	子	秀	俊	祖	冲	之	董	美
權	田	勇	姚	期	智	儲	楓	韞	中	惲	元
一	有	誠	張	系	國	朱	世	傑	鍾	金	芳
蓉	開	萊	胡	强	光	明	李	再	-PI)+- - i i -	林	身
劉	燭	朗	耀	榮	宮	正	室	賀	三	郎	野
下	浩	平	成	杉	藤	雄	財	丁	錦	王	亞
威	義	孝	山	尙	葉	志	堅	袁	師	弓	場
敏	嗣	蕭	仁								

五百里滇池奔來眼底披巾岸憤喜茫茫空闊無邊看東驤神
駿西書靈儀北走蜿蜒南翔縞素高人韻士何妨蓮勝登臨赴蟹嶼
螺洲梳裹就風鬟霧鬢更蘋天葦地點綴些翠羽丹霞莫辜負四圍
香稻萬頃晴沙九夏芙蓉三春楊柳

數千年往事注到心頭把酒凌虛歎滾滾英雄誰在想漢習樓
船唐標鐵柱宋揮玉斧元跨革囊偉烈豐功費盡移山心力儘珠厓
畫棟捲不及暮雨朝雲便斷碣殘碑都付與蒼煙落照只贏得幾杵
疏鐘半江漁火兩行秋雁一枕清霜

岸中披眼底來奔池滇百里五
駿神素高螺地香柳凌船偉畫碑疏霜
神素高螺地香柳凌船偉畫碑疏霜
縞蟹天葦圍楊酒樓囊簾殘杵清
東翔趨蘋負三春把習革珠碣幾枕
看南臨更幸蓉心漢跨儘斷得雁
來邊蜿蜒登鬢莫芙到在斧心雲只秋
無蜿蜒勝霧霞夏注誰玉山朝照行
池闊走選鬟丹九事英雄揮移雨落兩
滇空北妨風羽沙往英宋盡暮煙火
百里茫儀何就翠晴年滾滾柱費及蒼
喜耆靈士褒些頃千滾滾鐵功不與江
憤西耆韻梳綴萬數歎標豐捲付半鐘

63

SAMPLE Sunday, October 26, 1968 11:12:40 1405

63

