ELECTRONIC PUBLISHING, VOL. 6(3), 195–205 (SEPTEMBER 1993)

A curve fitting algorithm for character fonts

KOICHI ITOH[1] AND YOSHIO OHNO

Faculty of Science and Technology,
Keio University, 3–14–1 Hiyoshi, Kohoku-ku,
Yokohama 223, Japan

SUMMARY

This paper presents an algorithm that automatically generates outline fonts from a grey-level image of a character obtained by a scanner. Our algorithm begins by extracting contour points from the image and dividing the points into a number of segments at the corner points. The next step is fitting a piecewise cubic Bézier curve to each segment.

To fit cubic Bézier curves to segments, we use least-squares fitting, without fixing the end points of the curves. We locate the end points by computing the intersection of the adjoining curves. This algorithm greatly improves the shape of the corner of the outline fonts.

KEY WORDS Curve fitting algorithm Grey-level characters Kanji characters

1 INTRODUCTION

Recent developments in personal computers and laser-beam printers have made 'desktop publishing' systems very familiar. Such systems in Japan, however, are still poor in expressive power because only a small number of fonts for Kanji, characters used in writing Japanese, are available.

In many desktop publishing systems, the shapes of the characters are stored in the computer memory in terms of their outlines, and the outlines are expressed as cubic Bézier curves.

Currently, outline fonts expressed in Bézier curves are usually produced by scanning the original characters drawn on paper and then editing them interactively on the screen. This work is especially time-consuming for Japanese fonts, because 7,000 Kanjis are necessary for Japanese documents, and because each Kanji has a more complicated shape than Roman characters. This is the reason why only a few fonts are supported in Japanese desktop publishing systems.

On the other hand, because many font-selling companies have a lot of original character fonts drawn in ink on paper, an automatic generation technique from these original characters is required. In particular, automatic fitting of cubic Bézier curves to the grey-level character images that are input into a computer using a scanner is necessary to improve such a situation.

To date a lot of research has been done in this area (for examples, see [1,2,3,4,5]). Our interest centres on the parametric curve representation of outline fonts, especially on cubic Bézier curve representation. In this approach, Plass and Stone [3] gave some excellent B-spline outline fonts, but their algorithm needs huge computation mainly because of the dynamic programming for determining optimum breakpoints.

[1] Current affiliation: Sony Corporation

CCC 0894–3982/93/030195–11 *Received 15 August 1993*
©1993 by John Wiley & Sons, Ltd. *Revised 1 December 1993*

In [1], some defects of the former algorithms [6,7,8,3] are located, and solutions are proposed, but we believe that at least the inverse slope problem can be avoided by improving the contour point extraction phase.

In this paper, we propose a new algorithm for automatic generation of outline fonts from original characters on paper. Some results of the application of our algorithm to some styles of Kanji characters will be given, and the performance of the algorithm will be discussed.

2 THE ALGORITHM

Ideal outline fonts should satisfy the following two conditions:

1) faithfulness to the original characters, and
2) the need for a small number of curves.

From this point of view, we improve the previous algorithms in the two following respects:

1) more accurate estimation of contour points, based on the grey-level data which was discarded in the previous algorithms, and
2) fitting curves without fixing end points in order to avoid degradation around the corner points.

Our algorithm consists of the following steps:

1) extraction of contour points from the grey-level images using Avrahami's algorithm [9],
2) extraction of supposed corner points using Davis' algorithm [6],
3) fitting of piecewise cubic Bézier curves to each segment using a least squares method, and
4) determination of corner points using the Bézier clipping algorithm [10].

In the following subsections, we explain these steps in detail.

2.1 Extraction of contour points

We used the algorithm proposed by Avrahami and Pratt [9] for the extraction of contour points from the grey-level image that is input by the scanner. Using this algorithm, the effect of the error introduced by the conversion of the grey-level image to a bilevel image can be avoided. It can extract contour points in subpixel precision.

In the original form of their algorithm [9], however, some human intervention is necessary for the specification of initial trace points. We modified their algorithm slightly to avoid this.

Another disadvantage of Avarahami's algorithm is the degradation around the corner points. This problem will be discussed in the next subsection.

2.2 Extraction of supposed corner points

In this step, we extract the supposed corner points from the obtained contour points using Davis' algorithm [6]. The contour points are divided into groups using the supposed corner points.

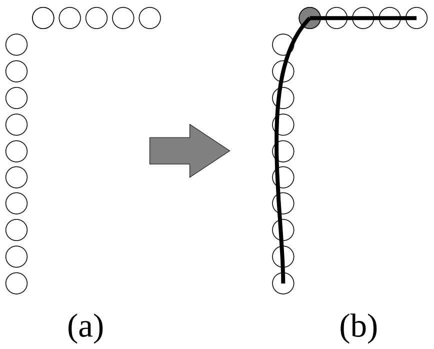

Figure 1. Degradation around a corner point

Davis' algorithm approximates the curvature $C^k(i)$ at each contour point $P_i = (x_i, y_i)$ according to the following formula:

$$C^k(i) = \frac{a_{ik} \cdot b_{ik}}{|a_{ik}| \cdot |b_{ik}|}$$

where

$$\begin{aligned} a_{ik} &= (x_i - x_{i+k}, y_i - y_{i+k}), \\ b_{ik} &= (x_i - x_{i-k}, y_i - y_{iik}). \end{aligned}$$

The contour points which take the local maxima are considered to be the corner points. The best value of k depends on several factors, such as the resolution of the original image. We set a threshold value T for $C^k(i)$ and take the point P_i as the corner point if $C^k(i)$ takes a local maximum and if $C^k(i) > T$. Without the threshold, the algorithm is too sensitive to small variations of $C^k(i)$.

We adopted this algorithm temporarily, but the precise detection of corner points is essentially impossible, because only the font designer knows whether a supposed corner point is a true corner point or just a point with large curvature. The best a computer system can do is to show candidates for the corner points and to provide the designer with the means for overriding the computer's proposals.

2.3 Curve fitting

Previous curve fitting algorithms [2,4,5] determine the end points first, and then fit a curve by fixing the end points.

Using this approach for the contour points in Figure 1a, such fitted curves as in Figure 1b would be obtained. We use Avrahami's algorithm for extracting the contour points as accurately as possible, but even with this algorithm a pattern of contour points such as Figure 1a is often produced. Inaccurate estimation of corner points is fatal for well-shaped curves when they are fixed as the end points in the first stage of the curve fitting.

Another disadvantage of curve fitting with fixed end points is the reduction in the degree of freedom. It makes the fitting process easier, but tends to increase the number of curves used unnecessarily.

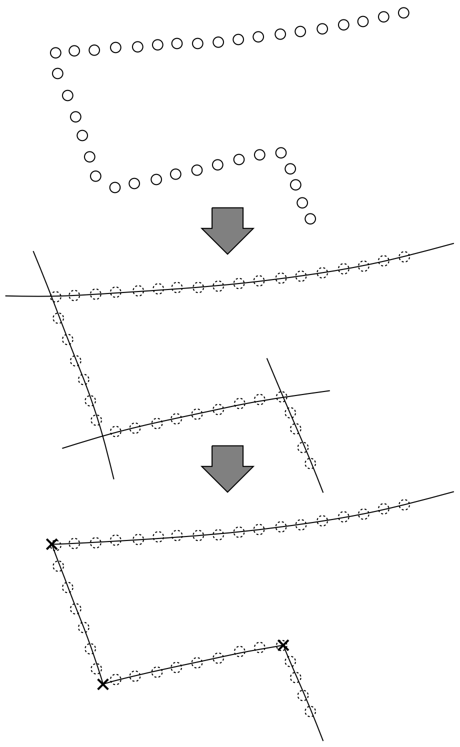

Figure 2. Our approach to curve fitting

From these considerations, we adopt the following approach (Figure 2): divide the contour points into groups at the supposed corner points (we call each group a segment), fit a curve to each segment by giving even weight to each contour point in the segment, and then determine the true corner points by computing the intersections of the adjoining curves.

In dividing the contour points into segments, we remove the supposed corner points, because the estimation of their positions is usually less accurate than other points.

2.3.1 *Our approach to curve fitting*

We use the least squares method for fitting a curve to each segment. The method of fitting one curve to a segment will be described first, then the fitting of more than one curve will be given. Suppose a segment is given as an ordered set of contour points $P_i = (x_i, y_i)$, $i = 1, \ldots, n$. To simplify the formula, we use the power basis expression for a Bézier curve $B(t) = (B_x(t), B_y(t))$:

$$\begin{cases} B_x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \\ B_y(t) = a_y t^3 + b_y t^2 + c_y t + d_y, \qquad 0 \le t \le 1 \end{cases}$$

$$\begin{cases} a_x\sum_{i=1}^{n} t_i^6 + b_x\sum_{i=1}^{n} t_i^5 + c_x\sum_{i=1}^{n} t_i^4 + d_x\sum_{i=1}^{n} t_i^3 & = \sum_{i=1}^{n} x_i t_i^3 \\ a_x\sum_{i=1}^{n} t_i^5 + b_x\sum_{i=1}^{n} t_i^4 + c_x\sum_{i=1}^{n} t_i^3 + d_x\sum_{i=1}^{n} t_i^2 & = \sum_{i=1}^{n} x_i t_i^2 \\ a_x\sum_{i=1}^{n} t_i^4 + b_x\sum_{i=1}^{n} t_i^3 + c_x\sum_{i=1}^{n} t_i^2 + d_x\sum_{i=1}^{n} t_i & = \sum_{i=1}^{n} x_i t_i \\ a_x\sum_{i=1}^{n} t_i^3 + b_x\sum_{i=1}^{n} t_i^2 + c_x\sum_{i=1}^{n} t_i + d_x n & = \sum_{i=1}^{n} x_i \end{cases}$$

(Similar system for y-coordinates.)

Figure 3. The least squares method

To determine the coefficients based on the given contour points, the parameter value t_i for each contour point should be determined first. We use chord-length parameterization for this purpose. That is,

$$t_i = \begin{cases} 0, & i = 1 \\ \dfrac{\text{length of polygonal line } P_1P_2\cdots P_i}{\text{length of polygonal line } P_1P_2\cdots P_n}, & 1 < i \le n \end{cases}$$

Note that these t_is are only the initial values, and they will be improved in the following stages.

Then the squared sum S of the distances between P_i and their corresponding points $B(t_i)$ on the curve is computed as:

$$\begin{aligned} S &= \sum_{i=1}^{n}[\text{distance between } B(t_i) \text{ and } P_i]^2 \\ &= \sum_{i=1}^{n}(a_x t_i^3 + b_x t_i^2 + c_x t_i + d_x - x_i)^2 + \sum_{i=1}^{n}(a_y t_i^3 + b_y t_i^2 + c_y t_i + d_y - y_i)^2 \end{aligned}$$

By setting $\partial S/\partial a_x$, $S/\partial b_x$, $S/\partial c_x$, $S/\partial d_x$, $S/\partial a_y$, $S/\partial b_y$, $S/\partial c_y$, and $S/\partial d_y$ to zero, we can obtain a_x, b_x, c_x, d_x, a_y, b_y, c_y, and d_y that make S minimum.

The two systems of linear equations in Figure 3 are obtained from this computation. The Gaussian elimination method can solve these systems stably.

The curves thus obtained are not necessarily the best ones, because the parameter values t_i used are not always the best. Therefore a reparameterization is necessary. This is a process for improving the parameter values based on the obtained curve (see [3]).

In our algorithm, the reparameterization is done for each contour point P_i using the following formula:

$$[B(t) - P_i] \cdot B'(t) = 0$$

This is a quintic equation in t, and can be solved by Newton-Raphson's method using t_i as the initial value. The solution t is used as the new parameter value t_i in the next stage.

After solving the quintic equations for all contour points, it is necessary to normalize the new parameters values so that the smallest t_i is zero and the largest t_i is one. Then apply the least squares method again to the new t_is to obtain the new curve. By repeating this process, the curve converges to the contour points.

Next, the method of fitting more than one Bézier curve to a segment is described. The two-curve case is considered first. Suppose that the two Bézier curves are expressed as:

$$\begin{cases} B_x^{(1)}(t) = a_x^{(1)}t^3 + b_x^{(1)}t^2 + c_x^{(1)}t + d_x^{(1)} \\ B_y^{(1)}(t) = a_y^{(1)}t^3 + b_y^{(1)}t^2 + c_y^{(1)}t + d_y^{(1)}, \qquad 0 \le t \le 1 \end{cases}$$
$$\begin{cases} B_x^{(2)}(t) = a_x^{(2)}t^3 + b_x^{(2)}t^2 + c_x^{(2)}t + d_x^{(2)} \\ B_y^{(2)}(t) = a_y^{(2)}t^3 + b_y^{(2)}t^2 + c_y^{(2)}t + d_y^{(2)}, \qquad 0 \le t \le 1 \end{cases}$$

In this case we have to consider the continuity conditions. We use G^1 to increase the degree of freedom. From the G^0 condition, we have

$$\begin{cases} B_x^{(1)}(1) = B_x^{(2)}(0) \\ B_y^{(1)}(1) = B_y^{(2)}(0). \end{cases}$$

As the two curves should have a common tangent line at the connection point, we have

$$\begin{cases} \alpha B_x^{(1)\prime}(1) = B_x^{(2)\prime}(0) \\ \alpha B_y^{(1)\prime}(1) = B_y^{(2)\prime}(0) \end{cases}$$

where α is a given positive constant. We determine α as

$$\alpha = \frac{\text{length of polygonal line } P_m \cdots P_n}{\text{length of polygonal line } P_1 \cdots P_m}$$

because α depends on the ratio of the lengths of the two curves. In this formula m indicates the division point.

Thus we have 12 unknowns $a_x^{(1)}$, $b_x^{(1)}$, $c_x^{(1)}$, $d_x^{(1)}$, $a_x^{(2)}$, $b_x^{(2)}$, $c_x^{(2)}$, and $d_x^{(2)}$. By setting $\partial S/\partial a_x^{(1)}$, $\partial S/\partial b_x^{(1)}$, $\partial S/\partial c_x^{(1)}$, $\partial S/\partial d_x^{(1)}$, $\partial S/\partial a_x^{(2)}$, $\partial S/\partial b_x^{(2)}$, $\partial S/\partial c_x^{(2)}$, and $\partial S/\partial d_x^{(2)}$ to zero, two systems of linear equations in six unknowns each are obtained. By solving these using Gaussian elimination, and by applying reparameterization, we have the two Bézier curves that are connected in G^1 to each other and that fit best to the contour points.

The method of fitting more than two Bézier curves can be obtained similarly.

Our algorithm for curve fitting for one segment can be summarized as follows:

1. Compute initial parameter values t_i based on the chord length.
2. Fit one Bézier curve using the least squares method.
3. If the fit is good, output the curve and exit; otherwise go to the next step.
4. Apply reparameterization and fit one Bézier curve again. If the fit is good, output the curve and exit.
5. Repeat the previous step. If a good fit is not obtained after a specified number of repetitions, fit two Bézier curves to the segment.
6. If the fit with two Bézier curves does not give good results, increase the number of Bézier curves and try again.

Table 1. Some statistics for a ‘Gothic’ character in Figures 4d and 7

	Number of curves	Largest squared error
Schneider’s method	34	0.249529
Our method	28	0.249840

We judge the goodness of the fit based on the largest distance between the contour point and its corresponding point on the curve. When we have to increase the number of Bézier curves, we divide the segment at the contour point that has the largest distance from the corresponding point on the curve.

2.3.2 Determination of end points

As we fit Bézier curves without fixing the end points, we have to determine the end points after we obtain all the Bézier curves.

The obtained Bézier curves in the previous section do not connect to each other. Therefore, we extend each curve in both directions so that each pair of adjoining curves has an intersection point. For the computation of the intersection point we use Bézier clipping [10], which gives the intersection of two parametrically expressed curves stably and efficiently.

Finally, normalize the parameter t so that $t = 0$ holds at the starting point and $t = 1$ at the end point.

3 EXAMPLES

The process of our algorithm is illustrated using a character in a ‘Gothic’ font.[2] Figure 4a shows the scanned image of the character, and Figure 4b shows the computed outline points and the supposed corner points. The obtained Bézier curves and their extensions are shown in Figure 4c, and the final outline character is given in Figure 4d.

We show two more examples in Figures 5 and 6.

4 EVALUATION

For the comparison, the character obtained by Schneider’s curve fitting method [2] from the same image is shown in Figure 7. Some statistics on these characters are summarized in Table 1.

4.1 Shape of curves

The largest errors in the two methods are very similar, but the shapes differ greatly, especially around the corners. This can be seen more clearly in the magnified figure (Figure 8). This shows the effect of our fitting method in which the end points are not fixed.

[2] In Japan a font similar to sans serif is called ‘Gothic’.

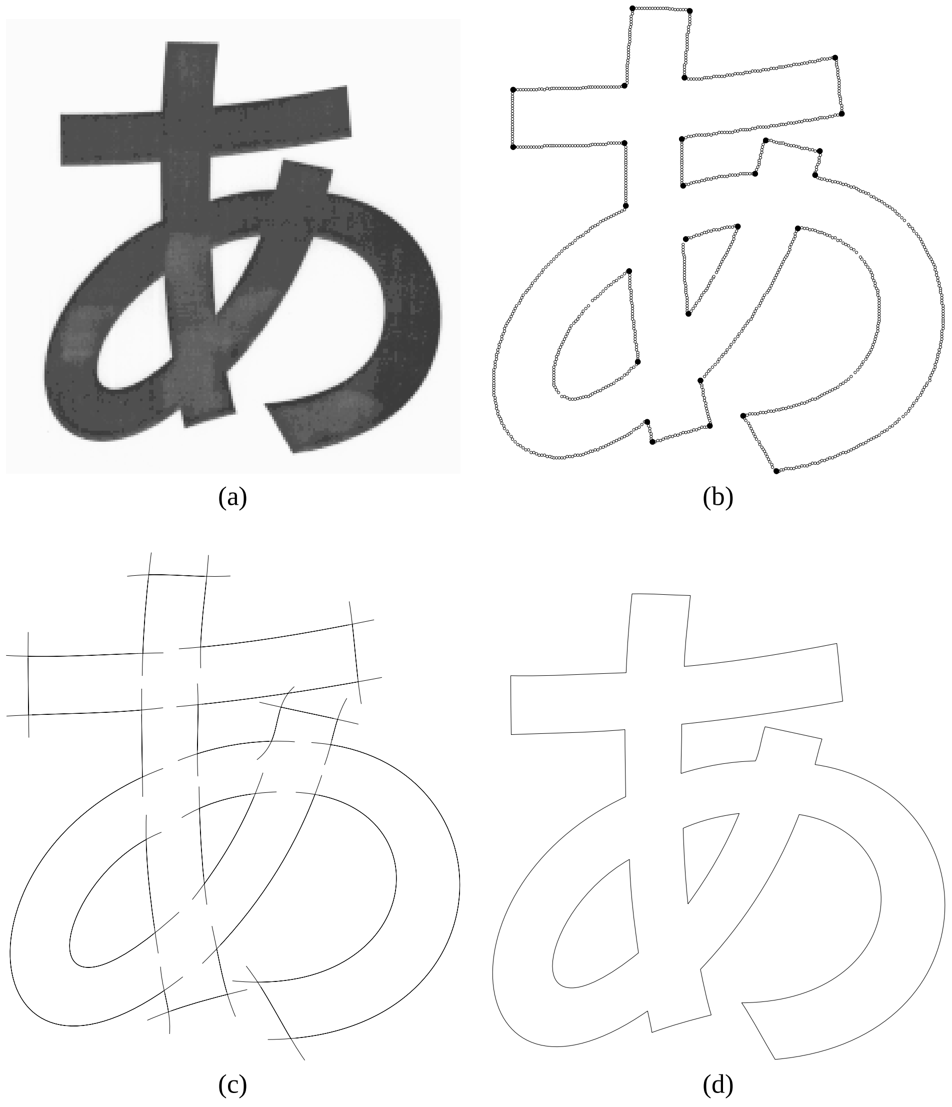

Figure 4. *A 'Gothic' character; (a) An input image; (b) Computation of contour points and corner points (● indicates a corner point); (c) Obtained Bézier curves and their intersections; (d) Final outline font.*

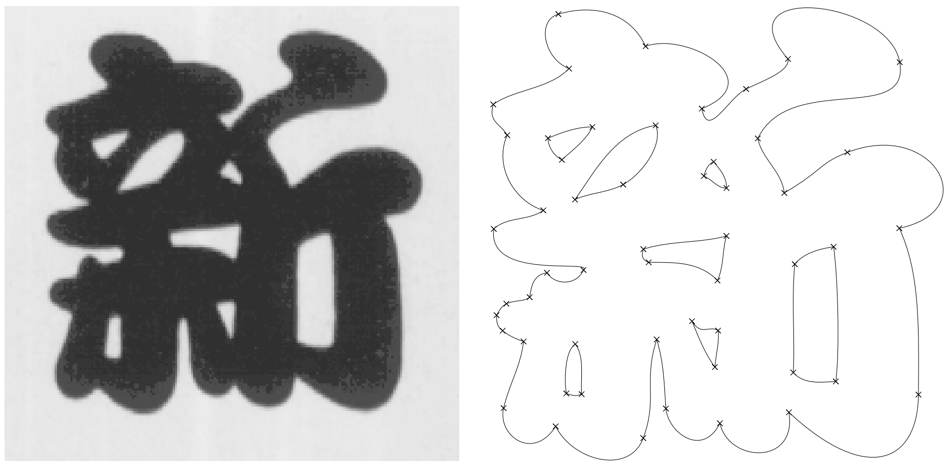
Figure 5. The second example – 'Kantei' style (a style based on calligraphic characters)

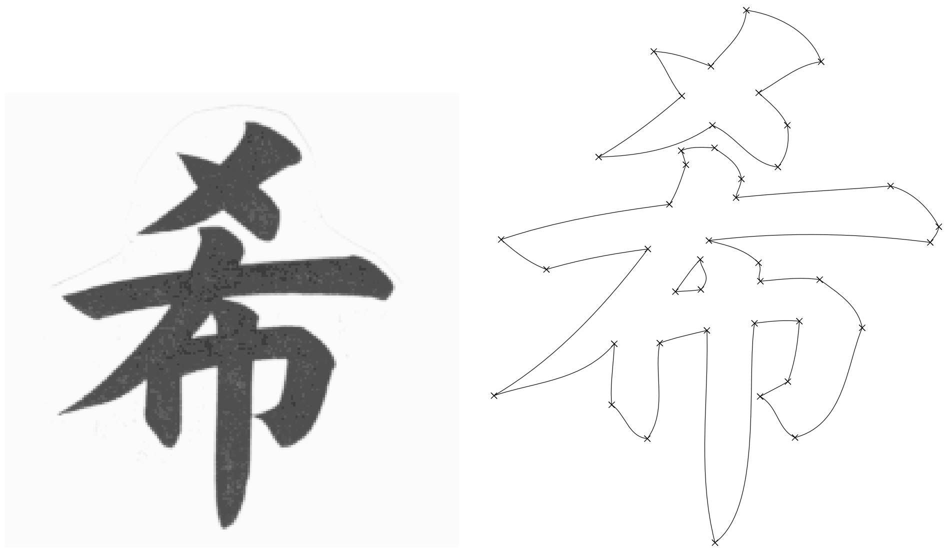
Figure 6. The third example – 'Kaisho' style (another style based on calligraphic characters)

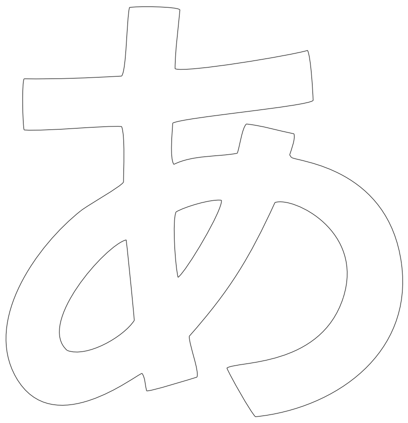

Figure 7. The outline font obtained by Schneider's method

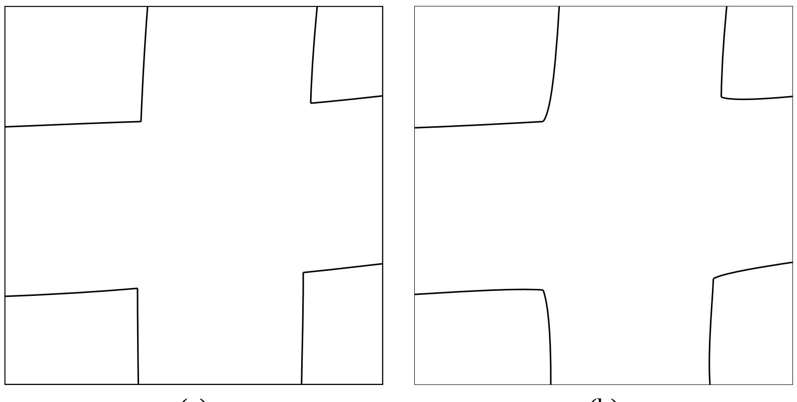

Figure 8. An enlarged portion; (a) An outline obtained by Schneider's method; (b) The corresponding outline obtained by our method.

4.2 Number of curves

The number of Bézier curves used is smaller in our algorithm. This is because by fixing the end points the degree of freedom is reduced in the previous method, and because to compensate for the unnatural shape at the corner more curves are used.

5 CONCLUSIONS

We proposed an algorithm for automatic generation of outline fonts expressed in cubic Bézier curves from the original character on paper. As we do not fix the end points in the fitting process, the generated font has smaller curves and has a shape more faithful to the original, especially at the corners.

To apply our algorithm to calligraphic characters which have very long smooth segments, however, we would have to fit many Bézier curves to a segment. To do this, some symbol manipulation mechanism like that of Mathematica or Maxima would be necessary to derive the systems of linear equations to solve the least squares method. Therefore, our method is especially suitable to such fonts as 'Gothic' (in the Japanese sense) or 'Kaisho' style (a font based on calligraphic characters but which has a very stiff impression).

REFERENCES

1. J. Gonczarowski, 'A fast approach to auto-tracing (with parametric cubics)', in *Raster Imaging and Digital Typography II*, eds. R. Morris and J. André, pp. 1–15. Cambridge University Press, (1991).
2. P. Schneider, 'An algorithm for automatically fitting digitized curves', in *Graphics Gems*, 612–626, Academic Press, 1990.
3. M. Plass and M. Stone, 'Curve-fitting with piecewise parametric cubics', *SIGGRAPH '83 Proceedings*, 229–239, (1983).
4. Torishima, Yamazaki, 'Approximation of character contours by piecewise polynomials', Technical Report PRU–87–107, IEICE Tech. Rep., (1988). (in Japanese).
5. Yamada and Sato, 'A conversion technique from polygonal approximation to cubic Bézier expression', *SIG on Graphics & CAD, IPSJ*, **38**(2), (1989). (in Japanese).
6. L. Davis, 'Shape matching using relaxation techniques', *IEEE Trans. PAMI*, **1**, 60–72, (1979).
7. J. Hoschek, 'Approximate conversion of spline curves', *Computer Aided Geometric Design*, **4**, 59–66, (1987).
8. J. Hoschek, 'Spline approximation of offset curves', *Computer Aided Geometric Design*, **5**, 33–40, (1988).
9. G. Avrahami and V. Pratt, 'Sub-pixel edge detection in character digitization', in *Raster Imaging and Digital Typography II*, eds. R. Morris and J. André, pp. 54–64. Cambridge University Press, (1991).
10. T. W. Sederberg and T. Nishita, 'Curve intersection using Bézier clipping', *CAD*, **22**(9), 538–549, (1990).