Parameterizable Fonts Based on Shape Components

Changyuan Hu and Roger D. Hersch
Ecole Polytechnique Fédérale de Lausanne, Switzerland

We propose a new, highly flexible font description method that explicitly describes characters as assemblies of parameterizable shape components. By varying global parameters, we can derive fonts that vary in weight, condensation, and shape.

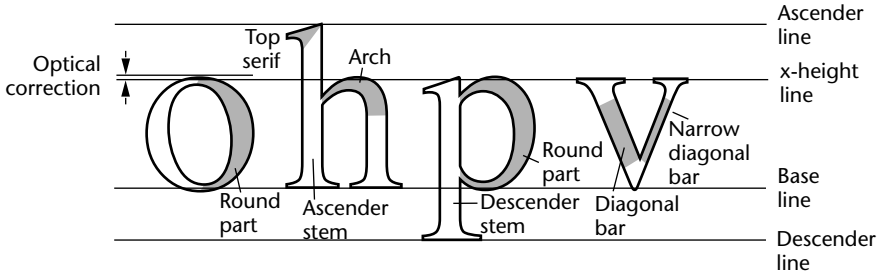
Typographic characters are carefully designed shapes incorporating type-design tradition,[1] the rules related to visual appearance, and the design ideas of a skilled character designer.[2] The typographic design process is structured and systematic: letterforms are visually related in weight, contrast, space, alignment, and style. To create a new typeface family, type designers generally start by designing a few key characters—such as o, h, p, and v—incorporating the most important structure elements such as vertical stems, round parts, diagonal bars, arches, and serifs (see Figure 1). They can then use the design features embedded into these structure elements (stem width, behavior of curved parts, contrast between thick and thin shape parts, and so on) to design the font's remaining characters.[3]

Today's industrial font description standards such as Adobe Type 1 or TrueType represent typographic characters by their shape outlines, because of the simplicity of digitizing the contours of well-designed, large-size master characters.[4] However, outline characters only implicitly incorporate the designer's intentions. Because their structure elements aren't explicit, creating aesthetically appealing derived designs requiring coherent changes in character width, weight (boldness), and contrast is difficult. Outline characters aren't suitable for optical scaling, which requires relatively fatter letter shapes at small sizes.[5] Existing approaches for creating derived designs from outline fonts require either specifying constraints to maintain the coherence of structure elements across different characters[6,7] or creating multiple master designs for the interpolation of derived designs.[8]

We present a new approach for describing and synthesizing typographic character shapes. Instead of describing characters by their outlines, we conceive each character as an assembly of structure elements (stems, bars, serifs, round parts, and arches) implemented by one or several shape components. We define the shape components by typeface-category-dependent global parameters such as the serif and junction types, by global font-dependent metrics such as the location of reference lines and the width of stems and curved parts, and by group and local parameters. (See the sidebar "Previous Work" for background information on the field of parameterizable fonts.)

Conceptual model and terminology

To provide highly flexible parameterizable fonts, we first describe a conceptual model for typographic characters based on shape components. A character's basic structure is font independent. Basic characters are made



1 Character structure elements incorporated into key typographic characters.

0272-1716/01/$10.00 © 2001 IEEE

Previous Work

The earliest work known to describe typographic characters by parameterizable shape primitives is that of Philippe Coueignoux, who designed one of the first fully digital typesetter controllers.[1]

The most serious published work done in the field of parameterizable fonts is the Metafont system, a programmable parameterizable font synthesizing system.[2] The Metafont system relies on a few basic paradigms for generating characters and symbols. The main character parts such as horizontal, vertical, and diagonal strokes and round parts are specified by describing the path of a pen with given orientations and pen widths. A sequence of pen positions and directions describes the pen's central path. From this information, Metafont computes a smooth centerline pen trajectory. With the given pen positions, widths, and orientations and with the centerline trajectory, Metafont infers the description of the corresponding pen stroke's boundary.[3] It adjusts serifs with font-dependent serif width, height, and depth information to the computed stroke boundary. The shape boundary resulting from the assembly of serifs and stroke can either be directly filled or traced by a small, circular pen and then filled. In addition to strokes defined by pen trajectories, Metafont specifies round character parts covering one or a multiple quadrants by superarcs, that is, scaled arcs defined within single quadrants without outlines whose boundaries are given by superellipses.[2]

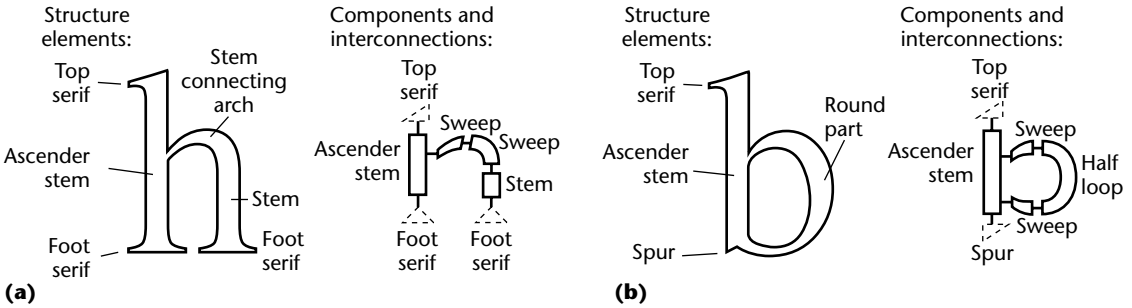
Donald Knuth used the Metafont program to create his Computer Modern typeface family.[4] He parameterized the Computer Modern typefaces so as to automatically generate optically scaled fonts and to generate sans-serif, typewriter, semibold, bold, condensed, and slanted roman fonts by a simple change of parameterizations. Metafont uses separate character shape descriptions for the italic font. Since Metafont is both a complete programming language and a flexible font design tool, using it requires both programming and typographic skills. Only a few individuals use Metafont, often for designing non-Latin characters.[5,6] One of the lessons learned by Metafont users designing Latin characters is that Metafont's pen paradigm doesn't offer sufficient freedom to exactly generate character shapes according to the designer's intention. This explains why most Metafont designs for Latin fonts besides Computer Modern rely heavily on outline descriptions.[7]

The recent Infinifont system (US Patent 5,586,241)[8] is a feature-based parameterizable font description and reconstruction system. Its authors describe the basic mechanism to assemble a character such as character E from parameterizable vertical bars, horizontal bars, and serifs. However, most of their work isn't published, so we don't know how they synthesize different typeface categories and which paradigms they use for synthesizing curved character shapes and serif variants.

Schneider describes a method for assembling parameterizable pen-based strokes into typographic characters.[9] Regarding Latin character shapes, the method suffers from the same limitations as Metafont. It seems, however, well suited for synthesizing Asian characters (such as Kanji and Hangul).

References

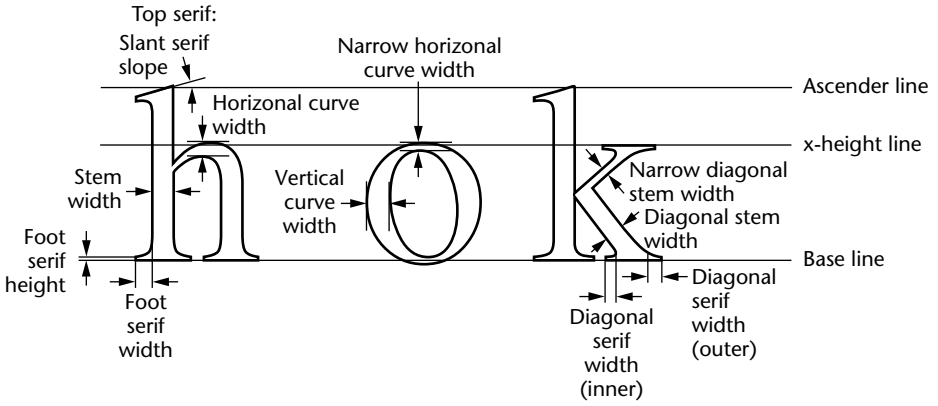
1. Ph. Coueignoux, "Character Generation by Computer", *Computer Graphics and Image Processing*, vol. 16, 1981, pp. 240-269.
2. D. Knuth, *The Metafont Book*, Addison-Wesley, Reading, Mass., 1986.
3. J.D. Hobby, "Rasterizing Curves of Constant Width," *J. ACM*, vol. 36, no. 2, Apr. 1989, pp. 209-229.
4. D. Knuth, *Computer Modern Typefaces* (Volume E of Computers and Typesetting), Addison-Wesley, Reading, Mass., 1986.
5. Y. Haralambous, "Typesetting Khmer," *Electronic Publishing: Origination, Dissemination, and Design*, vol. 7, no. 4, 1994, pp. 197-215.
6. R. Southall, "Metafont in the Rockies: the Colorado Typemaking Project," *Electronic Publishing, Artistic Imaging, and Digital Typography*, R.D. Hersch, J. André, and H. Brown, eds., LNCS 1375, Springer-Verlag, 1998, pp. 167-180.
7. N. Billawala, "Pandora, An Experience with Metafont," *Raster Imaging and Digital Typography*, J. André and R.D. Hersch, eds., Cambridge Univ. Press, Cambridge, Mass., 1989, pp. 34-53.
8. C.D. McQueen and R.G. Beausoleil, "Infinifont, A Parametric Font Generation System," *Electronic Publishing: Origination, Dissemination, and Design*, vol. 6, no. 3, Sept. 1993, pp. 117-132.
9. U. Schneider, "An Object-Oriented Model for the Hierarchical Composition of Letterforms in Computer-Aided Typeface Design," *Electronic Publishing, Artistic Imaging and Digital Typography*, R.D. Hersch, J. André, and H. Brown, eds., LNCS 1375, Springer-Verlag, New York, 1998, pp. 109-125.



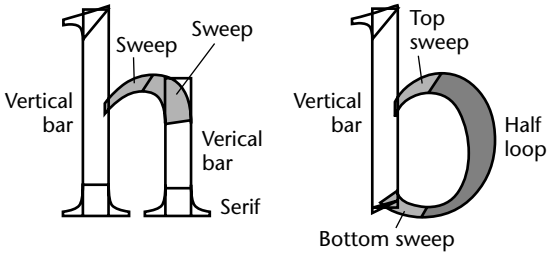
2 Component-based font-independent character models.

of stem, bar, sweep, half-loop, serif, and terminal components. Figure 2 shows the set of components and their interconnections for the characters h and b.

By specifying the junction between components (for example, the junction between a stem and a sweep) and by associating absolute and relative metrics to the set of



3 Global font parameters.



(a) **(b)**

4 Description of the specific features of the Times characters h (a) and b (b).

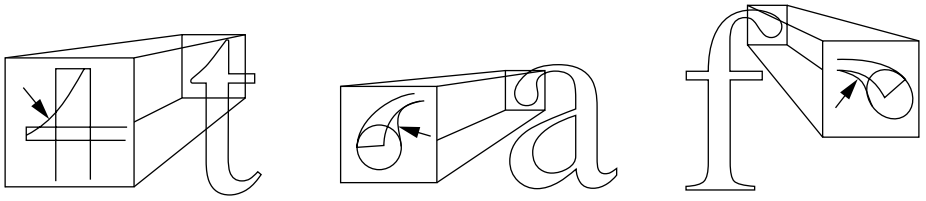
available parameters, we can fully specify character instances. We associate a software object with each character. A software object incorporates different character-synthesis methods that can support the generation of characters of different typeface categories—that is, of characters whose junctions between components considerably vary. We use global parameters to specify junction types, serif and terminal types, reference line positions, bar and stem widths, and serif and terminal metrics. Group parameters define the metrics associated with a group of related characters—for example, the junction depth in the characters b, d, p, and q. Local parameters are parameters describing features specific to a single character. For each font, an external file describes its global parameters (see Figure 3), its group parameters, and the local parameters associated to each character. We express local and group parameters as percentages of global parameters.

Typeface categories relate to character structure. Typefaces with strongly different structures—for example, different component junctions, different serif types, and possibly different character round-part parameters (such as squareness and obliqueness)—belong to different typeface categories. Strongly modifying a feature related to a typeface category changes the character structure, or earmark.[9] Modifying global width metrics changes relative weight, contrast, and certain character features (such as junctions, serifs, and terminals) without changing the basic character structure. Therefore, at the conceptual level, we propose two levels of design spaces: one at the structural level (typeface category) and one at the metric, boldness, and contrast level.

Regarding the terminology we use, *structure element* refers to typographic structures, and *shape component* (or component) refers to a set of well-defined parameterizable geometric shapes. *Metrics* generally refers to relative sizes and distances.

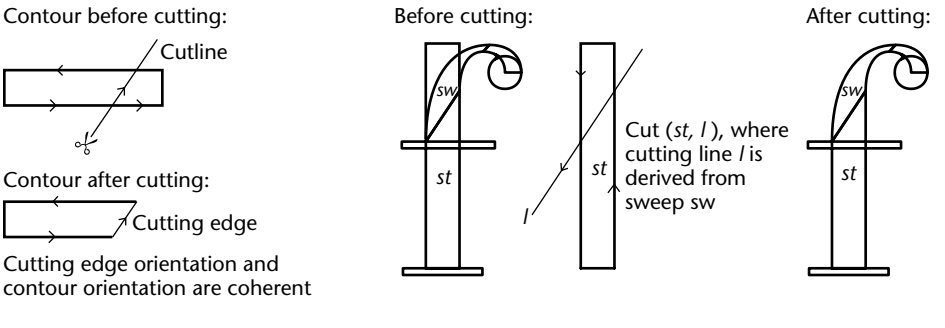
The basic shape components we use to implement structure elements are vertical, horizontal, and diagonal bars; the half-loop component used for round parts covering two quadrants; and the sweep component for connecting between two stems (see Figure 4a) and between a half-loop and a stem (see Figure 4b). We also use the sweep component to synthesize specific curved parts of the characters a, g, and s. We use serifs and ellipse-like components to synthesize terminal elements.

We can't synthesize some characters by simply assembling shape components such as bars, round parts, and ellipse-like terminal elements. Therefore, we introduce boundary correction curves, generally in the form of cubic Bezier spline curves, to produce the desired character outline shape (see Figure 5a) or to smooth out the junction between two shape primitives (see Figures 5b and 5c).



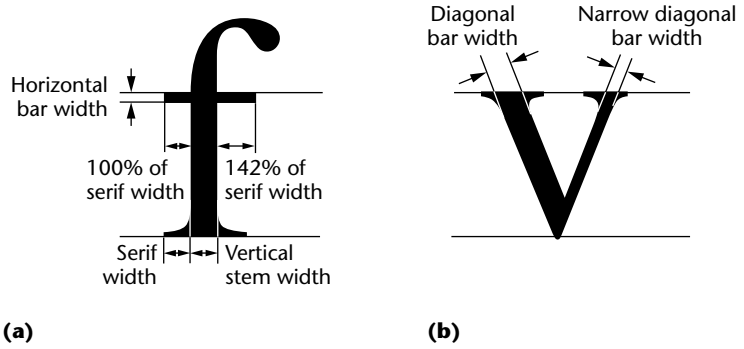
(a) **(b)** **(c)**

5 Application of boundary correction curves for synthesizing the Times characters t (a), a (b), and f (c).



6 (a) Definition and (b) use of the cut operator. Figure 6b uses a cutline to synthesize character f starting with a vertical bar and a top round part made of two sweeps and an ellipse-like terminal.

A further operator necessary for assembling character shape components into characters is the cut operator. It enables cutting a component into two parts and keeping one of the two parts for assembling the final character. The cutline orientation specifies the part we'll keep (see Figure 6a).



7 Characters incorporating vertical stems, horizontal, and diagonal bars.

Describing characters by structure elements

Traditional Latin letter shapes that derive from Antiqua and Grotesque (sans serif) typefaces[1] can be decomposed into basic structure elements: vertical stems, round parts (also called curved shape parts or bowls), arches, horizontal bars, diagonal bars, serifs, and terminals (see Figure 1). Conversely, we aim to synthesize traditional letter shapes by coherently assembling such basic structure elements.

The challenge lies in defining parameterizable geometric shapes (components) with which we can synthesize most instances of structure elements. We propose both components for structure elements representing straight strokes such as stems, horizontal bars, and diagonal bars and components for curved structure elements such as round parts and arches. We use similar components to describe serifs and terminals. To support different typeface categories, we also propose a set of standard parameterizable junctions between shape components.

Parameterizable components for straight structure elements

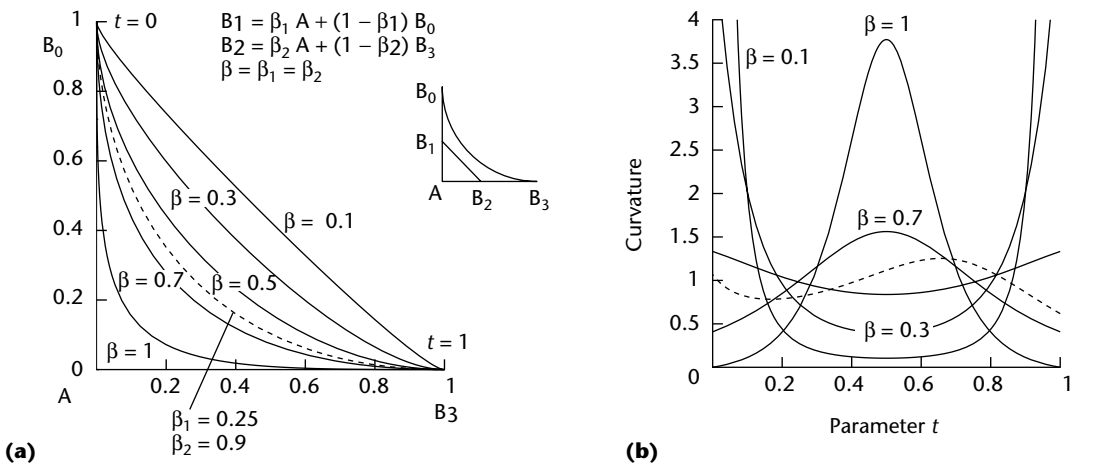
We describe vertical stems, horizontal bars, and diagonal bars with one point located on their centerline, their orientation, and their respective width, which is a global font parameter. Vertical stems and diagonal bars are bounded by their respective reference lines (baseline, x-height line, descender line, ascender line, or capsline). The length of horizontal bars, which are generally defined as local parameters (for example, in the characters f and z) can be given as percentages of other predefined parameters such as serif width or letter width (see Figure 7a). Stem, horizontal bar, diagonal bar (see Figure 7b), and curved element width are important global parameters that we use to synthesize fonts of variable weight, contrast, and condensation.

Parameterizable shape primitives for round structure elements

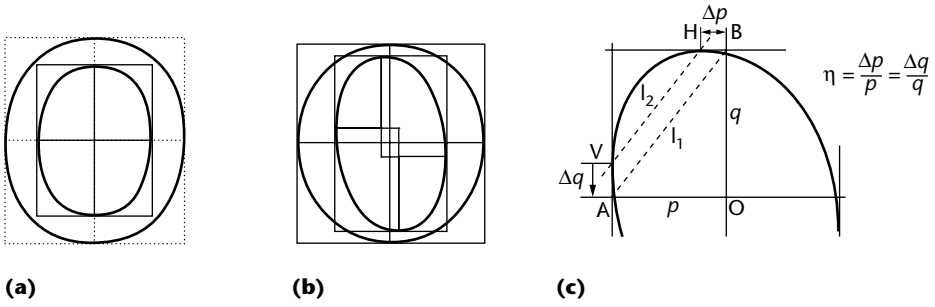
Round structure elements are difficult to synthesize. Both external and internal boundaries need to be generated to obtain the desired round-shape part. Let's first see how we can synthesize round-shape parts, or loops, covering one or several quarters of an arc. Such loops, for example, appear in the lower-case characters o, c, e, g, b, d, p, and q. The definition of loops must be flexible enough to synthesize quite different shapes, such as the round parts of the characters e, o, and b for different typeface families (such as Times and Bodoni).

Modeling quadrant arcs. To synthesize loops, let's analyze the flexibility offered by cubic Bezier splines with control polygon given by vertices $B_0B_1B_2B_3$ for generating quadrant arcs—that is, curves that have a horizontal and a vertical tangent at their endpoints. We define by parameters $\beta_1=\overline{B_0B_1}/\overline{B_0A}$ and $\beta_2=\overline{B_3B_2}/\overline{B_3A}$ the relative positions of the Bezier polygon control points B_1 and B_2 (Figure 8a, next page). By varying parameters β_1 and β_2, we can obtain families of curves whose *apex*—the point with tangent parallel to baseline B_0B_3—lies within circumscribed triangle B_0AB_3. Since curvature conveys the visual information associated with a given arc, Figure 8b gives curvatures corresponding to the plotted Bezier splines. Experience shows that curves having a high curvature at their end points and low curvature at their central part (for example, curves with $\beta < 0.4$) aren't visually pleasant.

We analyzed the Bezier splines (quadrant arcs) defining the external and internal contours of the character o in various font families. In most cases, β_1 and β_2 values are similar, and we can therefore merge them into a sin-



8 (a) Family of cubic Bezier splines covering a quarter of an arc and (b) their respective curvatures, obtained by varying parameters β_1 and β_2.



9 Quadrant arcs describing the (a) Helvetica and (b) Times character o, (c) modeled by their bounding-box and by their obliqueness η.

gle β value expressing an arc's squareness. Even in the case of widely different β_1 and β_2 values, we can compute an intermediate common β value by minimizing a function giving the difference between the original Bezier spline and its approximation with a single $\beta_1 = \beta_2 =$ β value. We minimize the sum of the squares of the distances between points of the new Bezier spline $B^{new}(u)$ at parameter values u = {0.1, 0.2, 0.3, 0.4, 0.45, 0.5, 0.55, 0.6, 0.7, 0.8, 0.9} and the original spline $B(t)$ to obtain good visual results.

We have computed the single beta values for the external and internal contours of character o, for many different typefaces.[10] The distance between the original contour and the contour synthesized with single beta values is negligible (less than 1/1000 of the capital letter height).

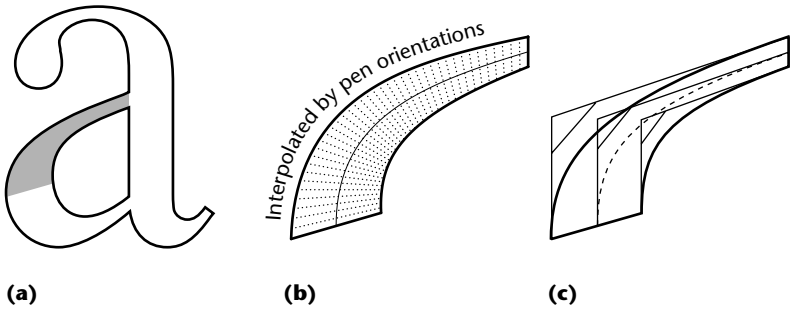
Modeling loops. A loop is a curved character-shape part covering a single or several quadrants. For example, we can model the Helvetica character o with a single loop covering four quadrants, the exterior, and respective interior contours defined by four connected quadrant arcs having a common center (see Figure 9a). The Times character o, however, is more complex. Although we can model its exterior contour—which looks like an upright ellipse—in the same manner as the Helvetica character o's contours, its interior contour looks like an ellipse with an oblique orientation and would require circumscribed quadrants with four different centers to describe it by quadrant arcs (see Figure 9b).

A simpler way to model oblique ellipse-like contours is by computing their bounding box. The bounding box center represents the center of symmetry of the pseudoellipse. We can show[10] that, within a single quadrant, the line $\overline{VH}$ connecting the horizontal and vertical tangential points is parallel to the line connecting points $A = (\pm p, 0)$ and $B = (0, \pm q)$ of the ellipse's bounding-box (see Figure 9c). Therefore, only a single obliqueness parameter $\eta = \Delta p/p = \Delta q/q$ giving the relative offset of the horizontal and vertical tangential points is necessary to define the four quadrant arcs making up an oblique ellipse-like contour.

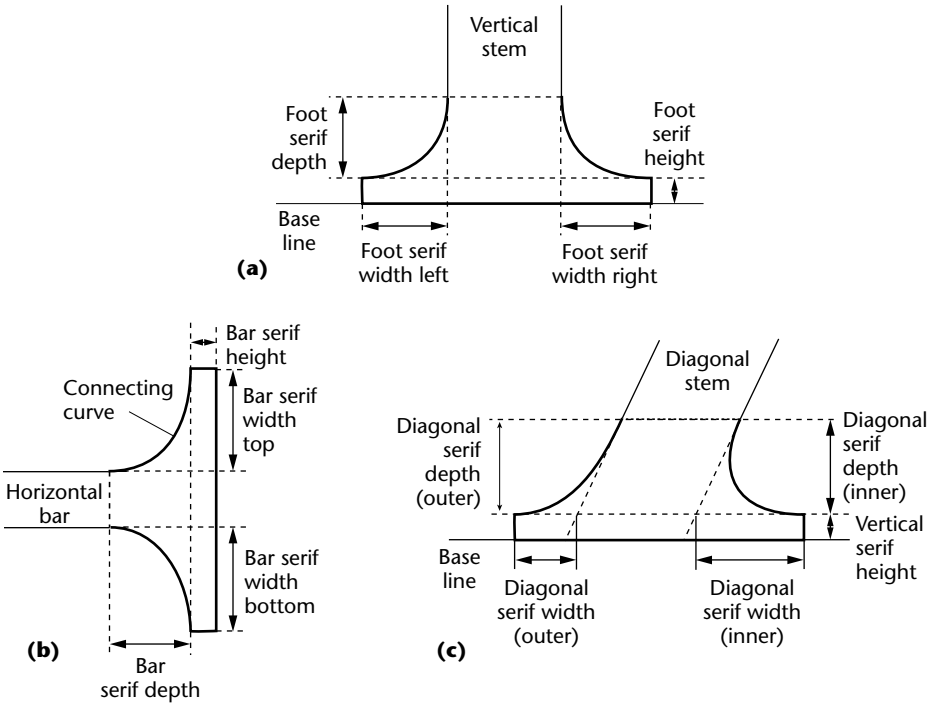
We model a full loop with exterior and interior ellipse-like contours having upright or oblique orientations. Such a full loop has a single center, which is the center of symmetry of both the exterior and the interior ellipse-like contours. However, half-loops used for synthesizing the round parts of the characters b, d, p, and q may have disjoint centers, one for the external contour and one for the internal contour.

The characterization of loops by two parameters, β for the squareness and η for the obliqueness of interior and exterior arcs, offers great design freedom for creating full or partial loops. It also maintains the coherence across characters incorporating full loops (o), half-loops (c, b, d, p, or q), and quarter-loops (e).

Modeling sweeps. We can use loops to render the round parts in the letters b, d, p, and q. When looking at one of these characters, we can see that the curved part connecting the loop to the bar shows a particular behavior. Its parameters aren't directly related to the



10 The sweep component. (a) Original sweep extracted from the Times character, (b) sweep synthesized by linear interpolation between sweeping pen departure and arrival orientations, and (c) improved sweep obtained by synthesizing its boundary control polygons.



11 Foot (a), bar (b), and diagonal (c) serifs and their basic parameters.

loop's parameters. Similarly, the arches of the characters h, n, m, and u aren't parts of loops. They're curved character parts connecting two vertical stems and therefore need their own shape description.

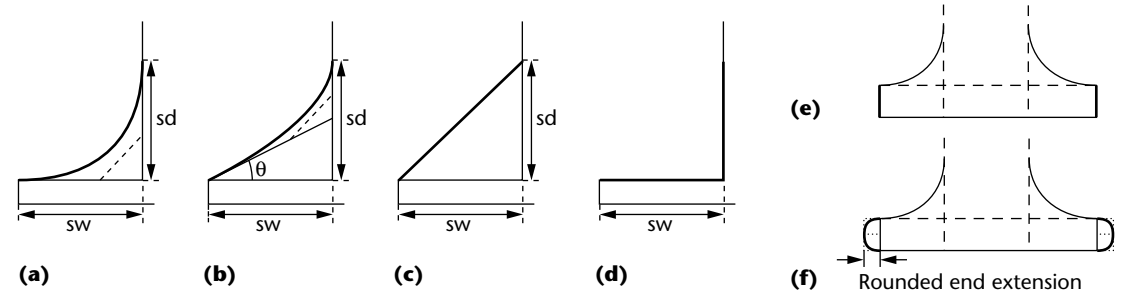
To support connecting elements made of curved parts, we introduce the sweep-shape primitive. Traditionally, sweeps were defined by a pen of a given shape and orientation sweeping along a centerline described by a Bezier spline.[12,11] Pen width and orientation are often given at sweep departure and arrival points and may be interpolated according to the centerline position parameter *t*. According to our observations, the sweeping pen paradigm isn't always suitable for generating Latin typographic characters. Especially if the sweep incorporates a long, flat part and strongly varying pen diameters at sweep departure and arrival positions, the resulting sweep (see Figure 10b) considerably diverges from an ideal sweep (see Figure 10a).

We can obtain a higher quality sweep primitive by generating for the left and right sweep boundaries Bezier splines that strongly resemble the centerline Bezier spline (see Figure 10c). We can achieve this by enforcing the same tangent directions at endpoints as the tangent directions of the centerline spline and by using the same $\beta 1$ and $\beta 2$ values. The generated control polygons for the left and right boundaries differ from one another by the size relationships of their respective B_0A and AB_3 control triangle sides. We can easily obtain elongated sweeps by using different $\beta 1$ and $\beta 2$ values for the centerline Bezier control polygon.

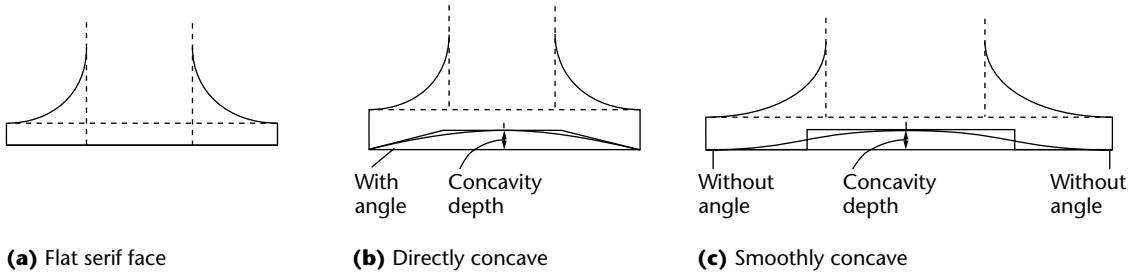
The sweep component is useful for establishing the connections between a loop and a vertical stem and between two vertical stems. We also use it to create curved parts such as the tail of the character g and the round parts of the characters a, which we can't model with quadrant loops.

Parameterizable terminal elements

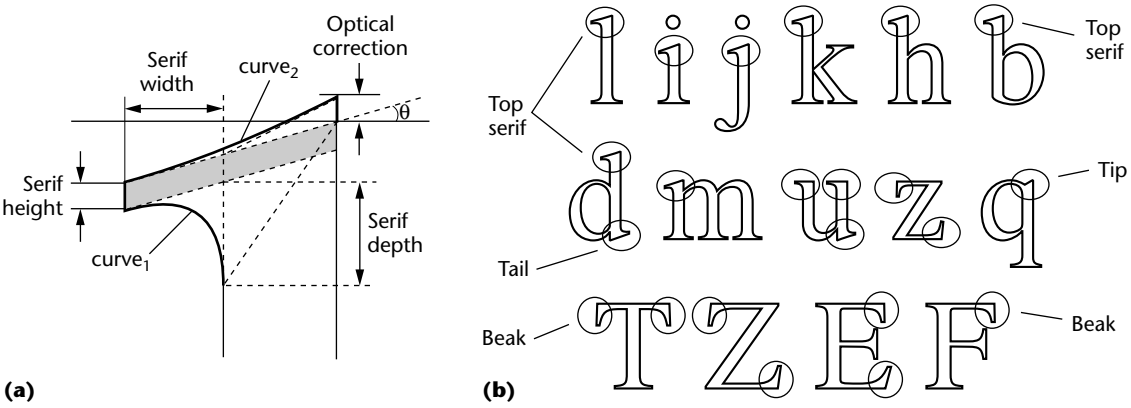
The shape of terminal elements such as serifs, bulbs (see Figure 5b and 5c), and ears (top right part of the character g) greatly determines a typeface's look. Terminal elements at the end of straight stems and bars are standard serifs, such as foot, top, bar, and diagonal serifs. They're composed of predefined elements whose main metrics are given by global parameter values (see Figure 11). To accommodate the large variety of serif



12 Variations in serif brackets: (a) smoothed, (b) angled, (c) straight, and (d) none, and serif ends: (e) butt and (f) rounded.



13 Variations of the serif face



14 (a) Description and (b) occurrences of slant serifs.

types,[9] we introduce variations in bracket styles, serif end styles and serif faces.

We can explicitly define terminals located at the end of curved strokes with a set of specific shape primitives that can comprise pears (ellipse-like elements) and small bars, as well as boundary-correction curves.

Serifs

Serifs are the most important terminal elements. Researchers have extensively described the foot and bar serifs (see Figures 11a and 11b) in the literature.[4,7,13] Half-serifs on each side of a stem or a bar are defined by three basic parameters: the serif width, the serif height, and the serif depth. In addition, serifs are characterized by their brackets—that is, by the way the serif slab is connected to the main stem or bar and by the serif ends (see Figure 12). The serif face can also vary (see Figure 13).

Thanks to the basic serif parameters, the variations in brackets, serif ends, and the serif face, most of the serif styles that Rockledge and Perfect[9] and Bauermeister[14] describe can be synthesized. If required, we can incorporate additional variations in serif ends and faces into the serif component model.

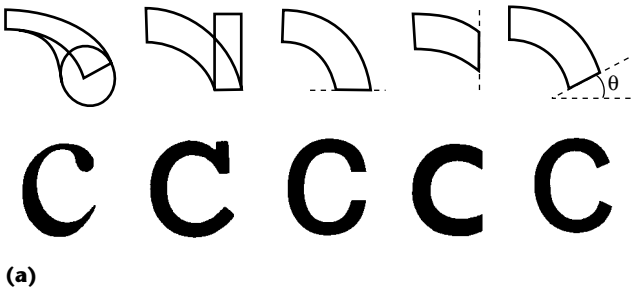
The slant serif is designed mainly for top serifs. We can also use it to synthesize the shape of beaks and tips as in the characters z and T (see Figure 14b).

We can describe a slant serif (see Figure 14a) by the intersection of its stem and of a small diagonal bar, called a slab, with orientation θ, by a certain amount of optical correction, by the curve connecting the stem to the slab ($curve_1$), and by the equivalent of the serif face—that is, a curve connecting the slab end to the optical correction point ($curve_2$). In the case of the Times character u, the slab is horizontal and doesn't require optical correction.

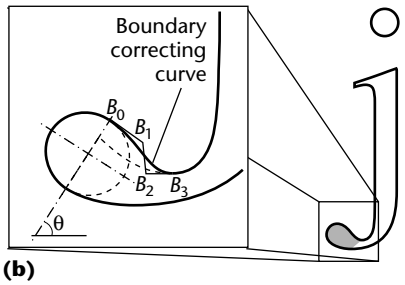
We can also apply the design variations we know from the foot and bar serifs—variations in stem to slab connection, variations in serif ends and variations in serif face ($curve_2$) to slant serifs.

Terminal elements and dots

For serif typefaces, terminals of curved strokes such

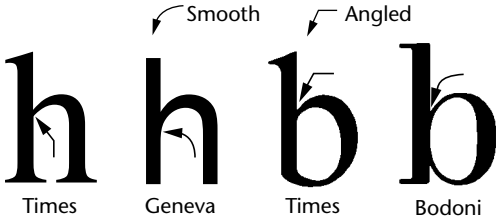


(a)



(b)

15 (a) Representation of curved stroke terminals and (b) junction between ellipse-like terminal element and its associated sweep component.



16 Junctions between vertical bar and sweep

as bowls, arches, and tails often incorporate pears and small bars (see Figure 15a).

Times characters such as a, f, j, and r have bulbs that are oblique, ellipse-like terminal elements at the end of their round strokes. We can obtain these terminal elements by rotating pseudoellipses made up of quadrant Bezier arcs and by smoothing out the junctions between ellipse-like terminal elements and associated sweeps. Figures 5b, 5c, and 15b show terminal elements made up of sweeps, pseudoellipses, and boundary-correction curves.

The junctions between ellipse-like terminal elements and associated sweeps are smoothed out by a boundary-correcting curve whose departure and arrival tangents are also tangents of the two intersecting outline segments (see Figure 15b). We give the length of each of the correcting curve Bezier-control-polygon segments B_0B_1 and B_2B_3 as a percentage of a global parameter.

Sans-serif typefaces don't generally incorporate any decoration at the end of curved strokes. Their curved strokes are terminated abruptly, leaving the end squared or pointed. An angle θ might define the orientation of the terminal's end segment (see Figure 15a).

We can also model the dots on top of the characters i and j as ellipse-like elements with appropriate β values defining their squareness.

Junctions between character components

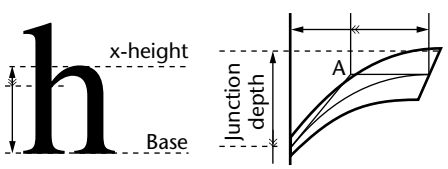
We aim to synthesize traditional Latin characters using bar components for straight structure elements and loop and sweep components for round structure elements. The way components are interconnected greatly influences a typeface's design. We first define features enabling design variations for the junctions between round parts and stems (vertical bars). Then, we introduce the parameters and geometric constructions enabling the synthesis of design variations for junctions between diagonal bars and other bars.

Junctions between vertical stems and sweeps

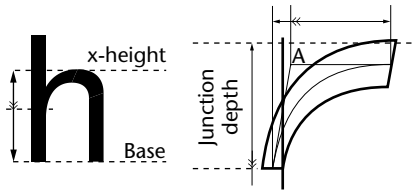
The junctions between vertical stems and sweeps define the interconnection between straight- and round-character structure elements. Generally, junctions are either angled or smooth. Figure 16 shows angled and smooth junctions for the characters h and b. The junctions in the character h are representative of the junctions in the characters n, m, and u. The angled and smooth junctions in the character b are representative of the junctions in the characters p, q, and d.

The sweep at the junction between straight and round-character structure elements determines the junction's visual appearance. Besides being smooth or angled, the junction depth parameter defines, as a proportion of the character height, the vertical distance between the corresponding horizontal reference line and the junction. The location of the control triangle's vertex A of the sweep's centerline Bezier spline also defines the sweep's shape. As Figure 17a shows, we give vertex A's horizontal position as a percentage of the horizontal distance between centerline Bezier start and end points.

The control triangle vertex A's junction depth and relative position define the connecting sweep's shape. By considering the control triangle vertex A's junction depth and relative position as group parameters (or as valid

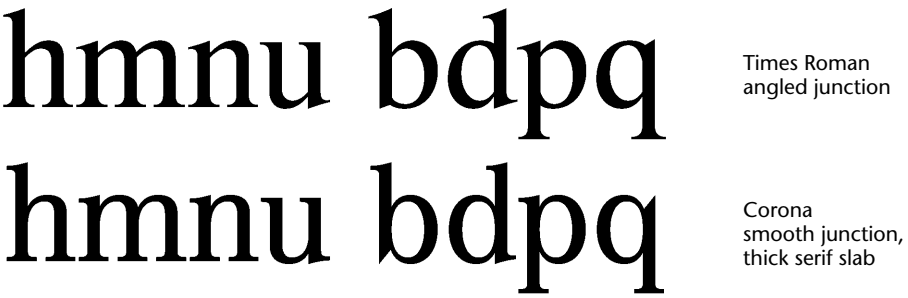


(a)



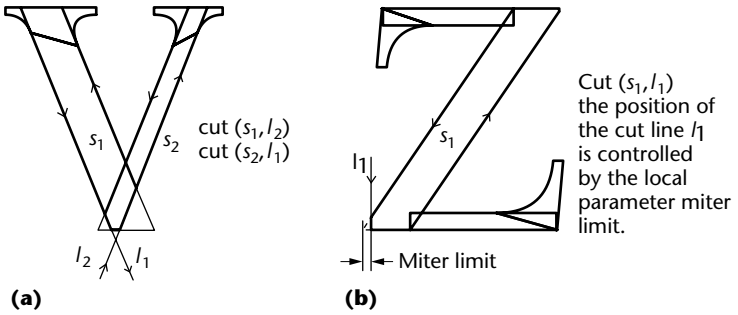
(b)

17 Defining the shape of the connecting sweep by the junction depth and the relative horizontal position of centerline Bezier spline control triangle vertex A.

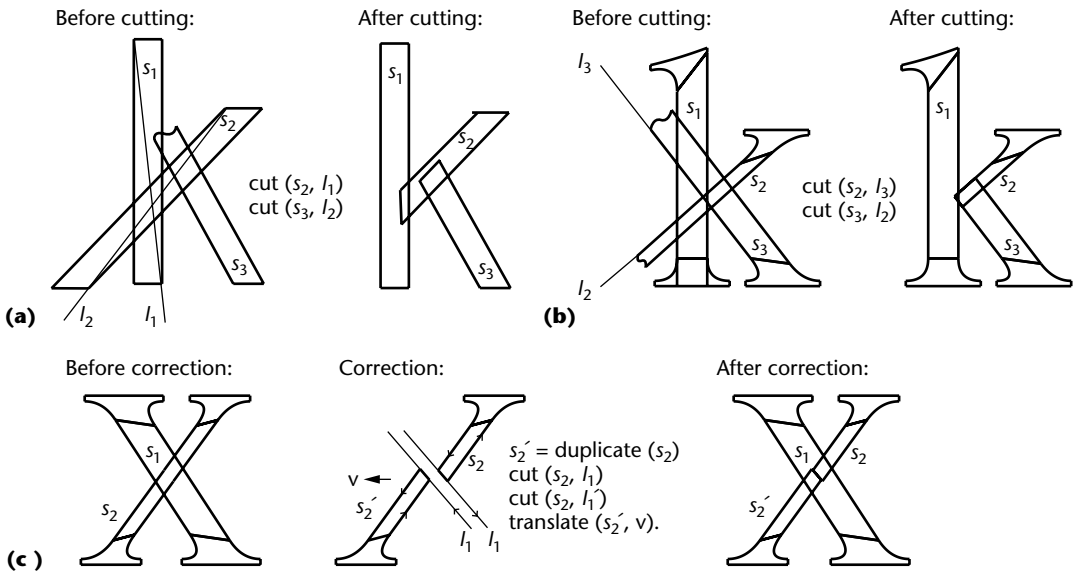


18 Synthesis of coherent Times and Corona characters.

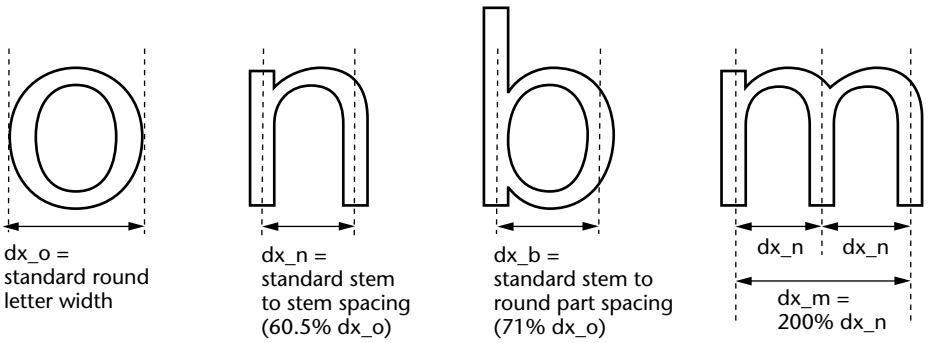
19 Coherent shape modifications across a group of characters (Times).



20 Synthesizing bar junctions of characters v (a) and z (b).



21 Bar intersections for the characters k and x.



22 Horizontal spacing of character structure elements (values for Times Roman).

parameters for the group of characters a, h, m, n, u, b, d, p, and q), we can generate coherent character shape modifications across these characters. Figure 18 shows coherent Times Roman and Corona characters that were resynthesized with components. Figure 19 shows coherent variants of Times Roman, generated by modifying the junction depth parameters. In the same way that Times Roman and Corona belong to different typeface categories, our experimental Times Flat and Times Round also belong to different typeface categories. They represent an attempt to explore a small part of the traditional typeface design space.

Junctions between diagonal bars and other bars

A bar is given by two points of its centerline and by its width, which is generally a global parameter (such as VerticalBarWidth, NarrowVerticalBarWidth, and so on). To synthesize characters such as v and w, we must intersect two diagonal bars and appropriately reshape them with cutlines (see Figure 20a). To synthesize the character z, an additional relative parameter defines the miter junction between the diagonal and the horizontal bar (see Figure 20b).

Character k comprises either a single or a double junction between its vertical bar and its two diagonal bars. As Figure 21a and 21b shows, we apply the cutline operator differently if a single or a double junction is present.

Character x consists of the intersection of two diagonal bars. Type creators[15] know that we need an optical correction: the departure of the lower part of the narrow diagonal is displaced horizontally to the left by 15 percent of its horizontal width (see Figure 21c).

Synthesis of characters with parameterizable elements

The components we described in the previous sections are the basic building blocks for synthesizing characters. To synthesize specific characters with completely defined metrics and shapes, the character designer needs to supply global, group, and local parameters.

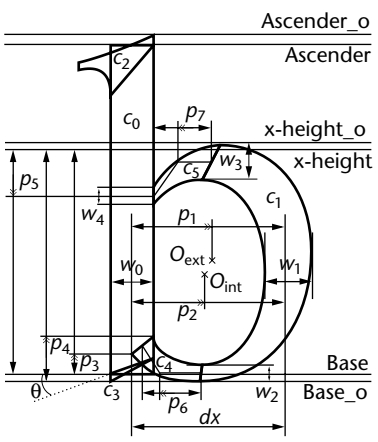
For example, consider the character b (see Figure 2b). A global lower-case stem width parameter defines the ascender stem. Global parameters specifying the serif type and the serif's main metrics define the top serif, and a local parameter defines the spur. The round part comprises a half-loop component and two connecting sweeps, defined by group and local parameters.

We'll now describe how we synthesize characters from their parameterized component description. The font's horizontal reference lines (baseline, x-height, caps-height, ascender and descender lines, as well as the corresponding optical correction lines) define exactly the placement of the character's components. The character synthesizing method places the character's main parts (the stem and half-loop) horizontally apart according to global spacing parameters. Secondary parts such as serifs or connecting arches are placed so as to correctly connect to the main parts.

The font creator defines global spacing parameters for the standard round-letter width, for the horizontal spacing between vertical stem and half-loop (the spacing between the stem and round parts of character b), and for the horizontal spacing of two vertical stems (see Figure 22). These global spacing parameters are a proportion of the standard round-letter width parameter.

Figure 23 (next page) illustrates how the character synthesizing software computes the character b. We synthesize its vertical bar using the standard stem width and position it at an arbitrary horizontal position. The half-loop's center O_{ext} is vertically in the middle of its respective reference lines (xheight_o and base_o). We compute the vertical position of center O_{int} as a function of the reference lines and the respective horizontal curve part widths (w_2 and w_3). The centers' horizontal locations are given by the vertical stem to half-loop spacing (dx) and by group parameters p_1 and p_2 specifying their relative horizontal positions between stem midline and half-loop midline. The half-loop's internal and external arcs are defined by their center positions, bounding boxes, obliqueness parameters η_{int} and η_{ext}, and squareness parameters β_{int} and β_{ext} (according to global parameters).

After placing the half-loop component, we can place the angled sweep components connecting the half-loop component to the vertical bar. Their starting sweep position and orientation are given by the half-loop component's extremities. The ending orientation of sweep C_5 is vertical, and the junction depth group parameter p_5 defines its position. The ending points of sweep C_4 are given by group parameters p_3 and p_4, (parameters valid for the characters b, d, p, and q). Slant serif component C_2 is defined by the global parameters defining slant serifs and is inserted at the top of the vertical bar. Compo-



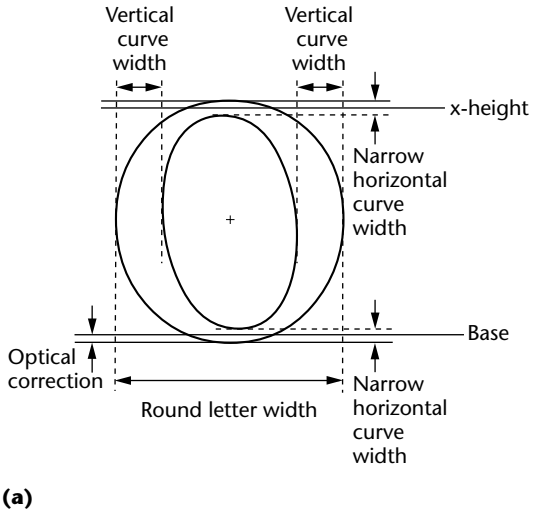
23 Synthesis of the character b with its parameterized elements.

Components:
- c_0 : stem, c_1 : half-loop covering quadrants IV, I
- c_2 : slant serif, c_3 : spur (slant serif)
- c_4, c_5 : connecting sweeps

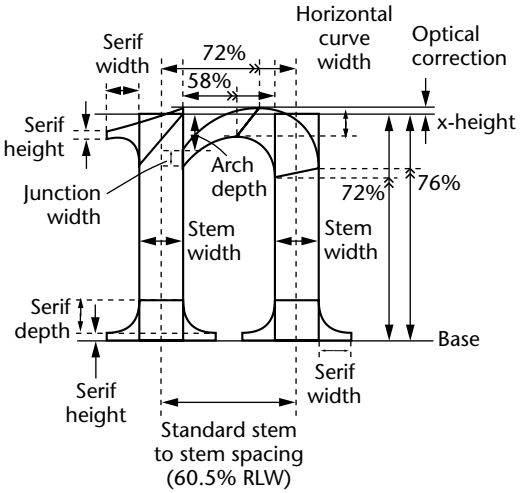
Global parameters:
- dx : standard vertical stem to half-loop spacing
- w_0 : the standard vertical stem width
- w_1 : the standard vertical curved part width
- w_2 : the standard narrow horizontal curved part width
- w_3 : the standard horizontal curved part width
- w_4 : width of the junction, often equal to w_2
- η_1, η_2: loop extreme corrections, external and internal

Group parameters (the group of b, d, p and q):
- p_1 : relative h-position of loop external center
- p_2 : relative h-position of loop internal center
- p_3, p_4 : relative h-position of the lower junction (c_4)
- p_5 : relative depth of the upper junction (c_5)
- p_6 : relative control point position of the sweep c_4
- p_7 : relative control point position of the sweep c_5

Local parameter:
- θ : spur angle



(a)



(b)

24 Parameters needed to synthesize the Times Roman characters o (a) and n (b).

nent C_3 (spur) is described by a slant serif component with zero width and zero height.

Font description parameters and implementation issues

Flexible character shape descriptions, which we can modify to create derived versions with varying width, weight, and contrast, need to incorporate their most significant features as percentages of modifiable global font parameters. For example, the round-letter width (RLW) defined as the horizontal width of the letter o is a global parameter used as a reference to describe the basic spacing of stems and curved elements (half-loops) of all lower-case characters of a font (see Figure 22).

The global width parameters specify the respective thicknesses of vertical stems, horizontal bars, diagonal bars, vertical curved parts, and horizontal curved parts. We also use proportions of their width to describe the starting and ending width of associated sweep primitives—for example, width w_4 of the sweep component c_5 in the letters b (see Figure 23) and n (see Figure 24b). Since width parameters distinguish between the thicknesses of vertical, horizontal, and diagonal strokes as well as between thin and thick strokes (Figure 7b), they both govern a character's *weight* and the amount of *contrast* incorporated into the font.

The global serif parameters govern the shapes of the serifs by specifying the serif type given by the selected variation in bracket, serif end, and serif face and by specifying their width, height, depth, and slant angle (see Figures 12 through 14).

As further examples, let's look at the definition of the Times characters o and n in Figure 24. Character o doesn't need any local parameters. We can completely synthesize it using global font parameter information. Synthesizing the character n requires placing the first stem at an arbitrary starting position and placing the second stem according to the given stem-to-stem spacing value. The serifs are defined according to global font parameters. The two sweep components are defined by global and group parameters. Figure 25 shows lower-case Times Roman characters recreated with components. Figure 26 shows the components and resulting characters for a few representative upper-case Times Roman characters.

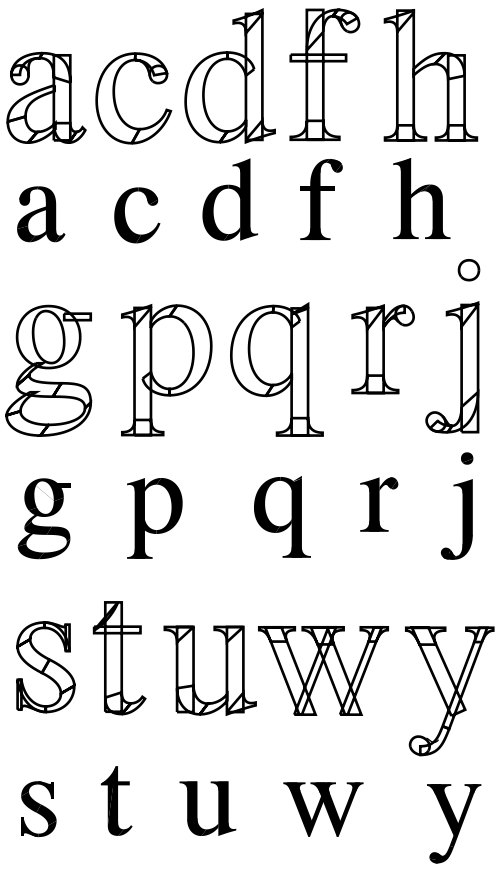
Component-based font descriptions require global, group, and local parameters. Table 1 shows how many parameters we need for our versions of resynthesized lower-case Times, Helvetica, and Bodoni characters. Bodoni requires fewer parameters, because all its serifs are slab serifs (uniform height). The sans-serif font Helvetica further reduces the number of required parameters.

A 26 lower-case character font with the complexity of Times requires approximately 540 parameters, or 1,080 bytes. Comparatively, TrueType requires approximately 1,500 bytes for the global parameters and a mean of 154 bytes per character for storing the outlines of lower-case characters. The grid-fitting instructions needed by TrueType for character generation at medium and low resolution require an additional 385 bytes per character. In contrast, component-based character descriptions already include all information about their structure (stems, half-loops, bars, connecting sweeps, and serifs). In addition, component-based fonts let us generate derived fonts (such as condensed and semibold) by changing the values of a few global parameters. A single component-based font can therefore replace several traditional outline fonts. Thus, we expect component-based fonts to require an order of magnitude less storage space than traditional outline fonts.

With our current software, we can synthesize characters by rasterizing the partly overlapping components and by excluding components that aren't part of the character, by using the cut operation (see Figure 6). Also, we can recover the full character outline by using a variant of Vatti's algorithm[16] to assemble components made up of straight and curved outline segments.

Table 1. Number of parameters for a font comprising the lower-case character a to z.

	Times	Bodoni	Helvetica
Number of global parameters	110	98	63
Number of group parameters	31	31	31
Mean number of local parameters per character	15.3	16.0	14.0



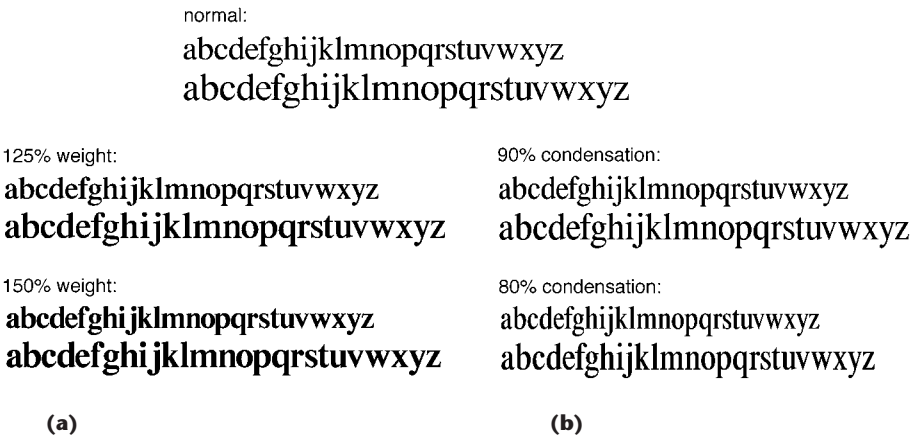
25 Lower-case Times Roman characters recreated with components.

Synthesizing derived characters by varying global font parameters

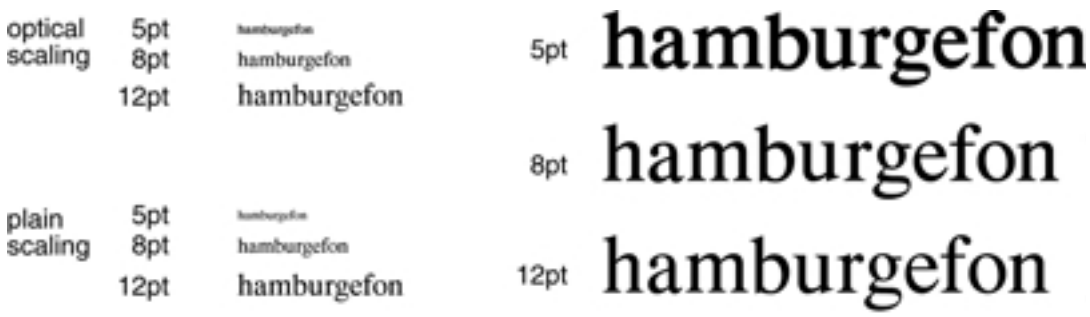
Here, we describe experiments showing the effects we can obtain by varying some global font parameters. Increasing the vertical stem width and vertical curve width parameters and decreasing the serif width parameters increases a font's weight. Figure 27a (next page) shows the alphabet at different sizes, at normal weight, and at 125 percent and 150 percent weights.

High-quality horizontally condensed fonts are needed where space is scarce—in telephone books for example. Furthermore, condensed fonts may offer increased flexibility for information presentation at display resolution—for example on Web pages.[17] Reducing the character width without reducing in the same proportion the thickness of the strokes generates high-quality horizontally condensed fonts. Because individual character width is a function of the round letter width, reducing the global round letter width parameter's size ensures the width reduction of the font's characters. Figure 27b shows text condensed to 90 percent and 80 percent. Condensation down to 90 percent is barely perceptible and enables the generation of high-quality characters. Eighty percent condensation considerably distorts the original character shapes.

26 Resynthesized Times Roman capital letters



27 Lower-case alphabet at different weight (a) and condensation factors (b).



28 Comparison of optical scaling and plain scaling. (a) Characters printed at high resolution, with (top) and without (bottom) optical scaling. (b) Optically scaled character shapes at large sizes.

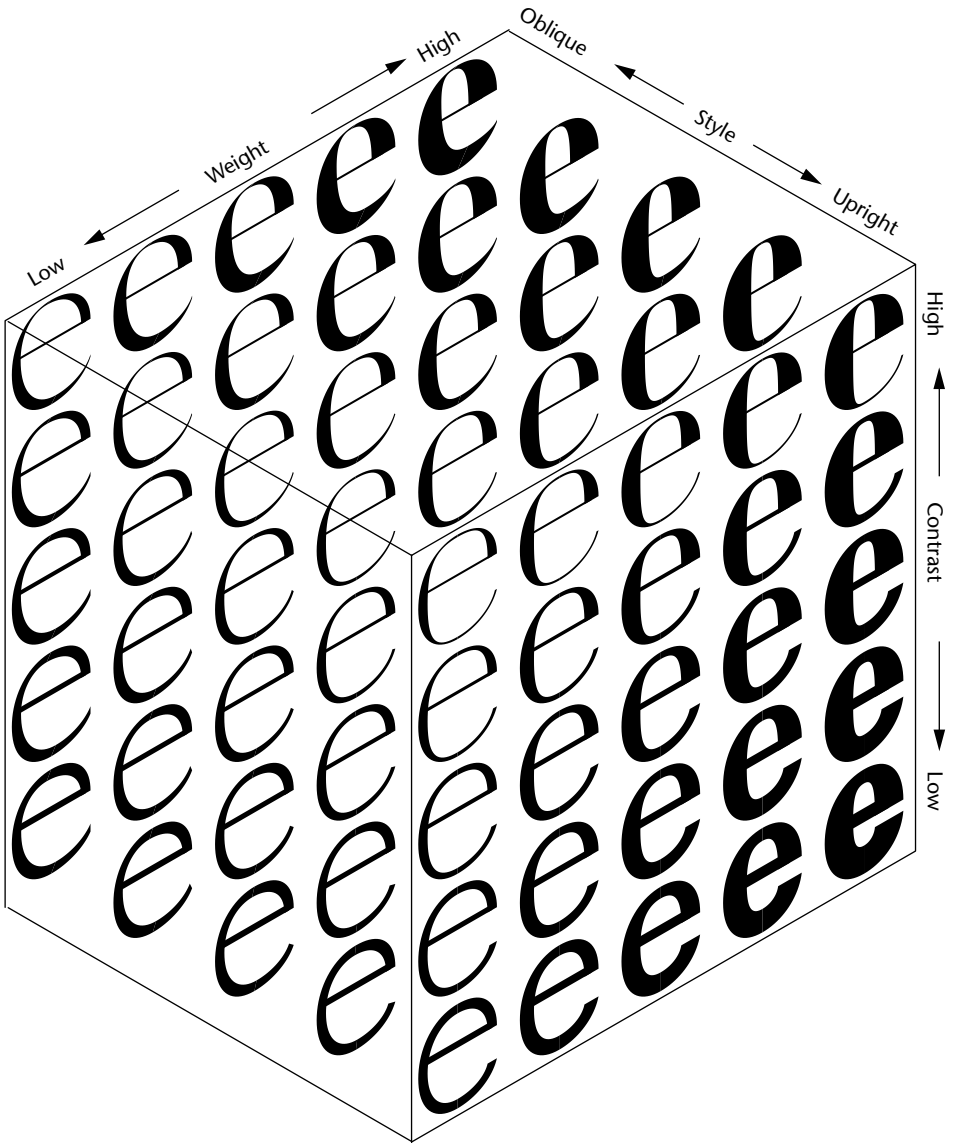
Optically scaled fonts are relatively fatter and larger at small sizes.[5] Their x-height and character width are slightly larger than corresponding values obtained by plain scaling. In the following optical scaling example, we use optical scaling correction factors to generate characters between 5 and 12 points. We experimentally determine maximum correction factors (maxFact) for the various parameters (letter, stem, bar, and curved element widths) for the 5 point character size. Between 5 and 12 points, these correction factors are interpolated by the parabola giving the maximum correction at 5 points and a zero correction factor at 12 points:

$$\text{corrFactor}(\text{ptSize}) = \frac{\text{maxFact}}{(5\text{pt}-12\text{pt})^2}(\text{ptSize}-12\text{pt})^2;\ 5\text{pt} \le \text{ptSize} \le 12\text{pt}$$

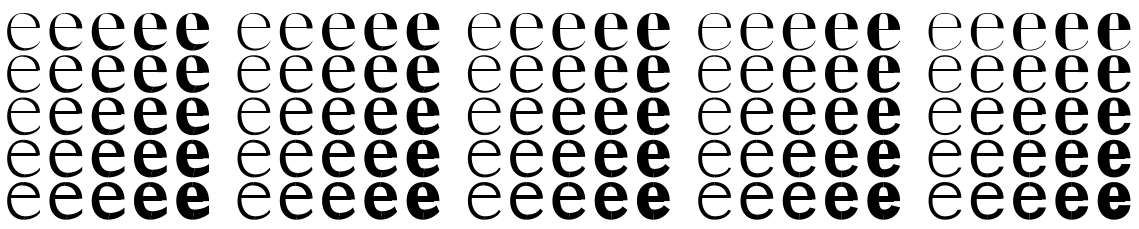
At 5 points, the maximum correction factor for the round letter width and for the stem-to-stem and stem-to-curved-element part spacing is between 106 and 110 percent. The stem, bar, and curved element width have a maximum correction factor of 125 percent. The narrow diagonal bar, narrow horizontal bar, and narrow curved element width have a maximal correction factor of 150 percent. Figure 28a shows characters printed at high resolution, with and without optical scaling. Clearly, optical scaling improves legibility. Figure 28b shows the corresponding optically scaled character shapes at large sizes.

Inspired by the designs of Gerrit Nordzij, a Dutch type designer,[18] we tried to vary the few parameters determining the character e's shape to generate derived designs along three different dimensions (see Figure 29). The first dimension is the character's *weight*, or boldness. All horizontal and vertical width parameters are equally influenced by the boldness parameter.

The second dimension is *contrast*. Increasing the contrast requires reducing the horizontal curve width parameters. The third dimension is *stress obliqueness*. Characters with oblique stress derive from pen-based manuscript writing and were created soon after the invention of moveable metal type—for example, the typeface Jenson, created in 1470. Typeface designs evolved from oblique stress to vertical stress characters. Vertical stress characters became fashionable with the designs of the Bodoni and Didot typefaces at the end of the 18th century. Increasing stress obliqueness requires increasing the obliqueness parameter of the internal ellipse-like arcs. Increased stress obliqueness also requires a slight increase of the horizontal curve-width parameters determining the stroke width at the bottom of the character. Figure 30 (next page) shows the para-



29 Variations of the character e's weight, contrast, and obliqueness.



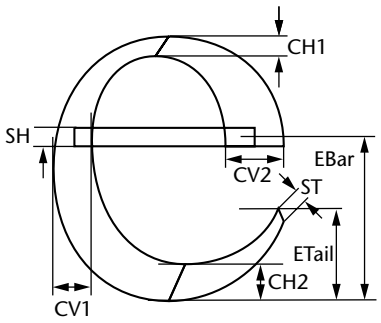
meters determining the character shapes and gives the interpolation rules for computing these parameters from input parameters weight, contrast, and stress obliqueness. Figure 29 shows a volumic representation of the designs obtained by varying these three input parameters.

Generating derived typefaces by parameter modifications is a high-level process, which considerably differs from interpolation between character outline control points.[8] Direct interpolation between outline control points requires the explicit design of character masters at the extremities of the design range. Interpolation between character outlines may lead to inconsistencies in stroke width and to tangential discontinuities at the junctions between curved outline parts. On the other hand, varying the character's global and local parameters according to different design axes such as weight, contrast, and stress obliqueness is a more robust process, since loop parameters β and η ensure the coherence of each of the generated loops or half-loops. At the joints between loops and sweeps and between consecutive sweeps, tangential continuity is ensured by construction.

30 Parameters determining the character e's shape.

Definition of interpolation:

iplt (a, b, percentage) =
a * (1 - percentage) + b * percentage.



1. CH1 = iplt [iplt [normalCH1, maxCH1, boldness], minCH1, contrast]
2. CH2 = iplt [iplt [iplt [normalCH2, maxCH2, boldness], minCH2, contrast], iplt [normalCH2, maxCH2, boldness], obliqueStress * 0.6]
3. CV1 = iplt [minCV1, maxCV1, boldness]
4. CV2 = iplt [minCV2, maxCV2, boldness]
5. SH = iplt [iplt [normalSH, maxSH, boldness], minSH, contrast]
6. ST = iplt [iplt [minCV2, maxST, boldness], minST, contrast]
7. EBar = iplt [maxEBar, minEBar, boldness]
8. ETail = iplt [iplt [minETail, maxETail, boldness], minETail, contrast]
9. ηext = minh = 0; hint = iplt [minh, maxh, obliqueStress]
10. βext = normalb; bint = iplt [iplt [normalb, maxb, (contrast + boldness) / 2], normalb, obliqueStress]

E xplanations:
1: Boldness increases CH1 value, contrast decreases CH1 value.
2: Boldness increases CH2 value, contrast decreases CH2 value, stress obliqueness increases partly CH2 value.
3, 4: Boldness increases CV1 and CV2 values.
5, 6: Boldness increases, respectively contrast decreases SH and ST values.
7: Boldness decreases EBar values.
8: Boldness increases, respectively contrast decreases ETail value.
9: Stress obliqueness increases obliqueness h of interior loops.
10: Contrast and boldness increase, respectively obliqueness reduces squareness b.

Conclusions

Using geometric properties as a means of enforcing coherence across different characters of a typeface has been a dream since the Renaissance. A fundamental question raised by Donald Knuth and Douglas Hofstaeder[19] is whether a significant part of the design space can be explored by turning a few knobs.

A first requirement along that path is to reduce the number of parameters governing the shapes of typographic characters. We've tried to achieve this by defining loops and half-loops with the width, squareness, and obliqueness parameters and by defining categories of terminal elements, each described with only a few parameters. Furthermore, we've developed a specific geometric construction that provides arches that have the shape required by traditional typefaces.

Our approach is validated by the fact that we've been able to recreate existing traditional Latin fonts such as Times Roman, Helvetica, and Bodoni (see http://diwww.epfl.ch/w3lsp/research/typography/compofont/). The experiments we've made show that we can generate interesting derived fonts from existing component-based font descriptions. However, further exploration in collaboration with type designers is required to verify the aesthetic quality of the generated derived font designs.

Fonts made of shape components are flexible parameterizable designs, which we can easily adapt to various display and printing conditions (such as condensed font when lacking display space, high-quality optical scaling, adaptation of fonts to existing character metrics). Beside applications related to typeface design, fonts based on parameterizable components may be used in portable devices where memory is scarce. A single parameterizable design and its variations may allow enough flexibility for providing both high-quality antialiased fonts at low resolution and typefaces for high-resolution printing. ■

Acknowledgment

We thank André Gürtler, well-known type designer and professor of typography at the Basel School of Design. Thanks to his criticism and encouraging remarks, we were able to significantly improve the quality of the characters we resynthesized using parameterizable components.

References

1. J. Tschichold, *Treasurey of Alphabets and Lettering*, W.W. Norton & Company, New York, 1995.
2. R. Southall, "Character Description Techniques in Type Manufacture," *Raster Imaging and Digital Typography II*, R. Morris and J. André, eds., Cambridge Univ. Press, Cambridge, Mass., 1991, pp. 16-27.
3. D. Adams, "abcdefg, A Better Constraint Driven Environment for Font Generation," *Raster Imaging and Digital Typography*, J. André and R.D. Hersch, eds., Cambridge Univ. Press, Cambridge, Mass., 1989, pp. 54-70.
4. P. Karow, *Font Technology, Description, and Tools*, Springer-Verlag, New York, 1994.
5. J. André and I. Vatton, "Dynamic Optical Scaling and Variable-Sized Characters," *Electronic Publishing- Origination, Dissemination, and Desig*n, vol. 7, no. 4, Dec. 1994, pp. 231-250.
6. B. Zalik, "Font Design with Incompletely Constrained Font Features," *Proc. Third Pacific Conf. Computer Graphics and Applications* (Pacific Graphics 95), S.Y. Shin and T.L. Kunii, eds., World Scientific, Singapore, 1995, pp. 512-526
7. A. Shamir and A. Rappaport, "Feature-Based Design of Fonts Using Constraints," *Electronic Publishing, Artistic Imaging, and Digital Topography*, R.D. Hersch, J. André, and H. Brown, eds., LNCS 1375, Springer-Verlag, New York, 1998, pp. 93-108.
8. J. Seybold, "Adobe's MultiMasters Technology: Breakthrough in Type Aesthetics," *Seybold Report on Desktop Publishing*, vol. 7, no. 5, 1991, pp. 3-7.
9. G. Rockledge and C. Perfect, *Rockledge's International Type Finder: The Essential Handbook of Typeface Recognition and Selection*, Rizzoli Int'l Publications, New York, 1991.
10. C. Hu, *Synthesis of Parameterizable Fonts by Shape Components*, EPFL thesis no 1905, 1998, http://diwww.epfl.ch/

w3lsp/publications.
11. D. Knuth, *The Metafont Book*, Addison-Wesley, Reading, Mass., 1986.
12. U. Schneider, "An Object-Oriented Model for the Hierarchical Composition of Letterforms in Computer-Aided Typeface Design," *Electronic Publishing, Artistic Imaging and Digital Typography*, R.D. Hersch, J. André, and H. Brown, eds., LNCS 1375, Springer-Verlag, New York, 1998, pp. 109-125.
13. D. Knuth, *Computer Modern Typefaces* (Volume E of Computers and Typesetting), Addison-Wesley, Reading, Mass., 1986.
14. B. Bauermeister, *A Manual of Comparative Typography, The PANOSE System*, Van Nostrand Reinhold Company, New York, 1987
15. M. Jamra, "Some Elements of Proportion and Optical Image Support in a Typeface," *Visual & Technical Aspects of Type*, R.D. Hersch, ed., Cambridge Univ. Press, Cambridge, Mass., pp. 47-55.
16. B.R. Vatti, "A Generic Solution to Polygon Clipping," *Comm. ACM*, vol. 35, no. 7, pp. 56-63.
17. R.A. Morris, R.D. Hersch, and A. Coimbra, "Legibility of Condensed Perceptually-Tuned Grayscale Fonts," *Electronic Publishing, Artistic Imaging and Digital Typography*, R.D. Hersch, J. André, and H. Brown, eds., LNCS 1375, Springer-Verlag, New York, 1998, pp. 281-291.
18. G. Nordzij, "The Shape of the Stroke," *Raster Imaging and Digital Typography II*, R. Morris and J. André, eds., Cambridge Univ. Press, Cambridge, Mass., 1991, pp. 34-42.
19. D. Hofstaedter, "Metafont, Metamathematics, and Metaphysics: Comments on Donald Knuth's Article 'The Concept of a Meta-Font,'" *Metamagical Themas*, Penguin Books, London, 1985, pp. 260-296.

Changyuan Hu *develops 3D Web authoring software in Montreal, Canada. His research interests are focused on computer graphics and digital typography. He graduated from the Department of Computer Science, Nanjing University and received his PhD from Ecole Polytechnique Fédérale de Lausanne.*

Roger D. Hersch *is a professor of computer science and the head of the Peripheral Systems Laboratory at the Ecole Polytechnique Fédérale de Lausanne. He was the conference chairman of the International Conference on Raster Imaging and Digital Typography 1998. He also directs the Visible Human Web Server project, which offers advanced visualization services for exploring the Visible Human Male and Female datasets (see http://visiblehuman.epfl.ch). He received his engineering degree ETH Zurich and PhD from EPFL.*

Readers may contact Hersch at the Computer Science Dept., EPFL, CH-1015 Lausanne, Switzerland, rd. hersch@epfl.ch, http://diwww.epfl.ch/w3lsp.

EDITORIAL CALENDAR 2001

JANUARY/FEBRUARY

Usability Engineering in Software Development

When usability is cost-justified, it can be integrated into the development process; it can even become one of the main drivers of software development.

MARCH/APRIL

Global Software Development

What factors are enabling some multinational and virtual corporations to operate successfully across geographic and cultural distances? Software development is increasingly becoming a multisite, multicultural, globally distributed undertaking.

MAY/JUNE

Organizational Change

Today's organizations must cope with reorganization, process improvement initiatives, mergers and acquisitions, and ever-changing technology. We will look at what organizations are doing and can do to cope.

JULY/AUGUST

Fault Tolerance

We used to think of fault-tolerant systems as ones built from parallel, redundant components. Today, it's much more complicated. Software is fault-tolerant when it can compute an acceptable result even if it receives incorrect data during execution or suffers from incorrect logic.

SEPTEMBER/OCTOBER

Software Organizational Benchmarking

How do you decide what to benchmark and how much detail is necessary? How do you identify the right information sources?

NOVEMBER/DECEMBER

Just Enough...

How little process and technology can your project get away with? This focus explores the ramifications of developing software from a minimalist perspective.

Ubiquitous Computing

This third wave of computing, after the mainframe and the PC eras, will allow technology to recede into the background of our lives.

IEEE Software