R. G. Casey
T. D. Friedman
K. Y. Wong

Automatic Scaling of Digital Print Fonts

New raster-based printers form character patterns using carefully designed matrices of dots. It is desirable to be able to use fonts designed for one printer on a different machine, but to do so the dot matrix patterns should first be scaled to the second printer's resolution. If the scaling is carried out as a simple interpolation, however, severe degradation in the appearance of the characters may occur. A new algorithm reduces such degradation by recognizing attributes associated with print character quality in the original patterns and then correcting the scaled patterns in order to maintain those attributes. Attributes that are detected and preserved during scaling include local and global symmetries, stroke width, sharpness of corners, and smoothness of contour. The method has been used both to scale low-resolution fonts to a finer representation and to reduce the scale of high-resolution photocomposer fonts for output on an office-type printer.

Introduction

A variety of raster-based printers have been introduced which form characters as matrices of dots. This has created a need for digital print fonts to support a range of publishing activities. The digital technology permits new character sets to be specified in only a few kilobytes of disk storage, and purchasers of the printers have been encouraged to develop their own fonts for particular applications. Indeed, some organizations have devoted considerable effort to designing their own character fonts for particular printers. One widely used font catalog lists 421 different digital fonts in 37 different styles [1].

The fonts require meticulous design since the appearance of the printed character depends upon the position of each dot in the matrix pattern. The cost of designing an extensive font library for a new printer may reach several millions of dollars, and it is extremely time consuming. Unfortunately, fonts designed for one printer cannot directly be used on machines having different print resolutions unless the patterns are first scaled to the new array size. The question arises whether automatic techniques can be applied to convert existing fonts to the resolution of a new printer. Digital interpolation has been used for this purpose, as described below, but it suffers from major limitations.

Interpolation

Conventional digital scaling methods have been based on interpolation (or filtering) [2] and, indeed, some digital printers already include such an interpolation facility for scaling. However, interpolation introduces distortions into dot array patterns and, moreover, in the machines where it is provided the proportional change of resolution is limited.

Interpolation is carried out as follows. Each pixel in the output is defined by means of a fixed mapping from a neighborhood of pixel values in the input pattern. One may visualize a window being placed over the input pattern at a location dependent on the coordinates of the output pixel to be evaluated. The pixel configuration that appears in the window determines the value assigned to the output pixel. The window is then moved to other locations and the process repeated in order to assign values to the remaining output pixels.

Inherent in this fixed-mapping procedure is a type of round-off error. If the window is moved by a small amount, the resulting pixel value can change. With binary inputs and outputs this phenomenon produces some undesirable effects (see Fig. 1). Corners that are squared in the input may be

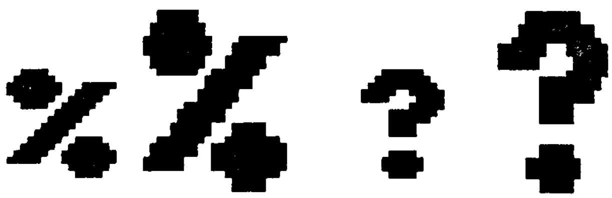


Figure 1 Scaling done with an interpolation filter.

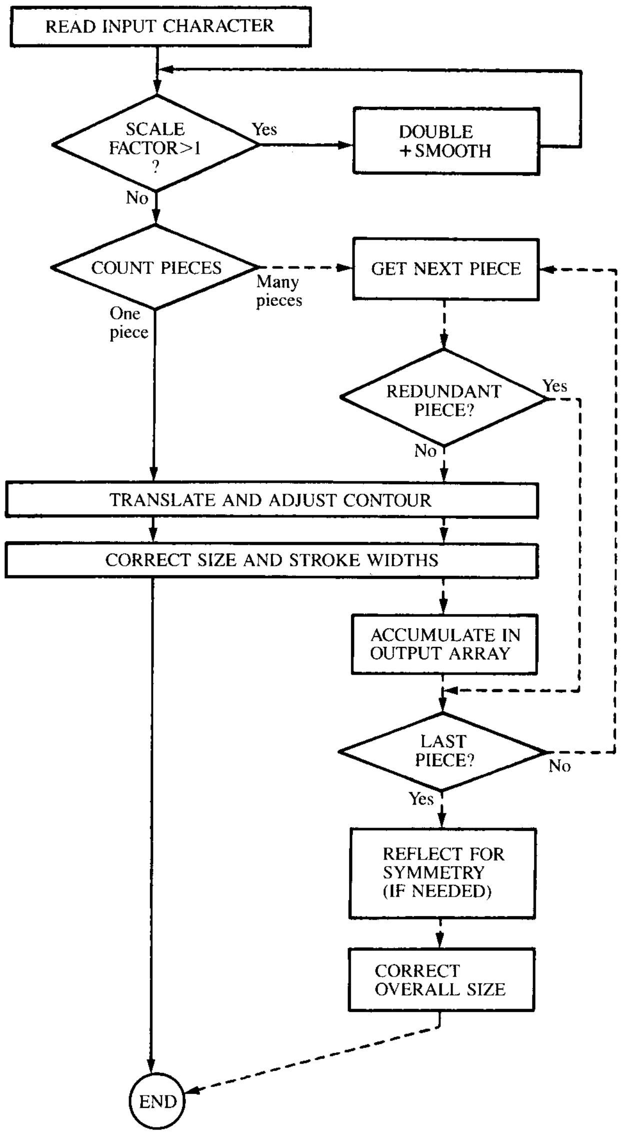


Figure 2 Flow chart of the new scaling method.

rounded in the output, or vice versa. Irregular steps may occur along an edge that was smooth in the inputs. Stroke widths may be inconsistent, symmetry may be lost, and other distinctive qualities of the input pattern may be badly distorted.

A scaling method based on pattern features

In the new scaling method described here, interpolation is augmented by techniques to distinguish attributes associated with print character quality in the input pattern, and those attributes are then preserved in the scaled character. Attributes that are detected and preserved during scaling include local and global symmetries, stroke width, sharpness of corners, and smoothness of contour.

The method has been considerably refined since first presented in [3] and has found several practical applications. A flow chart for the method is shown in Fig. 2. A large body of experimental results has been accumulated, including the scaling of a variety of high- and low-resolution fonts comprising several thousand characters, among which are Latin, Greek, Cyrillic (Russian), Kanji (Chinese), and Katakana (Japanese). Examples of these results are presented.

• *Operation over an arbitrary range*

The new scaling procedure operates over an arbitrary range of transformation ratios, permitting conversion of dot matrix characters from a given array size to any other array size, while retaining the essential properties of the original character. It has been used to magnify [4] characters by factors of 10 and more and to reduce them to as small as 20% of their original size. It has also been applied in cases where the horizontal scale change differed from the vertical scale change. This flexibility has been achieved in part by an initial magnification step and in part by a variety of techniques to track and maintain the correspondence between the input and output patterns.

In the initial step, the pattern is magnified by a power of two, if necessary, to cause it to become larger than the desired output pattern in both the x and y dimensions. The subsequent steps of the process then reduce the size of the magnified pattern to the exact dimensions required. A size-doubling algorithm has been developed which maintains all the desired characteristics of symmetry, stroke width, smoothness, etc. [3]. The algorithm replaces each pixel of the input pattern by a 2×2 array having the same white or black value, then smooths the contour to eliminate staircase effects. If a font is to be magnified, this algorithm is applied successively until both x and y dimensions exceed the specification, and then the appropriate reduction is applied. Figure 3(a) shows successive stages in the magnification of a Kanji character by an overall factor of 16.

A second general rule is that complex patterns are separated into constituent components which are then scaled individually (see Fig. 4). This permits symmetry and other features of the components to be maintained, and later the characteristics of the pattern as a whole are reconstituted by a careful reassembly procedure. As implemented, the separation is applied only to nontouching components; for example, an "i" is partitioned into its dot and stem. In principle, this concept can be extended to more complex decompositions.

- *Contour processing*

Once an input pattern has been magnified (if necessary) and analyzed into components (if any), the remainder of processing is done on the contour, leaving interior pixels to be filled in at the end. First, the coordinates of edge pixels are determined; then these are mapped into the row and column coordinates of the output resolution by a preliminary scale reduction step. This operation also produces a table that relates output contour pixels to input pixels so that in later steps, corresponding regions of the input and output contours can be compared and necessary corrections made.

A primary function of the reduction step is to ensure consistent stroke width scaling, which is important to the appearance of the final character. Horizontal or vertical strokes in the input pattern are detected and ranked according to length. A row-to-row mapping of horizontal strokes and a column-to-column mapping of vertical strokes are then constructed such that as many as possible of the strokes detected are properly scaled. The reduction step is also constrained to scale pattern height and width to the values computed by rounding off the input height and width multiplied by the respective scale ratios. In addition, symmetry properties of the input are detected and retained. A detailed description of the algorithm is presented in the Appendix.

Figure 3(b) shows an initial scaling for the Kanji pattern of Fig. 3(a), where the scaling factor is 11.7 times the input dimension. The mapping, as described above, operates on the successively doubled character that concludes the sequence in Fig. 3(a), and thus the actual scaling ratio for the reduction is 11.7/16. Note in Fig. 3(b) that stroke width is consistent relative to the input character.

While the initial scaling step deals with pattern stroke width and symmetry, it neglects local properties of the contour such as smoothness and curvature at sharp corners. Thus, with the input-to-output contour table as reference, the scaled pattern undergoes first a sequence of steps designed to reproduce directionality as closely as possible along the contour and, finally, several smoothing steps to ensure that corresponding lengths of arc are similar in appearance (see Appendix). During these procedures, the contours are represented not only as pixel coordinates, but also as a sequence of direction vectors from pixel to pixel along the contours. The latter representation, known as chain encoding [5], simplifies the problem of comparing corresponding sections of input and output and of making required changes. Figure 5 shows the changes made to the initial scaling for the Kanji example.

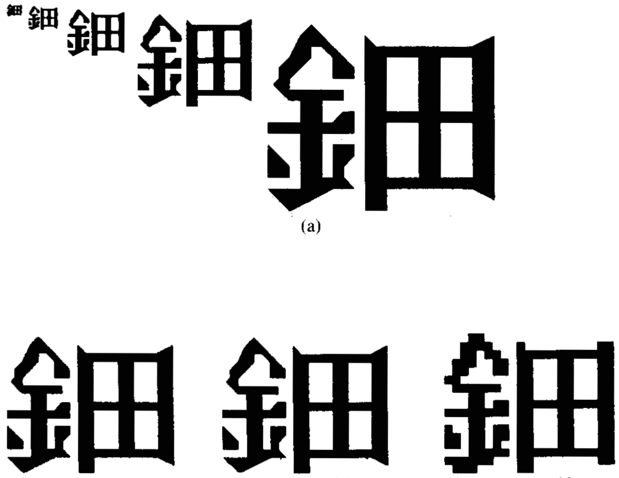


Figure 3 Example of the scaling procedure. (a) A Kanji character pattern successively doubled to a final magnification by a factor of 16. (b) An initial scaling of the magnified pattern by a factor of 11.7/16. The resulting pattern is 11.7 times larger than the original; it has proper stroke width but is not smooth and may differ from the original in fine detail. (c) The output pattern obtained after correction of (b). (d) A simple 12 × 12-pixel enlargement of the input for comparison.

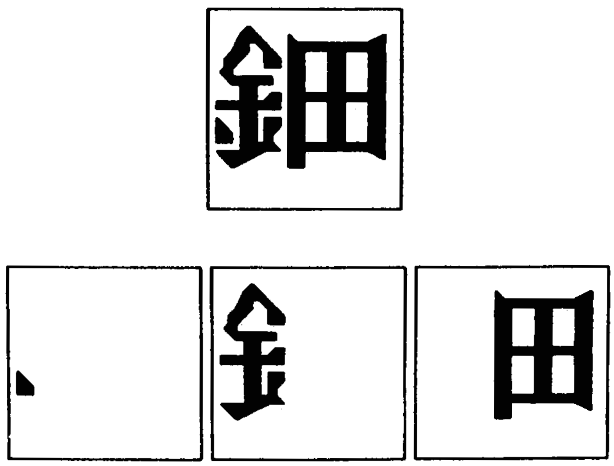
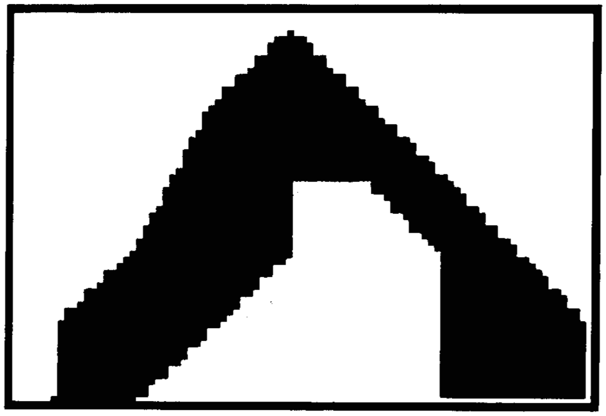
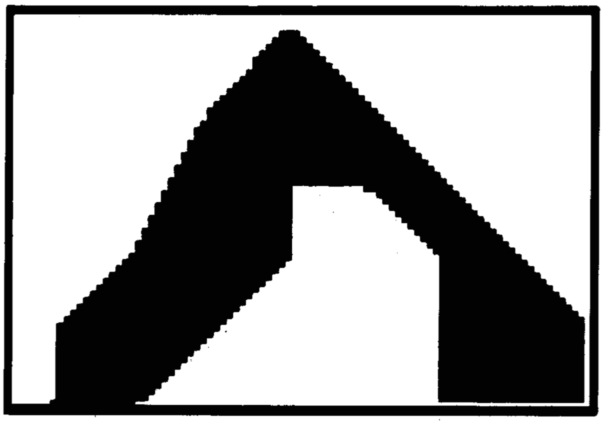


Figure 4 Decomposition of a character pattern into constituents. Each component is scaled separately, and the resulting patterns are assembled into a single output array.

When the contour processing of a pattern component has been completed, the contour pixels are plotted into an array and the interior pixels filled in with black. Other pattern components, if any, are processed in the same manner and assembled in the array, which becomes the output pattern of the scaling program. Figure 3(c) shows the completed Kanji character, scaled by a factor of 11.7. A direct 12-fold magnification of the character is also shown [Fig. 3(d)] in

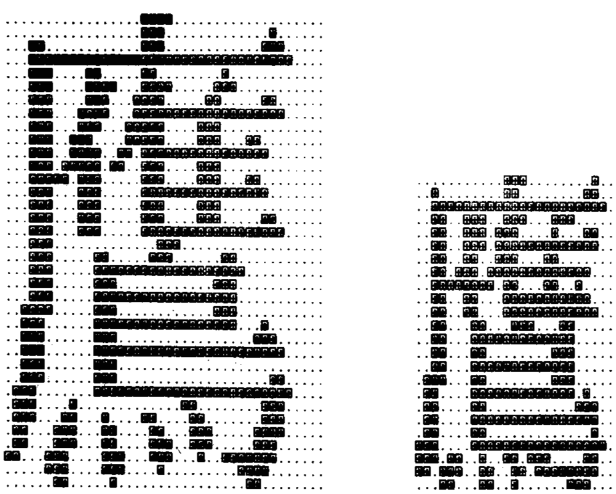


(a)



(b)

Figure 5 Detail of the final stages of the scaling process: (a) initial scaling, and (b) adjusted output.



(a) (b)

Figure 6 A complex character pattern that does not scale directly into the reduced array size. A faithful representation at 2/3 scale is not achievable.

order to illustrate the smoothing properties of the overall transformation. Note that the process tends to convert the single-pixel features at corners of the input pattern into triangular shapes at the output. The effect results from a design decision to treat single-pixel steps in the input contour as curved rather than square features, in the absence of knowledge of the font creator's real intent. For Kanji patterns in particular, which were originally derived from brush-stroked forms, this rule proves to be advantageous.

The scaling process is implemented in the APL language running on the VM system. The program is complex, consisting of several hundred APL functions, and requiring, for example, from 15 to 30 seconds of CPU time on an IBM 3033 computer in order to map a single high-resolution character from a 120 × 80 array into a 36 × 24 array.

Limitations

There are several inherent limitations to a general font scaling algorithm based on the prescribed input-output criteria.

First, the scaled font may not meet subjective aesthetic demands—for example, individual characters seem well shaped, but the font as a whole may not mesh well. In printed text, uniformity of character features is significant. More generally, good font designers are artists who are governed by aesthetic criteria that are difficult to codify in a set of computer commands. If the input to the scaling process is restricted to a dot matrix, as assumed here, without other parameters to express the designer's intent, then human participation may be imperative [6].

Nevertheless, an automatic scaling procedure can greatly reduce the human effort required. In certain applications, only a rough draft capability is needed, and the method may effectively be used without human intervention. In more demanding applications, the scaling operation can yield an initial character representation, to be modified by a designer if necessary.

Another limitation arises because the printer may not faithfully reproduce the pattern specified by the output array. Printer technologies have their own idiosyncrasies; particular printers, for example, may make character strokes either wider or thinner than specified by the dot array. To some extent the scaling procedure can be adjusted to suit a particular printer's behavior, *e.g.*, by setting a minimum width constraint on output strokes in order to adapt the method to a printer that tends to fade strokes. However, incorporation of constraints into the program is awkward at best, and visual review of printed text and subsequent manual revision of the font patterns is probably the wisest recourse in difficult cases.

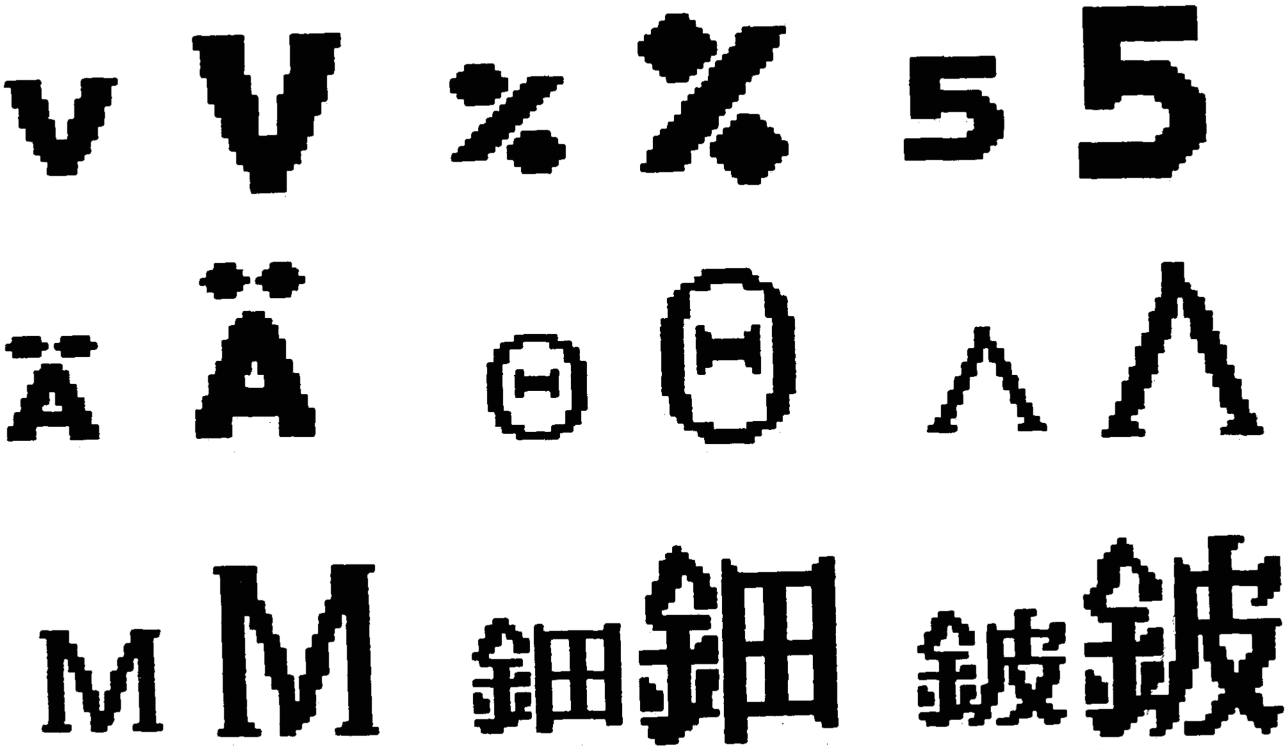


Figure 7 IBM 3800 character patterns scaled to 240-pixel/inch resolution.

It is also worth noting that pathological cases can occur in which an output meeting all the desired criteria is not possible. For example, Fig. 6(a) shows a Kanji character pattern meaning "hawk." When invoked to reduce this character by a factor of 2/3, the scaling routine produced Fig. 6(b). Inspection shows that it is simply not possible to maintain the calculated configuration of horizontal strokes while at the same time retaining small features appearing on these strokes, keeping separation between components, etc.

Experiments

The results shown below are illustrative of data obtained over a wide range of applications. To illustrate the experiments in the limited space of this paper, we show extracts from three classes of applications using Latin fonts. First is presented a series of non-square size increase conversions. Next are shown some size reductions from high-resolution input fonts. Finally, samples of converted output are shown in text form, permitting better evaluation of the scaling process over the font as a whole.

• *Magnification*

The IBM San Jose Research Laboratory is developing an experimental printer called *Sherpa* which has a programmable control unit and makes an IBM 6670 laser printer into an all-points-addressable printer that can plot images or composed text of arbitrary size. One class of output from Sherpa consists of data processing and office typing font styles. (Composed text output is discussed in the next section). A possible source of data processing fonts is a high-speed output printer, the IBM 3800. However, the 3800 prints at a resolution of 180 pixels/inch in the horizontal direction and 144 pixels/inch vertically, while the 6670 resolution is 240 pixels/inch in both directions. Therefore, the 3800 fonts must be scaled by 4/3 horizontally and by 5/3 vertically in order to appear at the proper size on Sherpa output.

Figure 7 illustrates a number of such conversions. The input dot array is shown on the left in each case, and the converted form on the right. Observe that the stroke size is consistent from character to character, as well as within a given sample.

• *Reduction*

Again the 240 pixel/inch Sherpa system is used as the target application. In this case, however, the object is to convert 800-pixel/inch photocomposer fonts to the lower resolution. This is a size reduction by a factor of 3/10 in each dimension.

Characters designed for a low-resolution printer such as the 3800 tend to be simple in shape, which alleviates the magnification problem since local variation along the con-

Figure 8 Photocomposer (high-resolution) characters scaled to 240 pixels/inch. Characters that are ordinarily thought of as symmetrical, *e.g.*, "pi," are often not symmetrical at photocomposer resolutions (here 800 pixels/inch). Thus the scaled version is not pixel-for-pixel symmetrical.

tour after scaling is small. In order to increase the array size of 3800 patterns, the primary requirement is to maintain symmetry and stroke width, while repeated contour smoothing is needed to minimize staircase effects. On the other hand, reducing the scale of high-resolution fonts involves different problems, since these fonts often contain serifs and decorative curvature. It is a challenge to the conversion algorithm to retain as many as possible of these features through the scaling process.

Figure 8 shows a number of photocomposer characters and the 240-pixel/inch patterns produced by the scaling algorithm. Note that the algorithm in some cases is forced by the array size constraint to compromise in the representation of local feature information.

Several other pertinent problems arose in these tests. The photocomposer characters contain "ink traps," extra black or white pixels placed at regions of sharp curvature in order to enhance the visual effect, but which are not actually part of the character pattern. The ink traps give the appearance of noise if retained through the conversion process, since each pixel carries 11 times as much weight at Sherpa resolution. Thus, a special filter algorithm was programmed to detect and remove the ink traps from the input arrays.

A more difficult matter is the tendency of Sherpa to print strokes somewhat finer than their array representations prescribe. Due to technical considerations in the laser printing process, a faithful scaling of a photocomposer character may appear to be broken in places where the stroke width narrows to a single pixel in width. A constraint was therefore imposed on the initial scaling routine to force vertical or horizontal strokes to be at least two pixels in width. This was surprisingly easy to do; indeed, the notion of such a constraint is almost inherent in the approach, since the thickness of each vertical or horizontal stroke is dictated by a conversion table. However, the thickness of curved or slanted strokes is more difficult to constrain. Manual editing of the patterns is possible at reasonable cost, since the problem occurs only sporadically. Another alternative that shows promise for Sherpa is a uniform thickening of the entire font by algorithm either before or after scaling.

• *Sample text*

In Fig. 9 are shown several examples of text using 800-pixel/inch photocomposer fonts scaled to Sherpa specifications (240 ppi). Composing characters into text poses the additional problem of scaling the parameters that govern the placement of characters with respect to each other in the lines of text. Text also reveals defects that may not be apparent when isolated characters are scaled, such as variations in height or in location with respect to the baseline of the text.

For example, the body of a lower-case "p" should rest on the baseline and its top should be at the same level as, say, the top of an "m," while the bottom should be at the level of the bottom of a "q." The conversion algorithm does not explicitly enforce all such constraints, though both overall height and baseline location are controlled. Nevertheless in most cases, as illustrated, an acceptable conversion is obtained without the added complexity that those constraints would entail.

Conclusions

It has been shown that by a combination of contour processing, feature detection, and smoothing, automatic digital font scaling can be accomplished while preserving essential pattern characteristics, such as symmetry, stroke width, smoothness, and local curvature. In scaling thousands of character patterns, relatively few have required human revision. The key to the transformation lies in the automatic recognition of attributes of font patterns associated with print character quality, permitting comparison of the input and scaled patterns to detect and correct discrepancies that occur due to the nature of the mapping task. The technique may thus be considered to be a type of pattern recognition procedure.

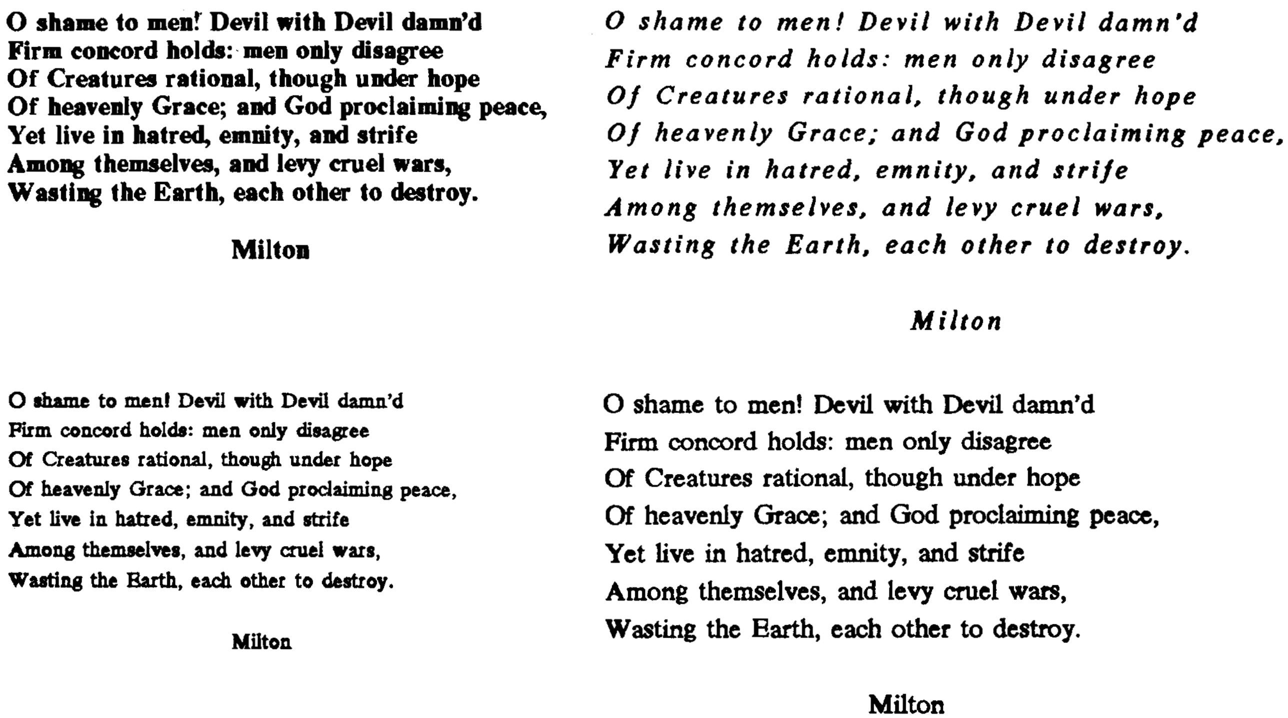


Figure 9 Scaled photocomposer characters composed into text. At the 240-pixel/inch resolution illustrated here a lower-case character such as "e" is only about 16 pixels high compared to over 50 pixels in the original font. The amount of detail captured at the coarser resolution is a good measure of the success of the scaling method.

Acknowledgment

A. Greene of IBM suggested the conversion problem and sponsored the algorithm development. J. King and K. Hitchcock of the Sherpa project provided data and programming that assisted in the evaluation and improvement of the scaling routines. D. Ngan of IBM Tucson helped in the application to 3800 fonts and gave feedback on performance. The support of these individuals and others who came into contact with the work reported here is gratefully acknowledged.

Appendix

• *An initial scaling that preserves stroke width and symmetry*

The problem considered here is to define two discrete mappings, T_x and T_y, that transform an input pattern P by scale factors r_x and r_y, respectively, to produce an output pattern P'. A contour pixel located at (x, y) in P is mapped into $(T_x(x), T_y(y))$ in P'. T_x and T_y are vectors of integers having lengths equal to the respective dimensions of P. We assume for convenience that P has at least one black pixel in each of its boundary columns and rows, so that its array dimensions are equal to its width and length. Also, $r_x \le 1$ and $r_y \le 1$.

The scale mappings are to be adjacency preserving, *i.e.*, for each integer $I <$ width of P, either $T_x(I + 1) = T_x(I)$ or else $T_x(I + 1) = T_x(I) + 1$.

The overall mappings are to be approximately linear, subject to the following three requirements:

1. *Consistent stroke widths* Let P contain a subpattern Q consisting of a vertical (or horizontal) black bar of width t. Then $T_x(Q)$ [or $T_y(Q)$] should have width $d_x(t)$ [or $d_y(t)$], where d_x and d_y are discrete mappings. d_x, d_y depend on the scale factors r_x, r_y and may be defined by a simple round-off rule as described in (3) below or by some other consistent formula.
2. *Symmetry* If there exist constants a and/or b such that one or more of the following relations holds in P, then there must exist constants a' and/or b' such that the corresponding relations hold in P':

horizontal symmetry:	$P(x, y) = P(a - x, y)$
vertical symmetry:	$P(x, y) = P(x, b - y)$
diagonal symmetry:	$P(x, y) = P(a - x, b - y)$.

3. *Overall dimensions* If (w, h) are the width and height of P, then the corresponding dimensions (w', h') of P' should satisfy the usual round-off formulas $w' = \text{trunc}\, 0.5 + r_x)w$ and $h' = \text{trunc}\, 0.5 + r_y h$, where trunc z is defined to be the next integer less than or equal to z.

These constraints preclude use of a simple scaling and round-off rule to determine T_x and T_y. For example, the specification

$$T_x(I) = \text{trunc}\, c_x + r_x x$$

affords control only over a single parameter c_x in order to meet all constraints. Simple examples suffice to show that this is not possible.

In fact, if P contains many strokes, it may not be possible to define a scaling transformation that maps all the strokes so as to satisfy (1) above. In such cases the method described here is made to fulfill constraints (2) and (3), but to satisfy (1) only for a selected subset of the strokes in P.

A further difficulty must also be resolved in the case of symmetrical patterns. Suppose, for example, that w, the width of the input pattern, is an odd number while the scaled pattern width w' is even. In this case, P is said to have *odd symmetry,* while P' is said to have *even symmetry* in the horizontal direction. These conditions imply that P' is made up of pairs of matching columns, while the center column of P has no symmetrical counterpart. The column in P' to which this column is mapped by an arbitrary T_x cannot be guaranteed to match the other member of its symmetry pair; *i.e.*, it may not be possible to satisfy (3).

To prevent this problem from occurring, while still retaining generality, at the start of the algorithm P is tested for symmetry, and its height and width are measured. If either w or h is odd, and if the pattern is symmetrical in the corresponding direction, then the doubling algorithm (see text) is invoked to produce a pattern having even height and width. The scale factors are halved, and the process continues as if the doubled pattern were the input.

T_x and T_y can be constructed independently of one another; *i.e.*, (1)–(3) can be resolved into one set of constraints involving only T_x and another set involving only T_y. Since they are determined in an identical manner, we describe the specification only of T_x.

The formation of T_x proceeds in the following steps:

1. A stroke table is calculated, listing the locations and sizes of the strokes to be scaled by T_x.
2. The stroke table is reduced to denote only a subset of "admissible" strokes.
3. The stroke table is used to create a scale interval table, S, giving the mapping from each interval width in P to a corresponding width in P'.
4. T_x is formed from the specification in S.

A more detailed description of these steps follows.

- *Stroke table*

Note that T_x controls the width of vertical strokes in P'. A vertical bar of width t in P consists of a number of sequences of t 1's that occupy the same columns in consecutive rows of P. Such bars may be interrupted by intersections with other strokes and by the presence of serifs on the contour. The algorithm used for stroke detection finds all sequences of 1's in P and tabulates those having a common length and initial column. A table is formed whose rows list for each such collection (1) the initial column, (2) the length t, and (3) the number of bit sequences represented in this entry. The rows of this stroke table having highest weight (the frequency of bit sequence occurrence recorded in column 3 generally pertain to strokes in the input pattern.

Next, using d_x as described above, the width desired for each stroke is added to the table as a fourth column. Pattern P is then complemented and the procedure repeated to tabulate the position, width, weight, and desired width after scaling for white bars. A fifth column is appended to each table to indicate 1 for black strokes and 0 for white, and a master table containing both sets of data is assembled and sorted on stroke weight. A bottom row is appended, containing in columns 2 and 4 the value of the overall width of P and the desired width of P', respectively.

- *Reduction of the stroke table*

T_x is primarily intended to set the correct black stroke width and overall width of P'. As a secondary objective it is concerned with maintaining the separation between closely spaced strokes. This is the reason for recording white strokes as well as black in the table. As an initial reduction, then, any row describing a white stroke is deleted from the table if the value in column 4 (desired width in P') exceeds 3. A second reduction is effected by selecting from the table a sequence of rows describing strokes (either white or black) that occupy nonoverlapping columns. The selection is done iteratively in order of stroke weight, and the selected rows are stored as a new table. The bottom row of the original table, containing the overall stroke width specification, is also placed at the end of this table.

- *Interval table*

The reduced stroke table, R, is ordered on column 1 (x-coordinate of the left edge of a stroke) and used to calculate the scale interval table, S. S is an array having 2 rows and $2M + 1$ columns, where M is the number of strokes identified in R. Column $2I$ of S (*i.e.*, the Ith even-numbered column) contains the input and output stroke widths, $R(I, 2)$ and $R(I, 4)$, respectively.

The remaining $M + 1$ odd-numbered columns in S describe the space between strokes. The first row of column $2I - 1$, where $I = 2, 3, \cdots, M$, contains the quantity

$$R(I, 1) + 1 - R(I - 1, 1) - R(I - 1, 2),$$

which is the number of pixels between the end of the $(I - 1)$th stroke and the start of the Ith. The second row is computed as

$$S(2, 2I - 1) = \text{trunc}\, 0.5 + (r_x)S(1, 2I - 1).$$

That is, the second row entries are calculated by scaling and rounding off the first row values. Columns 1 and $2M + 1$ contain similar data, except that the intervals described run from the left edge of P to the left edge of the first stroke and from the right edge of the last stroke to the right edge of P, respectively.

S describes the mapping from successive horizontal intervals of P to intervals in P'. The entries in row 2 of the odd-numbered columns are free; *i.e.*, they are adjustable to meet certain criteria. The even-numbered entries are fixed by the stroke width constraints.

S is next adjusted, if necessary, in order to satisfy the following requirements. Let $S_1(I), S_2(I)$ be the Ith elements in rows 1 and 2 of S, respectively. Then,

1. $\sum_{I=1}^{2M-1} S_2(I) = W'$, where W' = the desired width of P'.
2. $0 \le S_2(I) \le S_1(I)$.
3. If $S_1(I) > 0$, then $S_2 > 0$.
4. For I odd, $|S_2(I) - r_x S_1(I)| \le 1 + c\, r_x S_1(I)$,

where c is an arbitrary constant less than 1.

Constraint (1) maintains the desired overall width of P', while (2) is a feasibility condition. Constraint (3) maintains separation between successive strokes in P' if there exists separation in P. The last constraint (with $c = 0.25$ in the experiments reported in the text) determines the amount by which the space between strokes is allowed to differ from strict linear scaling. This requirement enforces "local" linearity to ensure that T_x does not overly expand or compress the space between strokes. The smaller c, the less variation permitted, but at least one pixel of variation from linear scaling is always allowed.

Initially S_2 will meet constraints (2) and (4). If (3) is violated, then elements of S_2 must be increased from 0 to 1. When this is done, all that remains is to satisfy the overall width constraint (1) by adjusting the odd-numbered elements of S_2 within the limits imposed by (4).

The direction of change required is determined; then, for each odd I the boundary for $S_2(I)$ in that direction is calculated. This limit, obtained by combining (2)–(4), is

$$\max\{1, r_x(1 - c)S_1(I) - 1\} \le S_2(I)$$
$$\le \min\{S_1(I), r_x(1 - c)S_1(I) + 1\},$$

for all I such that $S_1(I) > 0$.

The element $S_2(I)$ that is furthest from the constraint boundary is determined, and if this distance is at least unity, then $S_2(I)$ is increased or decreased by 1 as needed. This procedure is repeated until either (1) is satisfied or else no element of S_2 can be altered without violating the boundary constraints. In the former case, the specification of S is complete. In the latter case, the constraints cannot be met. The algorithm then returns to the stroke table R and deletes the row describing the stroke of least weight. The reduced stroke table is used to specify a new S table as above. In this way, a stroke table is eventually obtained that produces an S table satisfying all conditions (1)–(4).

- *Calculation of T_x from S*

Consider the 2-tuple (S_1, S_2) contained in an arbitrary column of S. By the usual division algorithm, if $S_1 > 0$, then we can write

$$S_1 = pS_2 + q,$$

where p, q are integers such that $q < S_2$ and $p > 0$. Thus the set of integers $1, 2, 3, \cdots, S_1$ can be partitioned into S_2 subintervals, such that $(S_2 - q)$ of the subintervals contain p consecutive integers, while q of the subintervals contain $p + 1$ consecutive integers.

An algorithm for generating the sequence of subinterval lengths is the following. Let $m = \text{trunc}\,(S_2/2)$. Construct the sequence of S_1 integers given by

$$v_i = m + (i - 1)S_2 \qquad i = 1, 2, \cdots, S_1.$$

Now define

$$f(v_i) = 1 + \text{trunc}\,(v_i/S_1)$$

and

$$l_j = |\{i: f(v_i) = j\}| \qquad j = 1, 2, \cdots, S_2.$$

The subintervals so determined correspond to a linear scaling and round-off of the interval of P corresponding to S_1. Note that the scale factor is not r_x, but S_1/S_2.

The procedure just described is repeated for each column of S, generating for the Ith column the sequence of subinterval lengths $l_j(I)$, where $j = 1, 2, \cdots, S_2(I)$. These are next arranged in order of increasing I to give the vector w defined as

$$w = 0, l_1(1), l_2(1), \cdots, l_{S_2(1)}(1), l_1(2), \cdots, l_{S_2(2M+1)}.$$

Let us call g_k the kth element of this sequence. The desired mapping T_x is obtained by setting

$$T_x\left(\sum_{h=1}^{k} g_h + j\right) = k + 1 \text{ for all } j = 1, 2, \cdots, g_{k+1}.$$

If P is symmetrical, then the right-hand side of T_x beyond the axis of symmetry is deleted. T_x is then completed by reflecting the left-hand side (*i.e.*, reversing the order of its elements), adding a value determined from the axis of symmetry, and concatenating the resultant to the left-hand side.

References and notes

1. P. Archibald, *Fonts Manual*, Version 2, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, September 1977.
2. R. Schafer and L. Rabiner, "A Digital Processing Approach to Interpolation," *Proc. IEEE* **61**, 692–702 (June 1973).
3. R. G. Casey, T. D. Friedman, and K. Y. Wong, "Use of Pattern Processing Techniques for Rescaling Digital Print Fonts," *Proc. 4th International Conf. on Pattern Recognition*, Miami Beach, FL, December 1–4, 1980.
4. Converting a pattern to a new resolution involves exactly the same process as changing its size, and the two operations are used interchangeably in this paper. Converting a pattern from, say, a coarse to a fine grid resolution while retaining the pattern's original size requires that the number of dot elements in the pattern be increased. This transformation is equivalent to enlarging the size of a pattern in a resolution whose grid size is fixed. It should be noted that in practice, designers alter relative proportions as they change the size of characters. This artistic rule has not been included in the present work.
5. H. Freeman, "On the Encoding of Arbitrary Geometric Configurations," *IEEE Trans. Electron. Computers* **EC-10**, 260–268 (June 1961).
6. Some scaling techniques, *e.g.*, Knuth's [7], call for a complex and explicit description of the character pattern, thereby avoiding the need to derive qualitative attributes of patterns as in the approach reported here. These systems, however, require that the font be created manually at a computer console to begin with in order to establish the parameters, and thus are not directly suitable for scaling a pre-existing font represented only as a set of dot arrays.
7. D. Knuth, *TEX and METAFONT, New Directions in Typesetting*, Digital Press, Bedford, MA, 1979.

Received May 6, 1982; revised June 29, 1982

Richard G. Casey *IBM Research Division, 5600 Cottle Road, San Jose, California 95193.*

(See page 656 for biography.)

Theodore D. Friedman *IBM Research Division, 5600 Cottle Road, San Jose, California 95193.*

Dr. Friedman joined the IBM Thomas J. Watson Research Center in 1963 and in 1966 became manager of the machine-assisted design project, producing the ALERT system, the first logic design compiler. In 1970 he transferred to the IBM San Jose Research laboratory, where he developed analysis tools for digital networks and devised an access control system for shared data. With Dr. R. G. Casey, he developed a facsimile compression method using an extendable decision-tree search technique. He also helped devise and implement a general-purpose digital image enhancement algorithm. Currently, Dr. Friedman is working on computer-aided design of VLSI. Prior to joining IBM, he was a member of the scientific staff at Technical Research Group, Inc., where he was responsible for the design specification of a space vehicle computer. He received his B.A. from the University of Michigan in 1958 and his M.S. and Ph.D. degrees in computer science from the University of California, Berkeley, in 1973 and 1976.

Kwan Y. Wong *IBM Research Division, 5600 Cottle Road, San Jose, California 95193.*

(See page 656 for biography.)