ELECTRONIC PUBLISHING, VOL. 7(4), 251–260 (DECEMBER 1994)

Towards a universal auto-hinting system for typographic shapes

JACKY HERZ

Open University, Tel-Aviv
and Hebrew University, Jerusalem
Israel

ROGER D. HERSCH

Swiss Federal Institute of Technology
EPFL
CH-1015 Lausanne, Switzerland

SUMMARY

This contribution presents a simple method for the automatic recognition and hinting of character structure elements such as horizontal and vertical stems. Stem recognition is based on successive steps such as extraction of straight or nearly straight contour segments, detection of hidden segments, merging of original and hidden segments into larger segments, sorting of segments into classes according to their slopes and, finally, composition of black and white stems. Reference values required for character hinting purposes are obtained by evaluating the regularity of the font through statistical analysis of features such as stem widths and stem angles. Knowledge about the location of stems and analysis of outline parts between stems is used in order to produce automatically appropriate grid constraint rules (hints). The presented outline analysis and stem extraction techniques are very general and may be applied to non-Latin characters as well.

KEY WORDS Digital typography Shape analysis Stem recognition Automatic hinting

1 INTRODUCTION

Character outlines are not sufficient for generating high-quality characters on raster devices such as displays and printers. Additional information in the form of *hints* is generally added to the outline descriptions in order to simplify and improve the quality of the rasterization process. Generally, such hinting information comprises a description of the location of important character structure elements such as vertical and horizontal bars, curved stems, and half-serifs [10]. The rasterizer takes this information into account by controlling the outline phase in respect to the rasterization grid, and, possibly, applying appropriate outline modifications [5].

Existing hinting systems are often conceived to match particular families of fonts such as Latin fonts, Kanji, Arabic etc. While this approach usually leads to good results, it suffers from a lack of generality. This restriction emerges when encountering new font families or non-conforming font designs within the same font family. A classical example for Latin fonts is the famous Optima design.

We believe that the reason behind this phenomenon is that existing hinting systems rely heavily upon a pre-established model of a specific font family [4]. Whenever the typographic shape encountered fails to match this model, the quality of the result is condemned to deteriorate rapidly.

We would like to present in this paper an alternative and hopefully more general scheme for auto-hinting systems and a first implementation. This is not the first work in this direction; some previous examples may be found in [12], [3] and [8]. Most automatic

CCC 0894–3982/94/040251–10

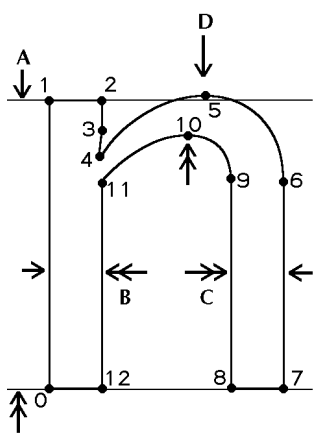
Received 15 August 1993
Revised 1 December 1993

hinting systems, however, are incorporated into proprietary software which is generally designed to deal with a myriad of particular cases. The merit of the presented approach consists in presenting a simple and general-purpose approach to the problem of automatic stem recognition and hinting. The outline deformation methods used for hint interpretation are relatively simple and are based on previous work [5]. However, the automatic stem recognition methods presented here can also be used by more elaborate, mathematically based outline deformation optimization methods [6]. Furthermore, they can be used for advanced tasks such as extraction of font features and decomposition of fonts into character structure elememts [11].

2 GRID-FITTING OF OUTLINE CHARACTERS

Hints are grid-fitting rules specifying which parts of outline characters should be adapted to the grid in order to obtain nice regular raster characters [5]. These grid-fitting rules mainly apply to character reference lines, stems, bowls and serifs. Grid-fitting rules for fitting stems and bowls require two outline support points specifying the stem or bowl width.

The evaluation of a given grid-fitting rule or hint produces a subpixel displacement which must be applied to certain parts of the character outline. As shown in Figure 1, a grid-fitting rule comprises a specification part describing the rule's type and parameters. Based on the rule specification, the outline character rasterizer computes at character rendering time one or two displacement values used for grid-fitting specific outline parts. The grid-fitting rule's application part specifies the character parts on which the computed displacements are applied by giving their starting and ending outline support points (Figure 1).



A: **hint specification:** vertical phase control of reference lines
hint application: complete character

B: **hint specification:** horizontal phase control of vertical stem
stem width given by Pt 0, Pt 12
hint application: complete character

C: **hint specification:** horizontal phase control of vertical stem
stem width given by Pt 8, Pt 7
hint application: fixed displacement: Pt 6 to Pt 9
proportional displacement: Pt 4 to Pt 6
fixpoint: Pt 4
max. displacement point: Pt 6
proportional displacement: Pt 9 to Pt 11
fixpoint: Pt 11
max. displacement point: Pt 9

D: **hint specification:** vertical phase control of shoulder with respect to reference lines
shoulder thickness given by Pt 10, Pt 5
hint application: proportional displacement: Pt 4 to Pt 6
fixpoint: Pt 4, Pt 6
max. displacement point:Pt 5
proportional displacement: Pt 9 to Pt 11
fixpoint: Pt 9, Pt 11
max. displacement point: Pt 10

Figure 1. Support points for the specification of grid constraints

3 SOME WORDS ABOUT THE TITLE

The use of the word *universal* means that our proposed hinting system should not be dependent on any particular underlying model of the font that the system is going to render. The methods employed should adapt automatically to any group of typographic shapes.

The expression *typographic shapes* refers to a group of shapes that have a coherent design, e.g. repetitive elements and metrics. When talking about characters, such a group of typographic shapes is usually called a *font*. However, here, the word *font* encompasses a broader scope: road-signs, musical notes or icons of a windowing system, all of them are distinct groups of typographic shapes and members of font families.

When talking about *auto-hinting* we refer to the information supplied to the system. Our system should rely exclusively upon the outlines of the shapes without any additional information, be it declarative (as in Type1 [1]) or procedural (as with TrueType technology [2]).

The first word of the title — *Towards* is intended to somewhat lower expectations. For the moment, the presented auto-hinting method has only been applied to vertical, horizontal and diagonal bars. Further work is required to extend the method to curved stems and to terminal elements.

4 FUNDAMENTAL ELEMENTS AND FONT METRICS

The basic assumption when trying to build a universal auto-hinting system is that human observation is very sensitive to particular visible structures, especially when these structures are repeated all over a group of typographic shapes. Furthermore, typographic shapes are basically defined by the existence of regular structures. From this point of view they differ from any other arbitrary group of graphic objects.

Stems (which in this paper refer to long straight strokes that may be vertical, horizontal or diagonal) are one of the most important and most common structural elements; they occur in almost any non-cursive font. Furthermore, hinting of stems is relatively easy, especially when dealing with horizontal and vertical ones.

Even in cursive (italic) fonts, stems having an oblique orientation are the dominant elements. After back-slanting the font, such stems can be regarded as belonging to the class of vertical stems.

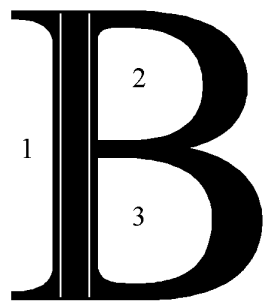


Figure 2. The vertical stem of 'B' is composed of three contours

In the following description, we assume that the shapes' outlines are expressed in a canonical form such as is described in [9].

Consequently we assume that the following conditions hold for the outline:

1. The outline is composed of curves and/or straight lines.
2. There is a point at every local extremum in X or Y.
3. There are as few points as possible.
4. There is an easy way to determine whether an arbitrary point is inside or outside the painted (black) area of the shape's outline.
5. As in most commercial font systems, character outlines don't cross each other. This is the main difference with character descriptions such as Metafont, where an "x" character can be made of two intersecting bars. In non-intersecting outline descriptions, the information about two bars crossing each other is given only implicitly.

An additional requirement is the existence of a filter that guarantees a minimal distance between consecutive points of a contour, thus hiding local details while emphasizing global structures.

5 STEM RECOGNITION

When looking at typographic shapes, the intuitive definition of a stem as the area limited between two parallel straight lines is both too broad and too narrow. A strict interpretation is clearly too limited because the geometric definitions of *parallel* and *straight lines* are too restrictive. Furthermore, the straight lines that are visible to the human eye often appear only implicitly in the outline. The vertical stem of a Latin 'B' character is a good example: its vertical stem is composed of three different contours (Figure 2), and it is only the human brain that traces an imaginary line through the painted area of the character, thus adding to the visible contour segments that are not explicitly included in the input data. We will call such segments *hidden segments* (Figure 3).

Figure 3. Automatically recognized hidden segments in a Kanji character

The first step towards stem recognition is the identification of hidden segments using the following stages:

- **A:** Collect all the segments that are straight or have a flat curvature into a group S. In the case of segments that are curved, the *flatness* is defined using a threshold that takes into account the maximum distance between a point on the curve and the straight line connecting its endpoints (Figure 4).

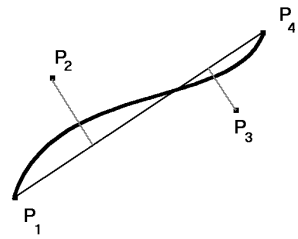


Figure 4. Flat curves. When the distance between the control points P2 and P3 and the segment [P1,P4] is smaller then a given threshold, the curve is considered to be flat

- **B:** At this stage we add hidden segments to S using the following procedure:
  FOR all flat segments [P1,P2] in S DO
  FOR all point P3 on the outline DO
  IF P1,P2,P3 are colinear (here again, using a threshold)
  AND
  IF the segment [P2,P3] can be traced entirely in the painted area of the shape
  THEN [P2,P3] is a hidden segment.
  END FOR
  END FOR
  At the end of this procedure we add to S all the hidden segments just found.

- **C:** When hidden segments are available as explicit data, it becomes possible to merge segments within S (original segments and hidden segments) in order to create continuous straight lines. Here again, the term *straight* is too restrictive; in reality, we are thinking about lines that appear to be more or less straight. In practice this means that the slope variations along the line do not exceed a predefined threshold value. Hence we will prefer the expression *stem-edge* to *straight-line*. The potential stem-edges are reconstructed iteratively as follows:
  WHILE S includes two segments [P1,...,P2] and [P2,...,P3]
  such that: The slope of [P1,...,P2] = The slope of [P1,...,P3]
  BEGIN
  remove [P1,...,P2] from S,
  remove [P2,...,P3] from S,
  add the segment [P1,...,P2,...,P3] to S.
  END
  END WHILE
  At the end of this procedure the group S includes the potential stem-edges.

- **D:** Now we sort all potential stem-edges in S according to their slope into several groups:
  SV — includes only vertical segments.
  SH — includes only horizontal segments.
  SD1...SDn — Each group includes diagonal segments of similar slopes.

- **E:** We are now in a position which makes stem recognition possible. To achieve this goal we proceed as follows. For each group defined above, try to compose couples

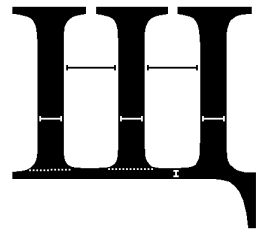


Figure 5. Black and white stems in a Russian character

of stem-edges into a single stem. Here we demand that the following conditions hold for each couple of segments:

1. The segments, should appear parallel (that is, within a predefined slope difference threshold). This condition is normally fulfilled by means of the group partition in the previous stage.
2. A stem is defined as a *black* stem if most of the trapezoid enclosed between both stem-edges is included inside the painted area of the shape.
3. A stem is defined as a *white* stem (commonly called “counter") if most of the trapezoid enclosed between both stem-edges is included outside the painted area of the shape.
4. All stems are either *black* or *white* (Figure 5).

6 COMPUTING THE INFORMATION REQUIRED FOR STEM HINTING

Recall the information we now have for each recognized stem:

1. Each stem is defined by its two edges.
2. Each edge is composed of an ordered list of points.

It is now easy to compute some other useful pieces of information:

1. The stem width is defined as the average distance between its edges.
2. The medial axis of a stem is a line traced midway between both edges.

This information is local, that is, it has been computed for each stem independently from the others. After discovering the stems throughout a given font, we can use histograms to find global values that reflect the regularity of the design. Some examples are listed below:

1. Stems with close width values (relative to a given pixel size or relative to the shape’s height) may be considered as having exactly the same width. These repetitive width values are inserted into a special table of reference width values.
2. Horizontal stems are often placed on *reference lines*. Reference lines are horizontal lines delimiting the extension of characters of a given font (e.g. baseline, x-height line, capsline).
3. Another useful global value is the slope of slanted stems. In italic fonts, slanted stems generally have an identical slant angle. After applying a back-slanting transformation, they can be handled, up to a certain extent, as vertical stems. Slant angles can generally be found in the font tile header. In certain cases, this information may have to be retrieved automatically.

The hinting process may be viewed as an attempt to emphasize the appearance of selected elements and their properties within the font. As resources are usually limited (e.g. output resolution), this emphasis has its price: other elements of the shape are considered as *flexible* and as a consequence their distortion (e.g. stressed or narrowed) enables a faithful rendering of specific parts within a shape. At rendering time, a vertical or horizontal stem is generally rendered by placing its medial axis at specific grid locations. The medial axis of a vertical (or horizontal) bar is placed exactly on pixel centres or at the frontier between two pixel columns.

Hinted character outline description languages and appropriate interpreters and rasterizers exist. By using the methods described above we are able to automatically supply the information needed for declarative hinting systems such as *Type1* [1]. For semi-declarative hinting systems, such as *RastWare* [5] or fully interpretative hinting systems such as TrueType [2], one has to add information about the application domain of the computed displacement, for example, about which outline segments have to be stretched or displaced. This additional information is easy to compute since the stretched segments are exactly those which connect the currently processed stem with the last stem that has already been placed correctly on the grid. If no stem was previously placed, the entire shape is translated onto the grid according to the computed stem displacement.

7 DETERMINATION OF OUTLINE PART DISPLACEMENTS ASSOCIATED TO HINTS

We will use our stem detection technique in order to automatically generate the hint description required by the *RastWare* hint interpreter and rasterizer [5]. In that system, the hinting task consists of associating predefined types of grid constraints (hints) to stems and, for each grid constraint, specifying which part of the outline of the character needs to be displaced at hint interpretation time.

The same techniques can be applied to automatically generate the hints in the TrueType format [2]. We will rely on the *RastWare* hinting principles and show that knowledge about the location of stems within a shape leads to efficient and automatic hinting. For the sake of simplicity, we intend to present only the automatic generation of vertical stem hints.

The general idea is very simple. A stem is a symmetrical structure, and its medial axis (central line) is its axis of symmetry. To maintain this very significant property, the medial axis of stems should be placed on the grid in a position which ensures that symmetry will be kept throughout the rasterization process.

There are exactly two possibilities for placing a vertical (or horizontal) medial axis of a stem symmetrically onto the grid. It should be placed exactly on the border of a column (row) of pixels when the width of the stem (in pixel units) is rounded to an even number, and exactly on the centre of a column (row) of pixels when the width of the stem is rounded to an odd number of pixels. We shall call these two possible positions *symmetrical positions*.

Following this general principle we may now describe a general procedure for vertical stem hinting:

Initialization:

1. Mark all the points on the outline as h-free (horizontally free). A point may only have one of two values: h-free or h-fixed.
2. Create a list L where all vertical stems in the outline are sorted from left to right according to their medial axis.

3. For each stem compute its width in pixel units.
4. Find all the outline parts (segments) that connect two consecutive stems in L. These segments are considered as *flexible segments*.

```
BEGIN
  IF L is empty THEN exit; /* No stem to process */
  /* Fixing the leftmost stem */
  1. Remove the first stem S out of L;
  2. Move the outline on the grid, in a way that will put the medial axis of S on its
  nearest symmetrical position;
  3. Add S to a new stem list FSL (fixed stems list).
  4. Mark as h-fixed all the points that are not located horizontally on the right side
  of S.
    /*
      These points are now horizontally fixed and will not move any more during
      this procedure
    */
    WHILE L is not empty DO
    BEGIN
    1. Remove the first stem S out of L;
    2. Move the outline on the grid, in a way that will put the medial axis of S
    on its nearest symmetrical position but this time the points of the outline are
    treated differently according to their location relative to the medial axis:
      A. All the points that are not located horizontally on the left side of S
      move horizontally according to the medial axis of S.
      B. All the h-free points on flexible segments that connect S to its preceding
      stem in FSL, are stretched proportionally in the horizontal direction, to
      compensate for the movement of S.
    3. Mark as h-fixed all the points that are not located horizontally on the right
    side of S.
    END /* of WHILE */
END /* of the stem hinting procedure */
```

In Figure 6 we see the shape of the character ‘m’, to which we have added all the information needed for vertical stem hinting. Note that all of this information has been collected automatically: the position of its three vertical stems, the position of their medial axis and the location of the flexible segments.

8 RESULTS

An example of the results is shown in Figures 7 (Kanji), 8 (Devanagari) and 9 (Latin). The figures labelled with (a) show the results of a naive rasterization without any hints. The figures labelled with (b) show a similar rasterization but after automatic stem hinting as described above. In the figures labelled with (c) an additional well-known mechanism — the *dropout control* [5] — is applied at character rasterization time. This mechanism enables *holes* to be avoided in thin character parts. In the present examples, this mechanism was applied to generate dropout-free curved stems and arches.

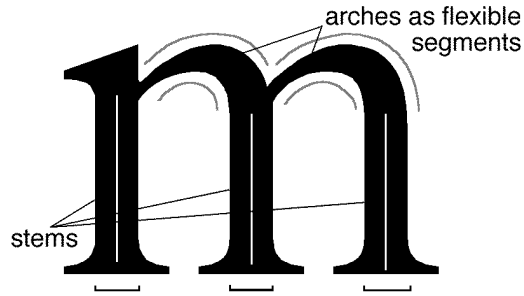
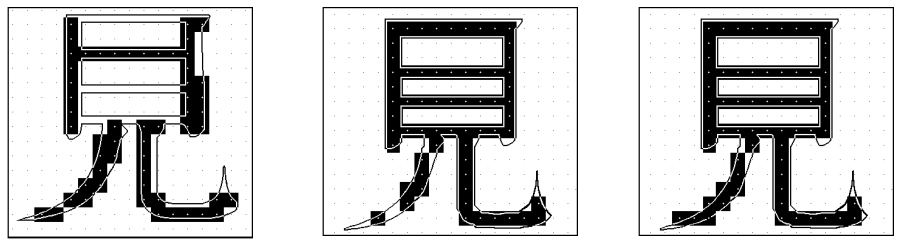
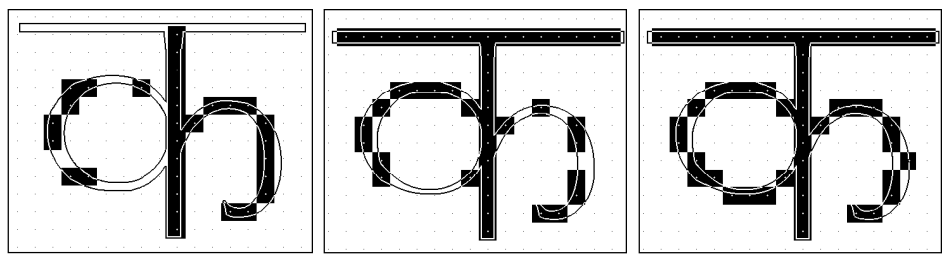


Figure 6. Information collected automatically about vertical stems in the character 'm': stem width, medial axis and flexible segments



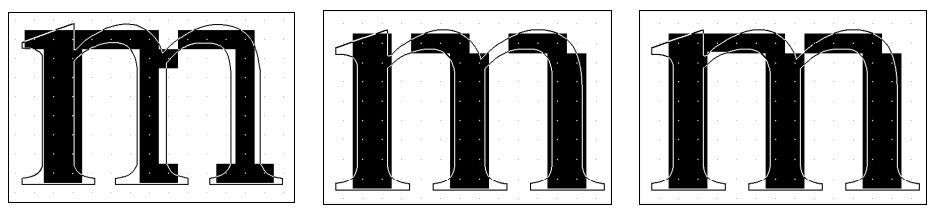
(a) *(b)* *(c)*

Figure 7.



(a) *(b)* *(c)*

Figure 8.



(a) *(b)* *(c)*

Figure 9.

9 COMPLEXITY

Assuming that a shape includes at most N segments, the price of examining all couples of segments for the existence of hidden segments is bounded by N^2 operations. This complexity may be reduced by sorting the segments according to their slopes and checking the existence of hidden segments only between couples of segments with similar slopes. The same time bound also holds for testing couples of stem-edges because their number cannot exceed N. Hence, if the number of shapes in a font is F then the complexity of extracting all stems out of the raw outlines is $O(F \times N^2)$.

In any case, the complexity of stem extraction is not very significant compared to the complete character preparation process. The information about stems has to be computed only once during the auto-hinting process. This information is used for hint interpretation each time typographic shapes are scaled and rasterized.

10 CONCLUSION

The automatic stem recognition method described above is very simple and follows an intuitive logic by trying to mimic human perception. Methods like the one described in this paper may simplify existing sophisticated and specialized hinting systems, enabling them to work correctly with any group of typographic shapes.

In spite of its simplicity the method is very general and can be used not only for automatic hinting, but also for solving a wide range of problems, such as rasterization of Kanji fonts on limited resolution displays, automatic regularization of typefaces, extraction of font features and font decomposition into character structure elements.

REFERENCES

1. Adobe Systems Inc., *Adobe Type 1 Font Format,* Addison-Wesley, Reading, MA, (1990).
2. Apple Computer, *TrueType Spec — The TrueType Font Format Specification*, July (1990).
3. S. Andler, "Automatic generation of grid fitting hints for rasterization of outline fonts or graphics", *EP90 — Proceedings of the International conference on Electronic Publishing, Document Manipulation and Typography*, September 90, ed. R. Furuta, Cambridge University Press, Cambridge, 1990 pp. 242–250.
4. R.D. Hersch and C. Bétrisey, "Model-based matching and hinting of fonts", Proceedings SIGGRAPH'91, *ACM Computer Graphics*, **25**, 71–80, (1991).
5. R.D. Hersch, "Font rasterization, the state of the art", *Visual and Technical Aspects of Type*, ed. R.D. Hersch, Cambridge University Press, Cambridge (1993), pp. 78–109.
6. John S. Hobby, "Generating automatically tuned bitmaps from outlines", *Journal of the ACM*, **40**(1), 48–94 (1993).
7. Louisa Lam, Seong-Whan Lee and Ching Y. Suen, 'Thinning methodologies — a comprehensive survey', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**, 869–885, (1992).
8. P. Karow, "Automatic hinting for intelligent font scaling", *Raster Imaging and Digital Typography*, Eds. J. André and R.D. Hersch), Cambridge University Press, Cambridge (1989), pp. 232–241.
9. P. Karow, *Digitale Schriften, Darstellung und Formate*, Springer Verlag, Berlin, (1992).
10. P. Karow, *Schrifttechnologie, Methoden und Werkzeuge*, Springer Verlag, Berlin, (1992).
11. C. D. McQueen and R. G. Beausoleil, "Infinifont, a parametric font generation system", *Electronic Publishing — Origination, Dissemination and Design* **6**(3), 117–132, (1993).
12. Chialing Ou and Yoshio Ohno, "Font generation algorithms for Kanji characters", in J. André and R. D. Hersch (eds.), *Raster Imaging and Digital Typography*, Cambridge University Press, Cambridge (1989), pp. 123–133.