Extraction of Typographic Elements from Outline Representations of Fonts

Ariel Shamir and Ari Rappoport

Institute of Computer Science, The Hebrew University of Jerusalem
Jerusalem 91904, Israel. {arik,arir}@cs.huji.ac.il

Abstract
Digital typefaces for computer graphics and multimedia applications must be capable of supporting operations such as font variations, transformations, deformations and blending. A powerful implementation of such operations must rely on the inherent typographic attributes of the typeface. However, even today's most advanced typeface representations support only geometric outline representations and basic font variations.
In this paper we discuss high-level typeface representations which we term Parametric Typographic Representations (PTRs). *We present an algorithm for automatically extracting typographic elements of typefaces from their outline representation, which is an essential initial step in converting typefaces from outline representations to PTRs. The extracted typographic elements include serifs, bars, stems, slants, bows, arcs, curve stems and curve bars. Most notable is the treatment of serifs, which are represented by finite-automata. The algorithm only needs to learn a serif type once, and is then capable of automatically recognizing it in different typefaces. We show an application of a PTR for automatic high-quality hinting of fonts, which is one of the most important stages in digital font production. Our system was used to generate hints for dozens of thousands of Kanji, Roman and Hebrew characters.*

Keywords: Digital typography; outline fonts; typographic elements; hinting; parametric typographic representations

1. Introduction

Printed pages have constituted a major way of communication between literate people ever since the invention of print by Gütenberg in the 15th century. The existence of electronic media has modified this situation. Typsetting, page layout, typefaces and fonts need to change their nature and become more flexible. A wide range of applications such as multimedia publishing, computer animation, and even mechanical or electronic computer-aided design systems must incorporate a diverse set of text manipulation functions, from simple affine transformations to advanced deformations. The basic technology which supports this functionality is encapsulated in the representation used for digital typographic typefaces, or digital fonts. Hence, the degree of sophistication of digital typeface representations is a crucial factor in the expressive power and usability of modern visual communication.

In the design and analysis of typefaces one uses terms such as stems, bars, serifs, style, size, weight and width [Lawson90, Rubinstein88, Bringhurst92, Bauermeister88] (see the figures in this paper for examples). However, even the advanced typeface representations in use today (e.g. QuickDraw-GX [Apple94], TrueType and TrueType Open [Microsoft90], Type 1 [Adobe90, Adobe92]) concentrate only on representing the geometry of a font and on the provision of relatively simple font variations. There is a large gap between the high-level terms used by designers and between current representations. A representation which supports high-level parameterization based on typographic terminology is needed. The general term which we use for such a representation is a *Parametric Typographic Representation (PTR).* Extraction of typographic elements from existing font representations is essential in order to convert today's outline representations into any parametric typographic representation.

1.1. Previous Work

Some previous work of identifying typographic elements has been done for the sake of automating the process of hinting an outline font ('hints' are needed in order to rasterize scaled fonts correctly, whether on a screen or in print). Basic hints can be added to the outline description automatically by recognizing horizontal and vertical bars and curvilinear shape extrema [Andler90], or even some serif parts [Karow89].

Hersch and Bétrisey have presented an elaborate automatic hinting method [Hersch91a] which requires a topological font-independent model for the description of each Roman letter shape. These models include specific hints connected to various typographic elements. In order to add hints to a concrete letter shape, a matching algorithm between the real shape and the models is performed. After a match has been found, the hints contained in the matched model are adapted to the real letter shape by identifying similar points in both the model and the actual letter shape. This scheme involves the definition of a model for each topology of a letter shape, a process which is both difficult and time and space consuming, for example in ideographic (Chinese, Japanese and Korean) scripts.

Several other techniques for automatic hinting of outlined characters exist inside the font development systems of commercial companies such as Bitstream, Adobe and Apple. These systems are considered a commercial secret, but by examining their output (e.g. TrueType or Type 1 fonts) one can see that a large amount of typographic information is still not identified by the automatic hinting process, and therefore a lot of manual proofing and hinting is taking place in order to achieve the desired quality.

Some research has been done on the extraction of strokes from the outline description, for applications such as display of Kanji characters [Chialing89], conversion to a different representation [Dürst93a, Dürst93b], and optical character recognition [Feng75]. The general spirit of these papers is opposite to ours, since they are interested in removing the subtle typographic details of typefaces in order to create a skeleton representation of characters, which is simpler and easier to recognize.

A font represented in METAFONT [Knuth86] is in principle parametric, since METAFONT is a procedural programming language. However, all the parameterization is done manually by the programmer. Conversion of PostScript fonts to METAFONT has been described in [Haralambous93], but the method is specific to these two representations and it does not treat automatic extraction of typographic elements.

1.2. Contribution

In this paper we present an algorithm to extract typographic elements of typefaces from an outline representation of a font and show how to use its results for automatic hinting. The conversion algorithm extracts basic typographic elements from the outline description of a character (Figure 7) and can gather information regarding the relationships between them, both inside each character and across the font. This information can then be used to assign parametric attributes to the typographic elements. With this capability, the algorithm can be viewed as an essential initial step in converting a typeface from an outline representation into a parametric typographic representation.

Our conversion method involves two main stages. The first stage is the designation and classification of characteristic points along the glyph outline, which is similar in spirit to some previous work [Hersch91a], but is given here in a more detailed manner. The second stage uses data from the first stage for the actual extraction of the typographic features from the outline. Among the basic features recognized are stems and bars, bows and arcs, curve stems and curve bars, slants, extrema, and, most notably, serifs of all types. These typographic features are in turn gathered to create higher degree elements such as groups of bars inside a glyph or a Kanji stroke-like element [Zhang92].

Special treatment has been given to the extraction of serifs. Serifs play a crucially important role in the design of a typeface, and there is an enormous diversity of serif designs. We create a finite-automaton [Lewis81] that defines the sequence or sequences of points which characterize each type of serif. Once this automaton has been defined, serifs of this type can be extracted and recognized in any outlined font input to the system. If a new typeface containing new types of serifs is introduced to the system, only a few new finite-automata need to be defined in order to extract the new typeface features.

Along with a raster-to-outline module, our system can convert a typeface in any lower-level description technique to a higher level parametric typographic representation. We do not require a model for each glyph,

since the information regarding typographic elements is stored in a higher representational level that applies to any glyph. The algorithm is time and space efficient: if n is the number of points defining the outline of a glyph and k is the number of features extracted from it (where $k << n$), the process for recognizing basic elements for one glyph takes $O(n \log n + k^2)$ time and $O(n + k)$ space.

Our method needs much less manual manipulation and proofing than previous work; it is multi-lingual in nature, not using language-dependent letter models; and it produces higher level typographic details suitable to modern visual communication applications.

The problem dealt with in this paper is very similar to the problem of *feature recognition* extensively researched in geometric and solid modeling [Woodwark92]. In both cases we desire to convert a boundary representation (Brep), which is a relatively low-level representation, to a higher-level one supporting terms taken from the application domain.

In Section 2 we describe the state of the art in digital typography today and the hierarchy of digital typeface representations. Section 3 describes the classification of the points describing the outline. Section 4 describes the actual extraction of the typographic elements. Finally, in Section 5 we discuss an important application of the resulting parametric typographic representation, namely automatic hinting of fonts.

2. Hierarchy of Digital Typeface Representations

Examination of the different representations available today in digital typography reveals a hierarchy in the degree of abstraction concerning the freedom to modify the font at the level of the typographic design axes [Rubinstein88, Karow94a, Karow94b, Southall91].

The hierarchy begins with bitmap fonts. A bitmap font is a collection of matrices of dots in a fixed size whose appearance corresponds to the original typeface design. Bitmaps are used in order to create the image of a character in most output systems today including phototypesetters printers and CRT screens. Therefore, every higher level representation should support its eventual conversion to a bitmap. Bitmaps are also the most constrained representation having a fixed coordinate in all axes of the typographic taxonomy, and almost no possibility for an appropriate variation in any dimension. The Macintosh and Windows operating systems have algorithms for slanting, condensing, bolding etc. of bitmap fonts but the outcome is very poor in quality.

Digital outline fonts were created mainly to solve the fixed size constraint in bitmap fonts (Figure 1). They hold the ability to create bitmaps in any size using some special process of scaling, gridfitting, rasterizing and filling [Rubinstein88, Karow87, Microsoft90, Adobe90, Southall91]. However, many have found that simply using an outline description of a typeface does not suffice in order to perform the task of bitmap creation [Rubinstein88, Karow89, Hersch87, Bétrisey89, Hersch91a, Hersch91b, Karow89]. Additional information in the form of 'hints' has been added to the font. Hints are special constraints that help the grid fitted outline, and thus the bitmap created from it, retain some typographic attributes (e.g., equality of stems, height of character, symmetry of serif) [Microsoft90, Adobe90, Hersch91a, Karow89, Changyuan93].

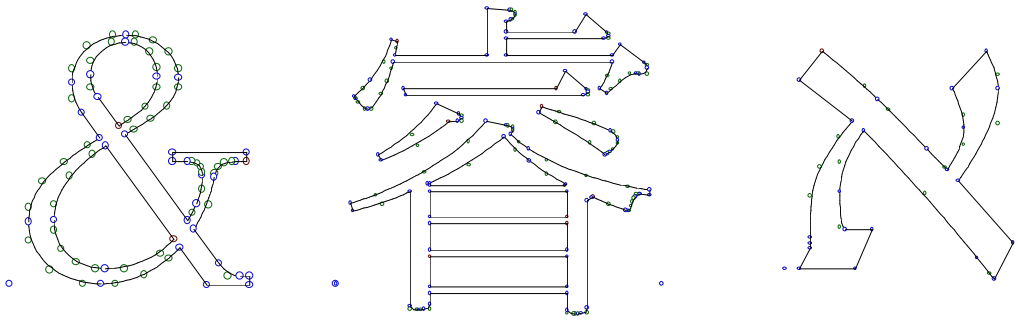

Figure 1: *Outline of characters from (left to right) Roman, Kanji and Hebrew fonts, including their defining control points.*

Outlines can also be used successfully in order to rotate text or to create outlined or slanted text subject to

special distortion algorithms which consider some more typographic constraints and probably using hints. This suggests that outline fonts could carry the ability to vary along any dimension in the typographic taxonomy. Attempts have been made to use several outline representation instances of one typeface in order to create a representation with the ability to vary along one or more dimensions using interpolation between outlines or extrapolation from one outline [Apple94, Adobe92]. The conclusion of these attempts has been that in order to succeed some more information concerning the typographic attributes of the typeface should be added to the outline representation, and later on used in the deformation algorithms.

Several other description techniques for digital fonts have been defined with the intention of remaining as much as possible in the typographic domain and saving as much information as possible while converting the designer images to the actual description of the font. Character parts in these representations are linked to their typographic definitions such as a stem, an arc, a dot or a serif. They include typographic attributes such as width, angle, special optical correction or offset values, which in turn could be parametrized creating variation instances of the typeface [Knuth86, Adams89, Stamm93a, Stamm93b, McQueen93]. We use the term **Parametric Typographic Representation (PTR)** when referring to this class of digital typeface representations.

A major problem of previous work on PTR techniques is that it deals mainly with the creation or definition of new fonts, disregarding the huge wealth of existing fonts already defined or converted to the outline form of representation. Conversion techniques such as [Haralambous93] rely on manually recognizing and inserting parametric values to the outline, which is very time-consuming and requires special skills.

3. Classification of Points

The first stage in the conversion from an outline to a PTR consists of geometric and topological analysis of the outline. Although there are several different methods for representing an outline of a glyph, almost all of them share some basic characteristics. A glyph outline is a closed path consisting of consecutive segments of two types: straight line segments and curve segments (see Figure 2, lines segments are bounded by squares and curve segments are bounded in both sides by circles). Each segment is uniquely defined by its start and end points which lie on the outline of the glyph. The points are numbered consecutively, and the direction of the outline at each point is the direction of travel from lower indexed point to higher indexed points.

Curve segments include some additional representation-dependent control points. These points lie outside the outline and do not include any special information regarding the topological nature of the outline therefore we do not include them in our classification. These points are important in the process of converting an outline representation from one form to another (e.g. cubic Bézier in Type 1 fonts to Quadratic B-Splines in TrueType fonts). This involves converting only curve segments defined by a set of some control points to the same (or approximated) segments defined by a set of other control points.

In order to perform the typographic decomposition of the outline properly, certain points have to be included in the glyph's outline. Such points are local maxima or minima in the X or Y direction, inflection points, points where a discontinuity of derivative occurs or tangent points (see definitions below). Often these points exist in the outline and are used to define its segments. However, in order to make sure that no point is missing and in order to correct misplaced points, the outline description of glyphs go through a well known preparation stage used often after digitization [Karow89]. Hence, the first degree classification of the outline points borrows its definitions from the digitization process (Figure 2):

- A **corner** point (denoted as C point) is a point where two outline segments meet with discontinuity of the derivative of the outline.
- A **curve** point (denoted as CT point) is a point where two curve segments meet with continuity of the derivative.
- A **tangent** point (denoted as T point) is a point where a curve segment and a line segment having the direction of the derivative of the curve meet.

The only neglected pair in these definitions is the case of two line segments meeting with continuous derivative. It is safe to say that for almost all cases this situation is undesirable, because it carries redundant information, and the preparation stage of the outline would recognize such cases and replaces the two line segments by a single one. A case where this information should not be discarded is, for example, when two master glyphs are being matched in order to create interpolated instances between them. It could be that in

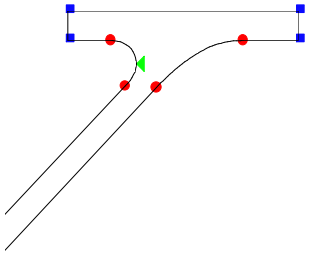

Figure 2: *Classification of points: corners points (squares), tangents points (circles), curve point (triangle).*

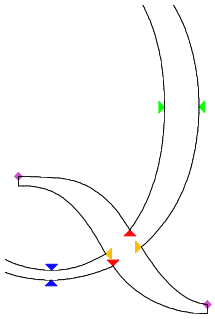

Figure 3: *Further classification of points: H points (dark in-facing triangles), V points (light in-facing triangles), CH points (dark out-facing triangles), CV points (light out-facing triangles), VR points (rhombus).*

one of them this point is in fact a corner point (discontinuity) and in the other the derivative of the segments are continuous. Discarding this point in one of the glyphs would make it hard to match these glyphs and to create the interpolations.

Second degree classification is further applied to the point in order to extract geometric properties (Figure 3):

- A **curve V-extremum** (V point) is a curve point which is a local extremum in the x direction.
- A **curve H-extremum** (H point) is a curve point which is a local extremum in the y direction.
- An **inflection point** (I point) is a curve point which is an inflection point.
- A **corner V-extremum** (CV point) is a corner point which is a local extremum in the x direction.
- A **corner H-extremum** (CH point) is a corner point which is a local extremum in the y direction.
- A **spike** (S point) is a point which is both a x-extreme and a y-extreme.
- A **V-corner** (VR point) is a corner with one of its neighboring segments defined as a vertical line segment.
- A **H-corner** (HR point) is a corner with one of its neighboring segments defined as a horizontal line segment.
- An **L-corner** (LR point) is a corner which is both a V-corner and an H-corner.

One important distinction should be made in these definitions between 'real' geometric properties and 'defined' geometric properties. The difference lies in the notion of tolerance. For example, the real definition of vertical segments would be:

Definition 1 Segment s is vertical if it is a line segment and its start and end points have the same x coordinate.

When we refer to *defined vertical segments* we generalize the previous definition to be:

Definition 2 Segment s is considered vertical if it is a line segment and for some predefined epsilon ϵ, the x coordinate of its start and end points do not differ by more than ϵ.

For certain type of typefaces, where straight line segments are rare (Figure 4), a further generalization is needed:

Definition 3 Segment s is considered vertical if for some predefined epsilon ϵ, the width (xmax-xmin) of its bounding box is not greater than or equal to ϵ.

We will call these kinds of generalizations the **epsilon rule**. Its implementation in geometric definition is crucial to the success of feature extraction both from non-regularized typefaces and from subtly designed typefaces.

Beside the type of a point, the additional information classified for each point includes its coordinates, the type of segments meeting at the point and the direction of these segments.

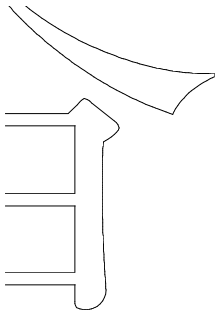

Figure 4: *Part of the outline of a character from the Macintosh Sai Mincho font. Notice that the stem is made of a left vertical line and a right vertical curve.*

4. Extraction of Typographic Elements

The second stage of conversion to a PTR is the actual extraction of typographic elements. This stage uses all the data acquired in the first stage when the topological and geometric analysis of the outline was performed. Every typographic element is defined in terms of the types of points and segments building it and the relationships between them. This section presents these definitions for each typographic element, and discusses the algorithm for recognizing them in the outline.

All upcoming definitions are subject to the epsilon rule stated earlier. Not only that the definitions of vertical or horizontal segments, the definition of equality of angles or sizes, the definition of the significant size of overlaps, etc. are all within a certain parameter epsilon, in fact the definition of a segment may also vary in turn and consist of several continuous or even broken segments with the same direction.

Color Figure 8 exemplifies the terms in this section.

4.1. Stems, Bars and Slants

- A **stem** consists of two vertical segments with opposite directions overlapping each other vertically with the space between the overlaps considered internal to the outline.
- A **bar** consists of two horizontal segments with opposite directions overlapping each other horizontally with the space between the overlaps considered internal to the outline.
- A **slant** consists of two slanted segments having the same angle, with opposite directions overlapping each other and the space between the overlaps is considered internal to the outline.

The algorithm for finding such elements is quite straightforward: we first find the segments and then match opposite direction segments. For example, the stems and bars extraction was defined for Japanese characters in [Chialing89].

Slants might present some complication because of the variety of angles. We used a data structure sorted by angle to insert non vertical or horizontal segments and match groups of lines created within a certain range of angles.

4.2. Bows and Arcs

- A **bow** consists of two V-extreme points having the same y coordinate and the same extreme type (min or max), the outline passing through them having opposite directions, and the line between them considered internal to the outline.
- An **arc** consists of two H-extreme points having the same x coordinate and the same extreme type (min or max), the outline passing through them having opposite directions, and the line between them considered internal to the outline.

Finding bows and arcs is done by finding all x (or y) extremum points, sorting them by y (or x) coordinate and matching them to each other within the epsilon ranges.

4.3. Curve Stems and Curve Bars

- A **curve stem** is a V-extreme point and a vertical segment at the side of the extreme (on the left if the V-extreme point is a minimum and on the right if it is a maximum) overlapping in some y coordinate, the outline passing through them having opposite directions, and the line between them considered internal to the outline.
- A **curve bar** is a H-extreme point and a horizontal segment at the side of the extreme (lower if minimum and higher if maximum) overlapping in some x coordinate, the outline passing through them having opposite directions, and the line between them considered internal to the outline.

The same sorted data structure having x (or y) extreme points and vertical (or horizontal) segments can be used to match extreme points to straight segments and extract curve stems and curve bars.

Actually, the three algorithms for finding stems, bars, arcs, bows, curve stems and curve bars are executed all in one pass. First all the basic features like extreme points and vertical or horizontal segments (or multi-segments) are inserted to a data structure sorted by x or y coordinate and then the matching is done to find the typographic elements. The complexity of the whole process is therefore $O(n \log n)$ where n is the number of points in the outline. The dominant part is the sorting stage.

4.4. Serifs

Extraction of serif parts is a powerful feature of our algorithm. Serifs are one of the most important elements in typographic design; previous works have usually not addressed this issue completely.

All other typographic elements (e.g. bars, stems, arcs) could be defined geometrically with acceptable variations incorporated in the epsilon rule. Serifs could carry almost any geometric shape and the rules whether a serif is included or not in certain parts of the character may vary dramatically from one language to another. For example, all bars in Japanese Mincho-style characters possess a serif at the right end. On the other hand, Hebrew serifs are located only at the left end of the topmost bar in a character. Therefore, there is no way to distinguish and recognize a serif without prior knowledge of the design of a font.

We define a serif as a sequence (or several possible sequences) of points having certain predefined types according to the design of the serif. For example, a Mincho style bar serif and a Roman style stem serif could be defined as seen in Figures 5 and 6.

The identification of a certain serif is done by using a finite automaton [Lewis81] built to recognize the sequence of points defining the serif. The outline is traversed and the types of the points encountered are fed by order of appearance to the different finite automata for different serif types. Whenever an automaton has reached a final accepting state, it means that its serif has been identified. All finite automata are then reset to the start state and the outline is further traversed. In order not to depend on the selection of the first outline point, the outline is traversed once again or until all automata have been reset.

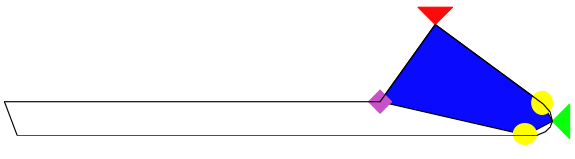

Figure 5: *Kanji serif defined by the sequence:* $\{HR\ point, CH\ point, T\ point, V\ point, T\ point\}$.

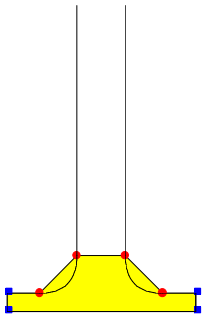

Figure 6: *Roman serif defined by the sequence:* $\{T\ point, T\ point, LC\ point, LC\ point, LC\ point, LC\ point, T\ point, T\ point\}$.

Whenever a new type of serif is encountered within a new typeface, only the sequence(s) of points defining

it should be inserted to the system. It is given a name (e.g. MinchoBarCap or RomanRoundHalfSerif) and a finite automaton recognizing the sequence is created and added to a pool of automates. Automates from this pool can then be turned on or off while converting a certain typeface, depending on the serif types expected to be encountered.

There are cases when one serif's defining sequence is a sub-sequence in some other serif defining sequence. Several methods could be used to overcome this situation. The simplest is turning off identification for the incorrect one. This might not always be possible, since one typeface could include both types of serifs. A better method would be to further classify the point types. For example, an L corner could be sub-classified to four different corner types according to orientation. This method can also be seen as adding geometric constraints to the transition rules in the finite automata.

When the methods above fail, we specify certain automata as being sub-automata of others. This means that when the sub-automaton has reached its final state, the including automaton is not reset. Both serifs are identified and the correct one is chosen. In our experience, the combination of these methods solves most ambiguities and clearly identifies all serifs.

5. Automatic Hinting

The process of hinting an outline font must recognize and use typographic features of the glyph and font in order to create the hint rules or constraints. In [Hersch91a] a topological model holding hinting information of each character (and variants of the same character) is built. A matching algorithm between real shapes of characters and the models is performed, allowing the transformation of hints from the model to the real shape. We have already mentioned the difficulty of creating such models for all huge variety of shapes representing Roman letters; moreover, the task seems close to impossible when looking at ideographic scripts.

Examining the different hint situations [Karow87] reveals that most of them are actually local in nature, meaning you must only examine a small portion of the outline of a glyph including several neighboring points (neighboring not only in the sense of consecutive points lying on the same contour, but also close in space) in order to decide whether a hint should be applied or not. This means there is no need for a topological model for the whole glyph, but rather a model for each hint situation. This is exactly the type of model a PTR is (see Figure 9). Once an outline font has been converted to a PTR, it already includes all information needed for creating grid-fitting hints to the outline of its glyphs, and so automatic hinting can easily be performed.

We have used our feature extraction technique on several different typefaces having widely different designs, converting them to a PTR. A large portion of the design difference between typefaces lies in the different types of serif design, or stroke endings in sans-serif typefaces. This difference can be handled by defining a number of typical finite automata to describe each typeface's serifs. Using this method we have managed to extract automatically all hinting situations from the outline of glyphs (see Figure 7) and have actually used this information to create constraints for Type 1 fonts and instructions for TrueType fonts.

Most typographic features are also cross-lingual in nature. Examining distinct types of scripts and writings, one can see that a major part of the differences between letter shapes lies in the different usage and likelihood of the same type of features. This of course is the outcome of all of them sharing the common source process of writing (in fact, a few major differences arise from differences in the writing techniques, for example using a pen in Roman scripts vs. a brush in Chinese and Japanese). Again, different serif designs are a major distinction between languages. Extracting typographic features should therefore be cross-lingual, which it is in our algorithm (Figure 10).

Hinting situations can therefore be defined almost exactly the same in all languages, and so we have also used a PTR for Japanese Kanji and PTR Hebrew typefaces in order to automatically hint and create digital fonts. Some difference in hinting different languages is of course inevitable, but can be overcome effectively using a PTR. As an example, Figure 11 shows the outline of the letter 'm'. This letter exemplifies one of the most difficult rasterizing problems of outline fonts, which even caused a special hint referred to as 'counter' to be defined. The typographic constraints for rasterizing this glyph are both 'black' and 'white' in nature. The three stems of the letter should retain their equal widths at all sizes (black) but also the two gaps between the stems should retain their equal widths at all sizes (white).

This task is not so simple to achieve when imposing a pixel grid of certain size on the glyph's outline, as can be seen in Figure 11. The question whether to make the leftmost and rightmost stems one or two pixels wide

could be answered only by looking at the middle stem. Furthermore, which pixel column should be chosen to represent the stems could be answered only by looking at the gaps between the stems.

In ideographic languages this problem is further enhanced because of the quantity and complexity of bars and stems in a glyph (Figure 10), and due to the nature of reading in these languages which uses mainly recognition of bars and stems structures in order to discriminate between symbols.

Our answer to this problem using a PTR is to define a higher level typographic element using lower level bars and stems. The special case where several stems are equal in width and length, and are placed in equal distances one after the other (all within the tolerance rule) is called a **stem-ladder**, or a **bar-ladder** in the horizontal case (see the X symbol in Figure 11). This typographic element is then used as a constraint while grid-fitting or deforming the letter, and can be further converted to rules or hints in a rasterizing language such as TrueType or Type 1 (Adobe Type 1 technology supports only triple ladders).

A PTR generated by our feature extraction algorithm is therefore a general and robust solution to the problem of automatic hinting of digital outline typefaces, which is one of the most important stages of digital font production today.

6. Conclusion

We have described a complete and efficient algorithm for extracting typographic elements from outline representations of typefaces. The algorithm is essential for converting existing typefaces into the modern family of representations which we call parametric typographic representations (PTR).

The algorithm was implemented as a part of a commercial system for conversion between different outline representations and for automatic hinting and regularization of typefaces. The system has been used on dozens of thousands of Roman, Kanji and Hebrew characters, and it is being successfully used on a daily basis. The fonts generated are consistently of higher hinting quality than most commercially available fonts.

A PTR is capable of providing answers to well known and difficult problems in font rasterization and digitization. such as hinting and regularization. In addition, it enables typefaces to take a more active part in electronic media using deformations, variations and animations while retaining their original typographic nature and design. Lack of space prevents us from describing these PTR applications in the present paper; they are more fully described in the technical report [Shamir95] and will be presented in future papers.

Additional future work should concentrate on the issues of parametrization of the extracted typographic elements and on the definition of typographic parts for extensively cursive and illustrated typefaces.

References

Adams89 Adams D., Abcdefg: a better constraint driven environment for font generation, in: André, Hersch (Eds.), Raster Imaging and Digital Typography, Cambridge University Press, 1989, pp. 54-70.

Adobe92 Adobe Developer Support, Adobe Type 1 Font Format: Multiple Master Extensions, 1992.

Adobe90 Adobe Systems Inc., The Type 1 Format Specification, Addison Wesley, 1990.

Andler90 Andler S., Automatic generation of gridfitting hints for rasterization of outline fonts or graphics, in R. Furuta (Ed.), EP90, Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography, Cambridge University Press, Sep. 1990, pp. 221-234.

Apple94 Apple Computer, QuickDraw-GX: Typography; Font File Formats; 1994.

Bauermeister88 Bauermeister B., A Manual of Comparative Typography: the PANOSE system, Van Nostrand Reinhold, New York, 1988.

Bétrisey89 Bétrisey C. and Hersch, Roger D., Flexible application of outline grid constraints, in: André, Hersch (Eds.), Raster Imaging and Digital Typography, Cambridge University Press, 1989, pp. 242-250.

Bringhurst92 Bringhurst, Robert, The Elements of Typographic Style, Hartley & Marks, 1992.

Changyuan93 Changyuan, Hu, and Fuyan, Zhang, Automatic hinting of Chinese outline font based on stroke separating method, Proceedings of the First Pacific Conference on Computer Graphics and Applications, Pacific Graphics '93, Vol.1, 1993, pp. 359-368.

Chialing89 Chialing Ou, and Yoshio Ohno, Font generation algorithms for Kanji characters, in: André, Hersch (Eds.), Raster Imaging and Digital Typography, Cambridge University Press, 1989, pp. 123-133.

Dürst93a Dürst, Martin J., Coordinate independent font description using Kanji as an example, Electronic Publishing: Origination, Dissemination, and Design 6(3):133-143, John Wiley & Sons, 1993.

Dürst93b Dürst, Martin J., Structured character description for font design: a preliminary approach based on Prolog, Proceedings of the First Pacific Conference on Computer Graphics and Applications, Pacific Graphics '93, Vol.1, 1993, pp. 369-380.

Feng75 Feng H.F., and Pavlidis T., Decomposition of polygons into simpler components: feature generation for syntactic pattern recognition, IEEE Transactions or Computers, vol. 24 June 1975, pp. 636-650.

Haralambous93 Haralambous, Yannis, Parametrization of PostScript fonts through METAFONT – an alternative to Adobe Multiple Master fonts, Electronic Publishing: Origination, Dissemination, and Design, 6(3):145-157, John Wiley & Sons 1993.

Hersch91a Hersch, Roger D. and Bétrisey C., Model-based matching and hinting of fonts, Proceedings SIGGRAPH '91, ACM Computer Graphics, Vol 25, July 1991, pp. 71-80.

Hersch91b Hersch, Roger D. and Bétrisey C., Advanced grid constraints: performances and limitations, in: Morris, André (Eds.), Raster Imaging and Digital Typography, Cambridge University Press, 1991, pp. 190-204.

Hersch87 Hersch, Roger D., Character generation under grid constraints, Proceedings SIGGRAPH '87, ACM Computer Graphics, Vol. 21, July 1987, pp. 243-252.

Karow94a Karow, Peter, Digital Typefaces: Description and Formats, Springer-Verlag, Berlin, 1994.

Karow94b Karow, Peter, Font Technology: Description and Tools, Springer-Verlag, Berlin, 1994.

Karow89 Karow, Peter, Automatic hinting for intelligent font scaling, in: André, Hersch (Eds.), Raster Imaging and Digital Typography, Cambridge University Press, 1989, 232-241.

Karow87 Karow, Peter, Intelligent FontScaling, Technical report, URW Unternehmensberstung, Hamburg, Germany, 1987.

Knuth86 Knuth, Donald E., The METAFONT Book, Addison-Wesley, 1986.

Lawson90 Lawson, Alexander, Anatomy of a Typeface, Hamish Hamilton, 1990.

Lewis81 Lewis, H.R., and Papadimitriou C.H., Elements of The Theory of Computation, Prentice-Hall, 1981.

McQueen93 McQueen, Clyde D., and Beausoleil, Raymond G., Infinifont: a parametric font generation system, Electronic Publishing: Origination, Dissemination, and Design 6(3):117-132, John Wiley & Sons 1993.

Microsoft90 Microsoft Corporation, The TrueType Font Format Specification, 1990.

Rubinstein88 Rubinstein, Richard, Digital Typography: An Introduction to Type and Composition for Computer System Design, Addison-Wesley, 1988.

Shamir95 Shamir, A. and Rappoport, A., Parametric typographic representation of typefaces: automatic conversion from outline fonts and applications, Technical Report TR-95-01, Institute of Computer Science, The Hebrew University, 1995.

Southall91 Southall Richard, Character Description Techniques in Type Manufacture, in: Morris, André (Eds.), Raster Imaging and Digital Typography, Cambridge University Press, 1991, pp. 16-27.

Stamm93a Stamm, Beat, Dynamic regularization of intelligent outline fonts, Electronic Publishing: Origination, Dissemination, and Design, 6(3), John Wiley & Sons, 1993.

Stamm93b Stamm, Beat, Object oriented and extensibility in a font scalar, Electronic Publishing: Origination, Dissemination, and Design 6(3):159-170, John Wiley & Sons, 1993.

Woodwark92 Woodwark, J.R., Some speculations on feature recognition, in: Geometric Reasoning, Kapur and Mundy (Eds), 1992.

Zhang92 Zhang, Fuyan and Ge Weimua, An automatic stroke separating method, Proceedings of the third international conference on Chinese information processing (ICCCIP '92), Beijing, 1992.

Figure 7: *The lower-case letters of the Bookman Light typeface, with the typographic features extracted marked in different colors. Vertical elements such as stems and bows are shown green, horizontal elements such as bars and arcs are shown blue, slants are shown yellow and serifs are shown red.*

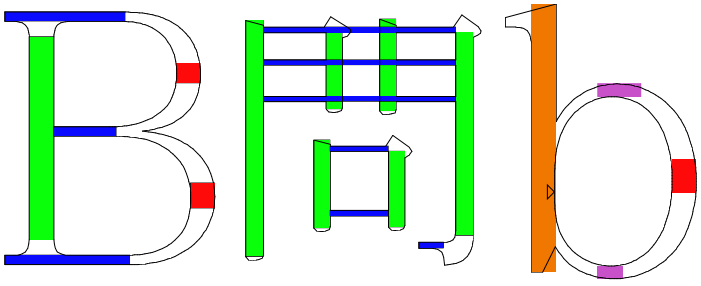

Figure 8: *Typographic elements: stems (green), bars (blue), bows (red), arcs (violet), curve stems (orange, notice the extremum triangle.)*

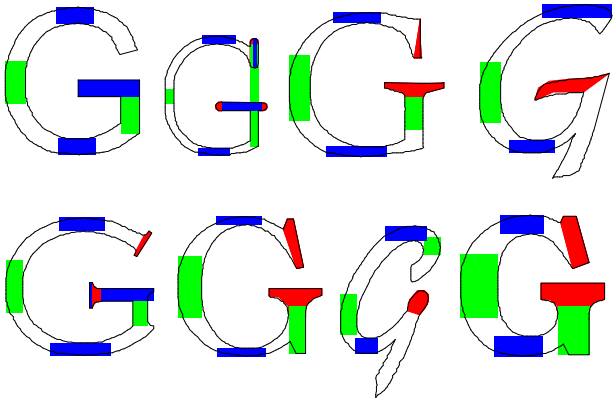

Figure 9: *Similar features extracted from different type of "G" glyphs.*

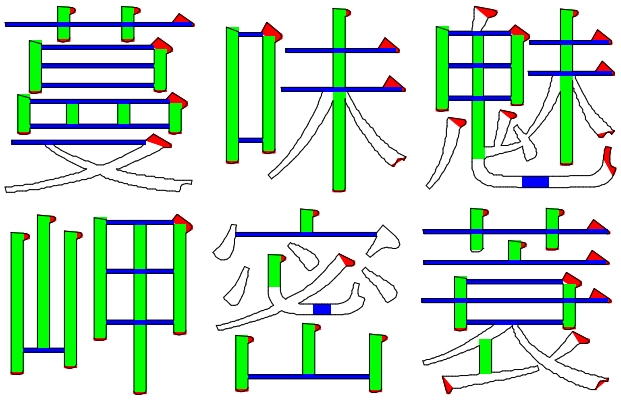

Figure 10: *Enhanced use of bars and stems in Japanese.*

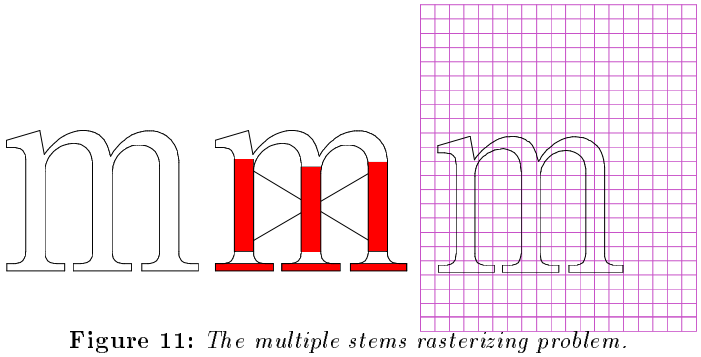

Figure 11: *The multiple stems rasterizing problem.*