A Window-based Method For Automatic Typographic Parameter Extraction

Jacky Hertz[1], Changyuan Hu[2], Jakob Gonczarowski[3], Roger D. Hersch[4]

[1]Open University, Jerusalem, Israel
[2,4]Ecole Polytechnique Federale de Lausanne, Switzerland
[3]Typographics Ltd. Jerusalem, Israel

Abstract. The synthesis of existing fonts with characters represented by parametrized structure elements requires determining a set of font-specific global and local parameters. Parameters comprise, for example, the widths of vertical stems, horizontal bars and curved elements, the spacing between vertical stems, the relative position of the junction between arches and vertical bars and serif measures. These parameters need to be extracted from existing outline fonts. This paper presents a window-based method for locating within existing outline characters the position of character features from which parameters can be measured. The method is based on the match between outline characters and their corresponding virtual skeletons.

1 Introduction

Most commercial font systems use outline fonts for the digital presentation of typefaces. Outline fonts are well suited for character rasterization and printing. They however only incorporate implicit information about important font features such as the width of stems, bars and bowls, the distance between vertical stems and bowls and the parameters determining junctions between character elements and determining the shape of terminal elements (serifs). Efforts are made to create a more flexible character representation incorporating explicit information about features of the character, thus enabling the synthesis of coherent font variations, for example by varying weight and contrast. The main idea is to describe characters by an assembly of appropriate structure elements. Several attempts have been made to describe characters by structure elements. D. Knuth described his Computer Modern fonts by horizontal, vertical, diagonal strokes and by round parts [9]. These strokes and round parts are given by sequences of pen positions and directions. More recently, Bauermeister et al. created the Infinifont system enabling resynthesizing an existing font from universal font generation rules and from parametric data specific to that font [2][10]. Our own efforts aim at defining character structure elements suitable for the quick generation of existing and new typographic fonts. Figure 1 shows a character assembled from structure elements, i.e. vertical stems, a round arch and serif components.

Components are predefined parametrisable shapes. A font engine reads the parameters, generates the components of a character and assembles them into the final printable character. To resynthesize an existing font by parametrisable structure elements, character parameters need to be extracted from the original font, either manually or algorith-

Electronic Publishing, Artistic Imaging and Digital Typography, (Eds. R.D. Hersch, J. André, H. Brown), 1998, LNCS 1375, Springer-Verlag, 44-54

mically. In this paper, we present a window-based method for the computerized extraction of parameters from an existing font. First, we describe the geometric meanings of font parameters by giving a detailed example. Then we introduce a window-based feature tagging algorithm and explain how it can be used to extract parameters from existing character shapes.

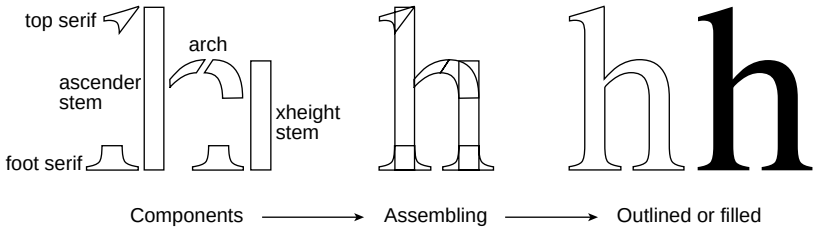

Fig. 1. A character is assembled by its components

Previous approaches to locate typographic elements within characters required either matching characters with a font-independent character outline model [12] or were based on contour tracking, analysis of subsequent contour elements and extraction of typographic elements such as vertical stems, horizontal bars, diagonal bars, curved character parts and serifs [8][5][13].

2 Parameters for font parametrization

With our font parametrization method, parameters have clearly defined typographical meanings. They are used to describe the characters' geometric features, such as width of stems and arches, serif metrics, angles of junctions, orientation of tails and peaks, etc. Some parameters are globally available and are applied to all or to a group of several characters to ensure the coherence of a font, respectively the coherence of a group of parameters [1]. Some parameters have only a local meaning for one specific character. Values of parameters may be the distance between two points, the intermediate position (proportion) of one point between two other points, and the angle of a junction or of a top serif. In Figure 2, we give the typographical meaning of the considered parameters for Times Roman character "h":

- p_1 and p_2 are proportion values which control the horizontal position of the external and internal vertical extreme points of the arch
- p_3 and p_4 control the departure position of the arch, as a proportion of the x-height
- θ_1 is the angle controlling the direction in which the arch joins the vertical stem
- θ_2 is the angle controlling the orientation of the top serif
- p_5 or *ArmPos* is a proportional value used by a group of characters controlling the position of the junction between the arch (arm) and the vertical stem
- *dx* is the distance between the two main vertical stems; this parameter is used for a group of similar characters, such as h, m, n, u etc.

- w_1 is the standard vertical stem width (global parameter)
- w_2 is the standard mathematical horizontal curve width (not a true visual stroke width, also a global parameter)
- w_3 is the width of the arch at the junction
- sw_1, sh_1 and sd_1 are respectively the foot serif width, height and depth (global parameters)
- sw_2, sh_2 and sd_2 are respectively the top serif width, height and depth (global parameters)
- *ovst* is the overshoot at the tip of the top serif, which aims at making the face of the serif slightly concave (global parameter)

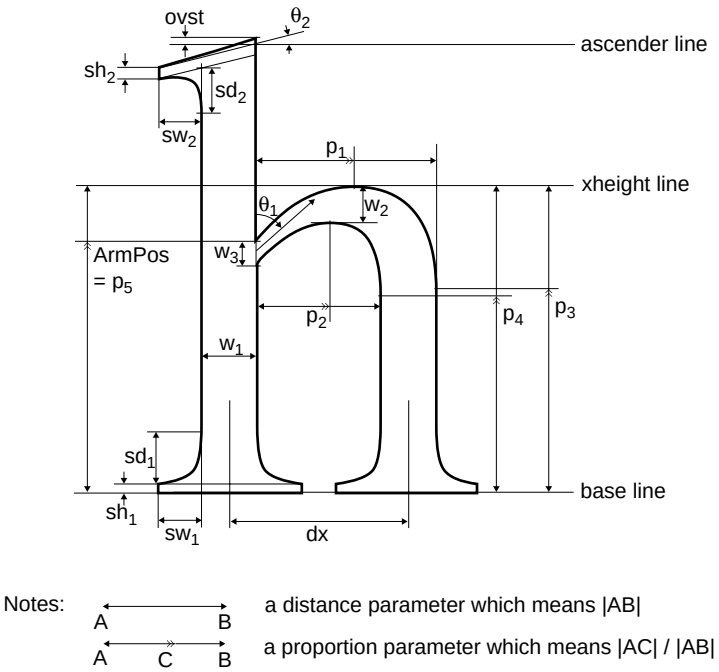

Fig. 2. An example of parameters for font parametrization.

The value of a distance parameter is not an absolute value. It depends directly on the height of the character. For lower-case characters, the height of a character is represented by the x-height, which is a global font parameter. As an example, parameter values of character Times Roman “h” (Fig. 2) are listed in Table 1. Here, the value of *x-height* is taken from the outline font header description. It can also be extracted from character “z”.

Table 1. Parameters of Times Roman “h”.

xheight	dx	w_1	w_2	w_3	p_1	p_2	p_3	p_4	p_5	θ_1	θ_2	sw_1	sh_1	sd_1	sw_2	sh_2	sd_2
445	266	86	56	34	.60	.61	.76	.72	.84	50^o	15^o	64	15	64	64	15	64

3 Parameter extraction using an automatic tagging system

In the previous sections we presented font parameters which may be used to create the components of a coherent set of characters. These parameters may then be varied to create coherent derived fonts sharing a similar basic structure. In this section, we describe a method for extracting these parameters by computer programs.

For this purpose, we use a system for tagging character parts. This system enables the decomposition of letter forms into their primary elements. The reader is referred to [6] for a detailed description of the system. Here we will review the main elements of the method that enables the semi-automatic extraction parameters.

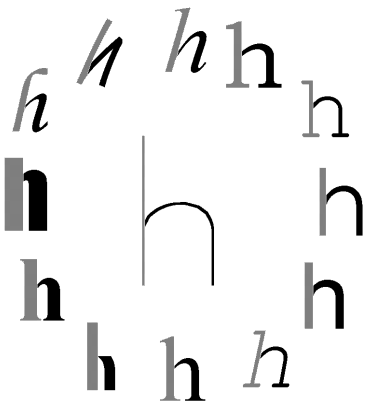

Fig. 3. A single virtual skeleton induces a similar partition in various fonts.

This method uses a universal model for letter shapes - the *virtual skeleton*. This model is simple and natural: Everyone can draw it, at least in one's native alphabet. Despite its simplicity this model is powerful. Its power is due to its double role: The virtual skeleton is at the same time a skeletal representation of the character [4] and a planar graph [11]. Like every skeleton, it has geometric and topological properties. On the other hand, the graph representation enables a natural and straightforward use of many computer algorithms. For instance, it is easy to decompose a virtual skeleton into its elementary elements by considering subgraphs of the virtual skeleton as the graph's elementary elements. Then, the mapping between a virtual skeleton and a particular letter shape provides a means of decomposing the letter shape into its structure elements. This

concept is demonstrated in Figure 3 where the partition of the virtual skeleton into two distinct subgraphs induces a similar partition in various instances of the same character.

The mapping between a virtual skeleton and a letter shape is achieved by placing appropriate windows on the character's outlines (Figure 4b). These windows match the vertices of the virtual skeleton (Figure 4a).

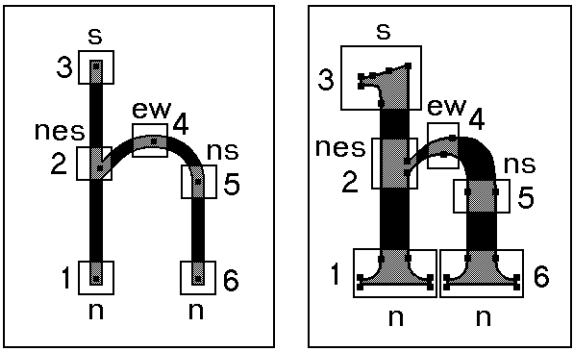

Fig. 4. Each vertex of the virtual skeleton (a) corresponds to a window placed on the outlines (b). Note the window numbers and the corresponding vertex numbers. We will refer to these numbers when mentioning a particular window or vertex.

The window placement procedure is detailed in the section 4. The criteria for a valuable match between a vertex on the virtual skeleton and a window placed on a letter shape are the following:

- A window must be well placed on the outlines: Its four corners must reside on the white part of the character. A window that is not well placed can't match any vertex.
- The vertex of the virtual skeleton and the corresponding window share a similar relative geometric position (i.e. top-left, or bottom-right region of the character).
- Every vertex or window incorporates one or more entering segments. These skeleton segments (or character parts) enter a vertex (or a window) from different directions. We classify these directions using four categories *north*, *east*, *south* and *west*. This enables us to mark each vertex or window with a direction string composed of the letters n, e, s, w. This direction string represents the entering directions. In order to match a vertex with a window, the vertex and the window must share the same direction string.
- Considering the windows on the character outlines as vertices, and their interconnecting outlines (black) parts as arcs, an isomorphism between the created graph and the virtual skeleton graph must obviously exist. This graph isomorphism (see Figure 4) ensures that the match between vertices and windows is coherent.

4 Looking for matching windows

In this section,we supply an intuitive description of the window placing procedure.There are four parameters that define the size and the location of a rectangular window:

- h - the rectangle height. We assume that this size may vary between H_0 and H_n.
- w - the rectangle width. We assume that this size may vary between W_0 and W_n.
- (x,y) - the 2 coordinates of the rectangle center. We assume that the coordinates values vary between X_0 and X_n, as well as between Y_0 and Y_n.

Given these parameters, a straight forward algorithm for finding a matching window would perform an exhaustive search but the complexity of an exhaustive search is high. Such a search must be composed of 4 nested loops, each varying a distinct parameter. Inside these loops there will be a test to decide whether an appropriate window has been reached. Unfortunately, most of the trials will probably not lead to matching windows. This probable failure is due to the resolution and the phase problems (see Figure 5). Furthermore, in an exhaustive search, any wrong value for a parameter in the most external loop leads to a long sequence of failures to find a matching window.

In order to avoid the complexity of an exhaustive search, we suggested in [6] (and demonstrated the feasibility) of a probabilistic algorithm to find a matching window. In this algorithm the exhaustive search is replaced by a random search. Note that this approach is significantly simpler:

A probabilistic algorithm to find a matching window

DO K times:
{
  Randomly create a value h, $H_0 < h < H_n$)
  Randomly create a value w, $W_0 < w < W_n$)
  Randomly create a value x, $X_0 < x < X_n$)
  Randomly create a value y, $Y_0 < y < Y_n$)
  {
    Create a window WM using the parameters h, w, and l(x,y).
    IF the window WM matches the vertex V,
    THEN EXIT with a positive answer:
      A matching window (WM) has been found.
  }
}
EXIT with a failure message: No matching window could be found.

In this Algorithm, the number of trials is limited by K. As shown in [6], a value of K=1000 largely suffices in most cases in order to obtain an answer which has a very high probability of correctness (at least in Latin fonts). Furthermore, the error probability of this algorithm may be reduced ad infinitum, simply by increasing the value of K.

This algorithm elegantly avoids the phase and the resolution problem encountered by any exhaustive search (Fig. 5).

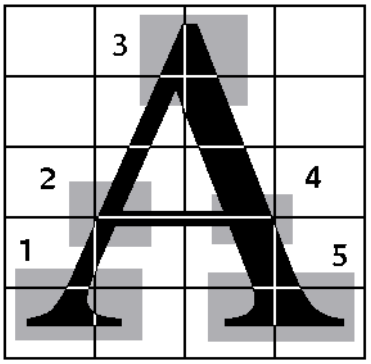

Fig. 5. The phase and the resolution problems. Assume that windows of the size and position of cells in the above grid are used to perform an exhaustive search for matching windows. This search will fail. No cell in this grid can ever represent any matching window because of the cells' positions (this is the phase problem). Note also that the matching windows numbers 1, 3 and 5 are too big to fit in any cell - this is the resolution problem.

While each parameter in the algorithm above varies randomly within its range: $[H_0..H_n]$, $[W_0..W_n]$, $[X_0..X_n]$ or $[Y_0..Y_n]$, the parameter range itself is (approximately) determined by the location of the window we are looking for. For instance, when looking for a matching window for vertex number 3 in Figure 4a (the upper end of the vertical stroke of the letter h), we can limit our search to the upper left side of the bounding rectangle of the character outlines. This information is supplied by the virtual skeleton that suggests a zone where we will be looking for a matching window. The searching zone boundaries limit the range of each window parameter thus reducing the search space and enhancing the efficiency of the search algorithm.

5 Parameter extraction - a simple example

In order to extract parameters, we take now a closer look at the windows of Figure 6b. For the sake of simplicity, we describe the extraction of a few parameters only. We list these parameters and explain briefly how they are extracted from the information provided by the matching windows. Again, the parameters are extracted from the outline representation of Times-Roman character "h". The parameter we are looking for belong to the following three classes:

A. Junction orientation
B. Stem, bar and curve width
C. Stem position
D. Serif metrics

A. Junction orientation

The slope we are looking for (θ_1 in Figure 6) is indicated by a small arrow emerging from the arm junction inside Window 2. This slope is the average slope of the slopes of the two contour tangents at the junction between stem and arch. Computing the precise value of this angle is particularly simple when the contour is given as a sequence of Bézier curves (e.g. [3]).

B. Stem, bar and curve width

1) Vertical stem width (w_1 in Figure 6).

The stem width is the horizontal distance between the pair of (vertical) contour segments captured inside Window 2 and Window 5.

2) Horizontal arch width (w_2 in Figure 6).

This is the distance between the two vertical extremes of the pair of contour segments captured within Window 4.

3) Horizontal arch width at the junction (w_3 in Figure 6).

This distance is measured within Window 2 between the two points that mark the transition between the vertical segments (which are part of the vertical stem) and the curves that start the round part of character “h”.

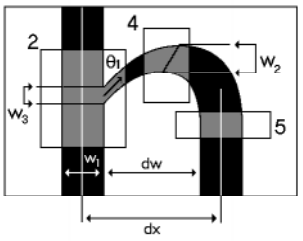

Fig. 6. Parameters that depend on windows number 2, 4 and 5.

C. Stem position

1) The distance between the center lines of the vertical stems (dx in Figure 6).

The center lines of the stems (marked by vertical white lines inside the stems) are obtained from Windows 2 and 5.

2) The white distance between vertical stems (dw in Figure 6).

In order to obtain this parameter, we measure the distance between the right side of the left stem (included in Window 2) and the left side of the right stem (included in Window 5).

D. Serif metrics

1) Serif width (sw in Figure 7).

This is the horizontal distance measured between the vertical stem's right edge and the rightmost shape primitive vertex in Window 1.

2) Serif height (sh in Figure 7).

This is the vertical distance between the vertices of the vertically oriented shape primitive (straight line or Bézier spline segment) defining the serif extremity (two rightmost vertices in Window 1)

3) Serif depth (sd in Figure 7).

This is the vertical distance between the starting point and the ending point of the curved part of the lower serifs.

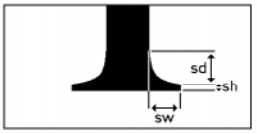

Fig. 7. Parameters that depend on window 1.

4) Angle of top serif (θ_2 in Figure 8).

This is the departure tangent of the segment that connects the top serif with the left (vertical) side of the serif. The top serif is included within Window 3.

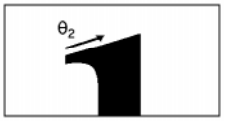

Fig. 8. The departure tangent of the top serif is measured in Window 3.

6 Conclusions and further research

The idea of using parameters as the basic tool for automatic font generation has emerged at the very beginning of digital typography. For instance, when describing the digital design of the Computer Modern typeface using the METAFONT system, D. E. Knuth states that this typeface is completely controlled by 62 parameters (listed in [9]). Knuth implicitly claims that for every alphabet there exists a final set of controlling parameters. Varying these parameters would create all possible fonts of this alphabet. This implicit claim has raised many objections. One of the most powerful arguments against it appeared in [7]. In this article, D. R. Hofstadter agreed that parameters are crucial for creating new typefaces, but he argued that there can never exist such a final set of parameters, (or knobs in his words). The reason, according to Hofstadter, is that the art of font design consists mainly of inventing new parameters. In other words, creativity is often expressed by the invention of new parameters - ones that were never known before.

Once we agree upon the importance of font parameters, the question to be asked is which are the relevant parameters and how can they be found. In this paper we tried to answer the second question. We first presented a list of parameters and showed where they appear on a letter shape. Then we suggested a simple method to extract these parameters.

Fortunately, parameters are often related to special areas of a letter form - like junctions, serifs etc. These are also the places where the vertices of the correspondent virtual skeleton are usually located. For instance, a junction of a virtual skeleton is marked by a vertex connecting three or more arcs (like Vertex 2 in Figure 4a). Therefore character and font parameters are likely to be recognized by looking into the windows which match the vertices of the corresponding virtual skeleton.

The presented window-based method distinguishes itself from automatic character analysis and feature extraction methods [8][5][13] by its ability to extract a feature at a precise location within a character. In character "g" for example, the parameters associated with the top round part can be differenciated from the parameters associated to the bottom round part. In addition, the window-based method enables to measure global distances such as the distance between the center lines of two successive stems.

The window-based method offers only a semi-automatic system for parameter extraction: For each parameter we explained explicitly what has to been done in order to extract it from the data included in the relevant window. A step further would be the design of a high level language able to formally describe matching windows and measuring methods for each parameter. Once this is achieved, parameter position could be specified by a few instructions and parameter extraction is likely to become more automatic, at least for traditional typeface families.

References

1. D. Adams, "abcdefg, a better constraint driven environment for font generation", *Raster Imaging and Digital Typography (Eds. J. André, R.D. Hersch)*, Cambridge University Press, 1989, 54-70.
2. Bauermeister et al., Method and system for creating, specifying, and generating parametric fonts, *United States Patent No. 5586241*, Dec. 17, 1996.
3. R. C. Beach, *An introduction to the Curves and Surfaces of Computer-Aided Design*, Van Nostrand Reinhold, New-York, NY, 1991.
4. C.H. Cox, "Skeleton: A Link between Theoretical and Physical Letter Descriptions," *Pattern Recognition*, Vol. 15, No. 1, 1982, 11-22.
5. J. Herz, R.D. Hersch, "Towards a Universal Auto-Hinting System for Typographic Shapes", *Electronic Publishing (EP-odd journal)*, Vol. 7, No. 4, Dec. 1994, 251-260.
6. J. Herz, Coherent Processing of Typographic Shapes, *Ph.D. thesis n^{o}1676*, EPFL, Lausanne, 1997 (will soon be available on the WEB).
7. D. R. Hofstadter, METAFONT, Metamathematics, and Metaphysics, republished in *Metamagical Themes: Questing for the essence of mind and pattern*, pp.260-299, Bantam books, N. Y. 1985.

8. P. Karow, “Automatic Hinting for Intelligent Font Scaling”, *Proc. Raster Imaging and Digital Typography 89, (Eds. J. André, R.D. Hersch)*, Cambridge University Press, 1989, 232-241.

9. D. E. Knuth, *The METAFONT book*, Addison-Welsey, Reading Mass. 1986.

10. Clyde D. McQueen III and Raymond G. Beausoleil, “Infinifont: a parametric font generation system”, *RIDT 94, Electronic Publishing*, Vol.6(3), pp.117-132, John Wiley & Sons, 1993.

11. J. Rocha, T. Pavlidis, “A Shape Analysis Model with Applications to a Character Recognition System”, *IEEE PAMI*, Vol. 16, No. 4, April 1994, 393-404.

12. R. D. Hersch, C. Bétrisey, “Model-based matching and hinting of fonts”, *Proceedings SIGGRAPH’91, ACM Computer Graphics*, 25, 71-80, 1991.

13. A. Shamir, A. Rappoport, “Extraction of Typographic Elements from Outline Representations of Fonts”, *Proc. Eurographics 1986, Computer Graphics Forum*, Vol 15, No. 3, 259-268.