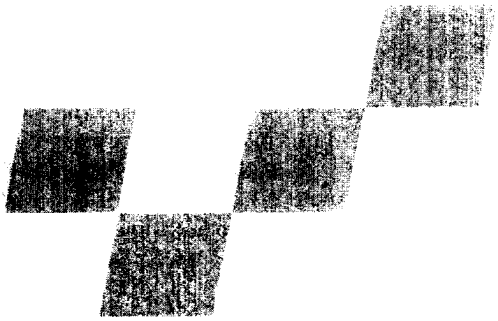


Perceptually Tuned Generation of Grayscale Fonts

Roger D. Hersch, Claude Bétrisey, and Justin Bur
Ecole Polytechnique Fédérale, Lausanne

André Gürtler
Basel School of Design

A new approach to grayscale font generation improves on the fuzzy characters produced by existing methods by incorporating the expertise of type designers into rules for character-outline weight and phase control.

For more than a decade, computer scientists and type designers have tried to use gray levels to compensate for the poor resolution of black and white display fonts.[1-4] The idea was to remove the jaggies induced by the coarse resolution and to improve the rendition of character details. Creating optimal grayscale characters, however, involves a manual pixel-by-pixel design that must follow strict typographic rules.

So far, success in automating the production of grayscale fonts has been limited. The filtering and resampling method used to generate grayscale characters at display resolution produces characters with poor contrast. Display windows made of such characters appear fuzzy, particularly at font sizes smaller than 12 points. Thus, most PC and workstation interfaces support the display of only bilevel (black and white) characters. Of course, bilevel characters suffer from aliasing (jaggies and staircases) at the low resolutions typical of most displays, but the improvements from the use of grayscale characters have not been sufficient to induce most software manufacturers to invest in grayscale font rasterization. Windows 95 is one of the first platforms to attempt to support this function (see the sidebar "Font Smoothing in Windows 95").

Some researchers believe that the quality of the characters can be improved using different filter kernels. However, we show in this article that all linear filtering methods assign different contrast profiles to the same character parts. This leads to fuzzy characters regardless of the filter because it violates one of the design rules that experienced type designers use to ensure high-contrast type: All identical character parts should have an identical look.

Poor contrast does not ultimately impair legibility, according to research in psychophysics,[5] but there is a "reading comfort" level that depends on the degree of contrast, as well as on the spacing between characters.[6] However, there seems to be no consensus on the factors that determine reading comfort.

The method we propose for synthesizing high-contrast grayscale characters relies instead on the set of visual rules type designers have derived from their many years of manual design. We call our grayscaling method *perceptually tuned* font generation, because it is based on human perception—that of both the designer and the reader.

Our method has three important features:

- It translates expert designer guidelines into a set of grid-fitting rules that govern the grayscaling software. Grid fitting consists of fitting the master character outline to a pixel grid in a way that maximizes the sharpness, uniformity, and high contrast of characters. The result is fonts that are very close approximations to what an experienced type designer would produce manually.
- It promotes accurate weight and phase control. Grid fitting requires that we be able to control both the weight of each character part and the phase of the character outlines with respect to the underlying grid. This ensures that the character elements have a minimum width, even at a small size, and that identical outline parts, such as bars, stems, or serifs, are rendered by an identical set of pixels in all the grayscale characters produced.
- It automatically computes the most visually even spacing of successive grayscale characters using a model of the way the human vision system perceives space between characters. This is an important part of our method because the most beautiful set of single characters cannot improve the quality of text if those characters are not spaced correctly. At display resolutions, our spacing method provides a more accurate spacing of successive characters than other automatic kerning algorithms that are independent of the display resolution.

0272-17-16/95/$4.00 © 1995 IEEE

Font smoothing in Windows 95

The new TrueType rasterizer included in Windows 95 supports grayscale font rasterization. It is actually an intermediate step between bilevel and grayscale font generation because the characters produced are partly black-white and partly grayscale.

The rasterizer first adapts the outline font to the grid in the same way as if a bilevel bitmap would be produced (bilevel hinting): It places the edges of vertical and horizontal bars and the horizontal and vertical tangential points of curved outline parts midway between pixels on pixel boundaries. The subsequent grayscaling step leaves vertical and horizontal bars as black and smoothes out the curved character parts. The resulting character is the smoothed version of the corresponding hinted bilevel character.

The designer can specify in which size ranges to apply grayscaling only (box-filtered grayscale), bilevel hinting, or combinations of hinting and grayscaling.

In its World Wide Web documentation, Microsoft advocates grayscaling only at very small sizes, without any hinting; hinted bilevel characters at intermediate sizes; and combined grayscaling and hinting at large sizes.

Font smoothing in Windows 95 lets designers produce smoothed fonts without modifying the width of the characters, thus producing the same text line length for bilevel and grayscale fonts. Nevertheless, since the vertical and horizontal bars are represented as solid black, the potential of perceptually tuned grayscaling remains largely unexploited at small sizes.

Furthermore, since characters, words, and lines must have the same width regardless of display size and resolution, optimal spacing techniques are not supported. To obtain high-quality grayscale fonts at small sizes, the designer must code the grayscale-specific hints as a set of alternative TrueType instructions that differ from the standard TrueType instructions for bilevel hinting.

For more information on font smoothing support in Windows 95, see http://www.microsoft.com/DEVONLY/PRODINFO/MSDNPROD/LIBRARY/SPECS/win95TT.htm or http://www.microsoft.com/windows/thirdparty/plus/lookgr.htm.

We have produced fonts using this method that have much improved contrast over those produced with any filtering and resampling method. Without optimization, we have generated and spaced 12-point grayscale characters on a Sun Sparc-2 workstation at a speed of 12 characters per second.

Characters generated by filtering

To support our conclusion that different filter kernels do not produce characters with significantly different degrees of contrast, we compared the character string "Hamburgefon" as generated by three common filters—cubic spline, Hamming, and box-filter[7]—and compared these with a manually generated design. Figure 1 shows the results. Hamburgefon is a character string that is typically used to test type designs because it contains instances of all the character groups (capital letters, round letters, characters with ascenders, and so on) and has successive characters that let designers check for correct spacing.

Despite minor differences, all grayscale characters obtained by filtering show some fuzziness because similar character parts do not have the same intensities of black, or *contrast profile*. As Figure 2 shows, the contrast profile in the stems of "m,"

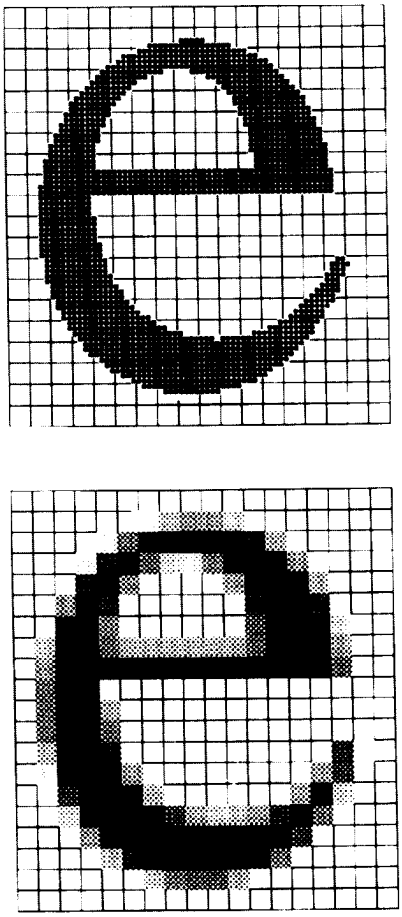
1 Filtered grayscale characters obtained by convolution with (a) a spline approximation of the sinc function, (b) a Hamming function, and (c) a rectangular function (box filter). These compare with (d) characters created manually pixel by pixel.

2 How grayscale characters differ according to the filter used to generate the characters: (a) Characters produced by a Hamming filter, and (b) characters produced by a box filter. This shows that contrast is due mainly to phase control (how the master character is situated on the grid), not on the filtering technique per se.

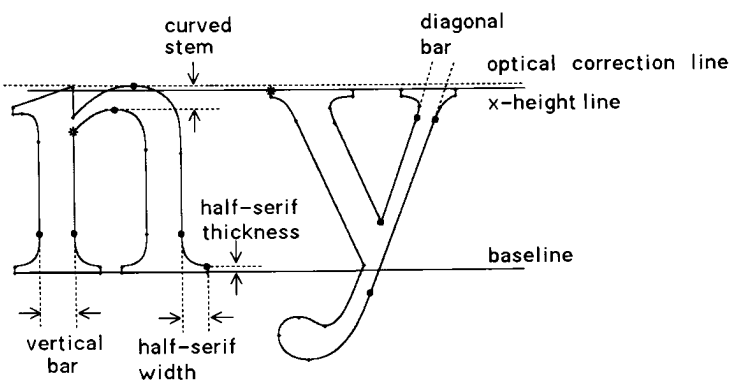
"n," "u" and "r" differs from one stem to another with both the Hamming filter and the box-filter technique.

Also, as Figures 1 and 2 show, the different filters produce pretty much the same degree of contrast. For this reason, we chose the box filter as representative of what filtering methods can generate.

We generated box-filtered characters using a two-step averaging technique. First we generated bilevel master characters from their outline descriptions in bitmaps with k times the target grayscale pixmap resolution. The resulting master-character bitmap represents target-resolution pixels by squares of size $k \times k$ pixels. We then used these bitmaps to compute the percentage each target-resolution pixel covers. This percentage indicates the gray intensity level. Figure 3 shows the binary master character "e" and the resulting grayscale character.



3 A grayscale character produced by box-filtering a binary master character using an averaging technique.



4 Location of some important character features from which feature descriptions, or hints, are derived for the "n" and "y" characters.

Role of phase control

To understand our method and how we used phase control to obtain sharp, high-contrast characters, it is helpful to understand exactly how character-outline phase control works. This is most easily explained in terms of bilevel characters, but the same principles hold for grayscale characters.

Fitting the master-character outline to a pixel grid requires that we perfectly control the phase of each character element's outline with respect to the underlying grid. This tight control ensures the subsequent coherent rasterization of the characters. We obtain this control by slightly varying the position of the borders of character elements, such as vertical bars and curved stems, to ensure that identical elements will have the same phase with respect to the grid and therefore be represented by the same set of pixels on the resulting bitmap.[8]

Phase-control mechanisms require two things: a character outline and a description of the character's most important features. Feature descriptions[9] may be generated automatically.[10]

Character outline

To rasterize character outlines at middle and low resolutions, you must precisely control the phase of the characters' reference lines and structural elements such as vertical and horizontal bars, round stems, serifs and diagonals.[8]

In bilevel phase control of vertical bars, for example, the phase determines the width of the bar. To be able to respect original relationships between different horizontal and vertical bars, the bars must be centered on the pixel grid. This operation also minimizes the difference between the width of the continuous bar and the width of the resulting discrete bar, which has an integer number of pixels.[8]

Feature descriptions

When characters are rendered, feature descriptions, or *hints,* in the form of grid-fitting rules are generated automatically to help adapt the character's outline to the rasterization grid.[10] This in turn helps preserve features like symmetry, thickness, and uniform appearance in the rasterized character. Character-outline parts like bar borders are displaced to conform with specified phase constraints between the borders of the outline part and the underlying grid.

At their most basic level, hints describe character reference lines, optical correction lines, the horizontal position of vertical bars and curved stems, and the vertical position of horizontal bars and curved stems.

Advanced hints describe extreme

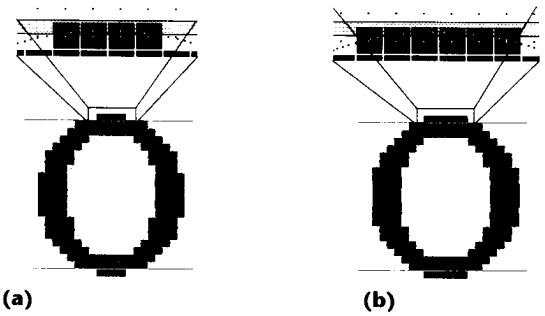
points of half-serifs and the guiding points of diagonal bars. Figure 4 shows some common locations of hints for the "n" and "y" characters.

Displacement

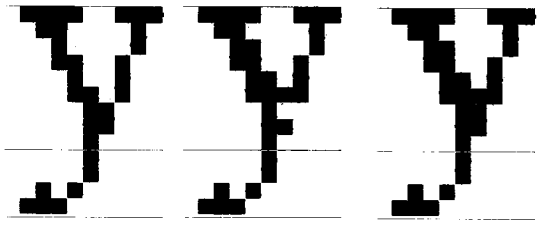
In bilevel rasterization, the software controls the bowls and curved character parts by keeping the phase of the vertical or horizontal extremity of arcs within a given range.[8] The phase range influences the flatness of the produced arc, as Figure 5 shows.

Similarly, controlling the phase of diagonal bars ensures a constant bar width. This constant width can be guaranteed if the horizontal distance between the two borders equals an integer pixel width because an integer bar width ensures that both border lines will have the same phase.

The rasterizing software interprets the grid-fitting rules expressed through hints by shrinking or enlarging the diagonal bar to obtain the required integer width. In Figure 6, for example, the software applies a translation to a single border or to both borders of the bar.



5 The flatness of an arc in the "o" character is controlled by altering the phase range—the range between two pixel centers where the continuous contour is allowed to pass. The white dots in the center of the squares are the centers of the pixels; the other dots represent the trajectory of the phase-controlled outline. In (a), the phase range (distances from the pixel center) is 1/16 to 9/16. In (b), it is 3/16 to 11/16.

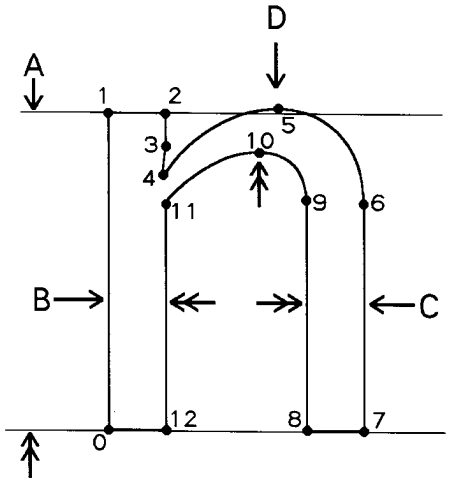


6 Phase control of diagonal bars in the "y" character: (a) the original diagonals, (b) phase-controlled diagonals, and (c) phase-controlled diagonals after reintersection.

Figure 7 shows hints for fitting the "n" character onto a grid. Each hint contains a *specification* part, which defines its type, and an *application* part, which defines the scope of the character-outline part to be modified. Using one or two pairs of points, the specification part defines the current location of a character element (bar, curved stem, serif, and so on) that must be fitted according to given phase constraints.

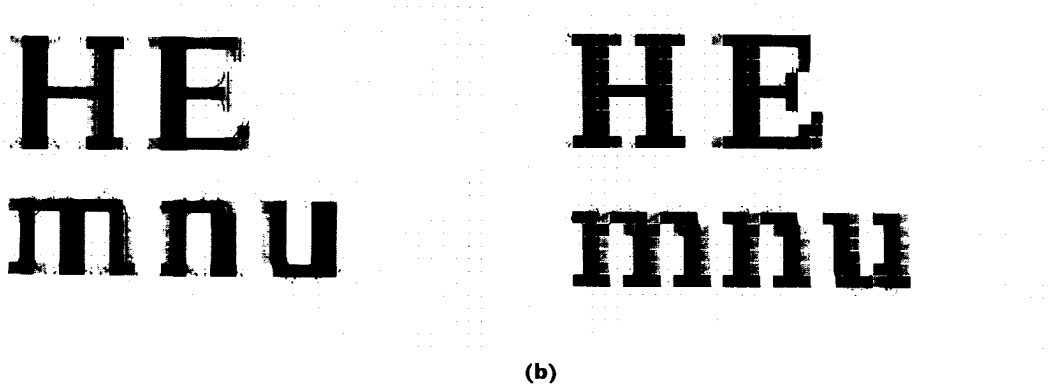
As the figure shows, displacements may be applied elastically to given outline parts, and a given part may have a fixed point and a maximum displacement point. Each control point is displaced in proportion to its relative position between these two points.

The rasterizing software evaluates a hint specifica-

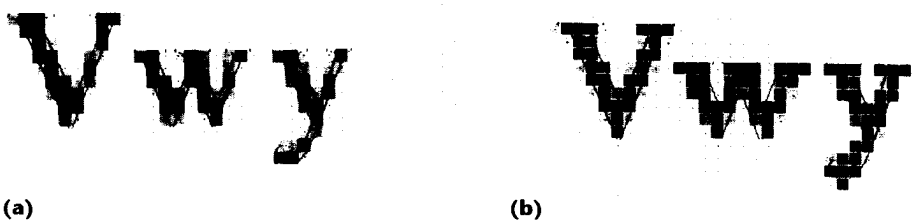


A: hint specification: vertical phase control of reference lines
hint application: complete character
B: hint specification: horizontal phase control of vertical stem
stem width given by Pt0, Pt12
hint application: first displacement
fixed displacement: Pt0 to Pt2
hint application: second displacement
fixed displacement: Pt2 to Pt0
C: hint specification: horizontal phase control of vertical stem
stem width given by Pt8, Pt7
hint application: first displacement
fixed displacement: Pt8 to Pt9
proportional displacement: Pt9 to Pt11
fixpoint: Pt11
maximum displacement point: Pt9
hint application: second displacement
fixed displacement: Pt6 to Pt8
proportional displacement: Pt4 to Pt6
fixpoint: Pt4
displacement point: Pt6
D: hint specification: vertical phase control of shoulder
shoulder thickness given by Pt10, Pt5
hint application: first displacement
proportional displacement: Pt9 to Pt11
fixpoint: Pt9, Pt11
maximum displacement point: Pt10
hint application: second displacement
proportional displacement: Pt4 to Pt6
fixpoint: Pt4, Pt6
maximum displacement point: Pt5

7 Support points for the specification of grid constraints. Each hint (designated by A through D) contains a *specification* part, which defines its type, and an *application* part, which defines the scope of the character-outline part to be modified.



8 Examples of characters with vertical bars: (a) characters obtained with a box filter, and (b) manually tuned characters.



9 Examples of characters with diagonal bars: (a) Characters obtained with a box filter have thin bars, represented by light gray values. (b) Manually tuned characters have a sufficient number of black pixels for the eye to recognize the diagonal bar.

tion and generally produces two displacements. Each displacement is applied to one border of the element under consideration. The constraint application part specifies the character parts that are to receive the computed displacements by giving their starting and ending outline-support points.

Grayscale-specific concerns

Knowledge about character shape, incorporated in the form of hints into the character-outline description, is sufficient for generating perceptually tuned grayscale fonts. The difference is that to produce grayscale characters, there must be weight-control as well as phase-control rules for tuning the contrast in bars, curved stems, and stroke ends (terminals). Therefore, the grayscale character rasterizer must have its own grayscale-specific hint interpretation. In our method, the grayscaling software computes displacements for each character part to ensure a predefined contrast distribution in bars, curved stems, serifs, and terminals according to the design rules described in the next section.

Grayscale design guidelines

Type designers with decades of experience can adequately express requirements about the appearance of grayscale characters. Tools like Letraset's FontStudio allow them to experiment with manually designed grayscale characters and formulate visual design rules. These rules provided us with a sound framework for generating perceptually tuned grayscale characters.

The characters in Figures 8 through 12 were designed by experienced type designers using a pixel editor. The designers began with a grayscale character produced by averaging a binary master character four times the size of the target grayscale character. The characters illustrate some of the more important type-design guidelines.

Figure 8 illustrates the guideline about vertical bars:

> Vertical bars of characters with the same width should have the same contrast distribution, with the left edge having the sharpest possible contrast. Depending on the width of the stroke, a stronger or lighter gray level is acceptable for the right edge.

At screen resolution, gray cannot always replace the lack of resolution. Especially for thin strokes, two neighboring gray pixels don't replace a single dark pixel. All similar bars should look identical to the eye. That is, they should have the same contrast profile and therefore the same sampling phase, as in Figure 8b.

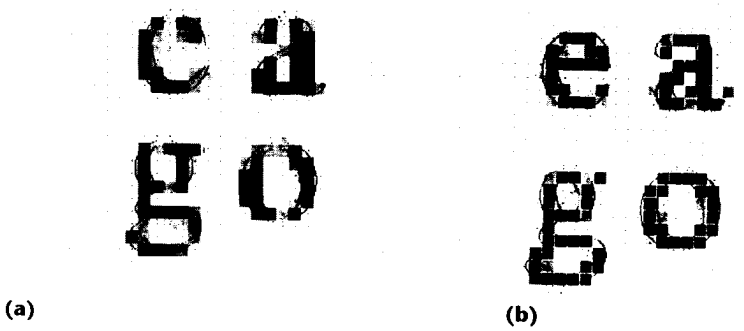
Figure 9 illustrates a guideline about diagonal bars:

> Diagonal bars should behave like vertical bars of the same weight with sufficient black pixels to transmit the visual structure to the eye.[11]

As Figure 9a shows, thin diagonal bars, represented by light gray values tend to disappear. They must be reinforced by making them darker, as in Figure 9b.

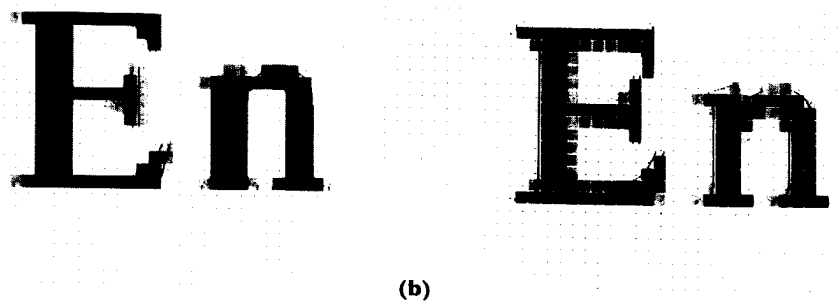
Figure 10 illustrates a guideline for rounded characters like "e," "a," "g," and "o."

> Characters with a thin round part should be part-



(a) (b)

10 Examples of characters with many round parts: (a) Characters obtained with a box filter tend to disappear. (b) Manually tuned characters have blackened character parts that define the overall round shape more accurately.



(a) (b)

11 Emphasizing the structure of vertical and horizontal serifs: (a) Serifs obtained with a box filter seem to fade away. (b) Manually tuned serifs contain enough black pixels to be captured by the eye.

ly blackened to strengthen the overall appearance of the shape.

Thin character parts tend to disappear if not sufficiently strong, as Figure 10a shows. Similar curved stems should also have similar contrast profiles. Profiles of curved stems may start at the exterior side as light gray, then become black and eventually end up as black or gray, as in Figure 10b. Designers can render optical correction nicely by having gray pixels overlap their respective reference lines (baseline, *x*-height line, cap line).

Figure 11 illustrates the guideline concerning serifs:

> Small serifs require just enough emphasis to be captured by the human eye. Large vertical serifs like those found in "E" and "F" should be sufficiently structured with dark spots so as to avoid falling apart.

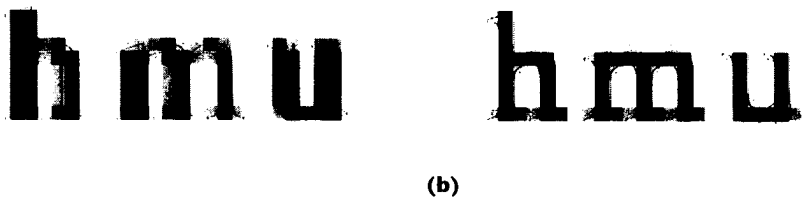
Rules for modifying the outline character

We translated the grayscale design rules just described into an adequate set of outline-modification mechanisms for modifying the character outline using phase and weight control. Phase control allows us to define character phase with respect to the pixel grid, which in turn gives us a way to control the contrast of bars, curved stems, serifs, and terminals.

We rendered the grayscale characters by first applying phase and weight control with respect to the grayscale grid. We then computed the coverage percentage of each gray pixel by generating from the phase- and weight-controlled outline a binary master character four times larger and higher than the target grayscale character. We obtained the intensity values of gray pixels according to coverage percentages in corresponding 4×4 cells of the binary master character (similar to the method in Figure 3).

We emphasized thin bars by widening them, computing the original bar width from the bar's hint information. If the bar width was below a certain value, we enlarged the bar so that its final size came close to a width of one pixel. For example, if the bar was 1.0 and the value chosen was 1.2, the bar width would be enlarged to 1.1. That is, we took the mean value between a reasonably wide bar (such as 1.2 pixels) and the actual bar width; which became the new bar width. We then placed the newly computed bar width on the grid in such a way that one of its bar boundaries, preferably the left one if the bar is vertical, was on pixel boundaries. This ensured that one side of the produced bar would have a high contrast.

We obtained the newly computed and placed bar from the unmodified original bar in two steps. We first computed one displacement and applied it to the original first bar boundary. We then computed and applied a second displacement to the original second bar boundary.



12 Controlling the contrast of bars and emphasizing thin strokes: (a) Characters without phase and weight control (grid fitting) and (b) characters with phase and weight control.

(a) (b)

13 Contrast of curved character parts: (a) Characters without phase control and (b) characters with phase control.

(a) (b)

14 Examples of grid-fitted round characters (a) without weight and phase control and (b) with weight and phase control.

(a) (b)

As Figure 12 shows, the combined operation of bar widening and contrast control produces well-structured characters with repetitive bars.

Controlling the contrast of curved stems requires that we give the horizontal extrema of external curves the same gray value as the gray pixels on the vertical extrema of curves. To do this, we placed the curved stem on the grid in such a way that the phase of the extrema of its external contour was the same as the phase of the vertical contour extrema at the *optical correction point* of "O" for capital letters and "o" for lowercase letters. The phase at this point is the difference between the character's optical correction line and the cap-line, respectively the *x*-height line. The phase location at the optical correction point determines, for a given character size and resolution, the contrast intensity profile to be found at all other horizontal and vertical curved stem extremes. Figure 13 shows this technique applied to the "D," "o," and "c" characters.

Round thin character parts such as those in Figure 13a give the impression that they are disappearing. The corresponding character looks as if it will fall apart. The appearance of such character parts can be improved through weight control. Weight control is based on a procedure that checks for a minimum distance between exterior and interior arc boundaries at horizontal and vertical maxima as well as at junctions between arcs and bars. This allows us to boost the frail character part.

We applied weight control to curved stems by first subtracting the optical correction value from the original stem width. We controlled the thinned stem width in the same way we controlled the width of the horizontal and vertical bars: If the width was below a certain value, we enlarged it so that its final width was close to one pixel. We again used the mean value between the widths of a reasonably wide curved thinned stem and the actual curved thinned stem and made this value the new thinned curve's stem width.

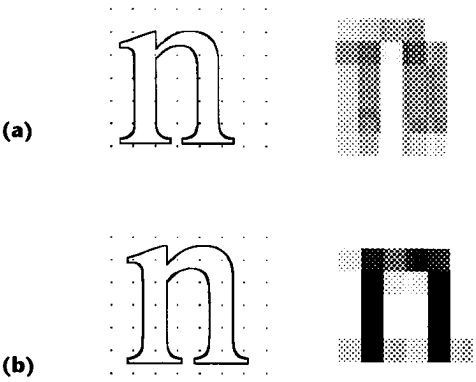
We then placed the resulting thinned curved stem on the grid as if it were a vertical bar or a mirrored vertical

bar, depending on its orientation. We moved its exterior contour by the computed optical correction value. We obtained the new curved stem from the original unmodified curved stem by applying one displacement to the external curved stem contour boundary and another to the internal curved stem boundary. Figure 14b illustrates the resulting grayscale characters.

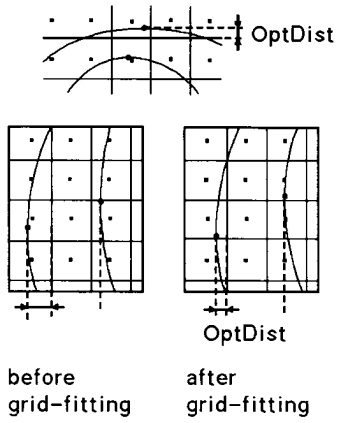
Figure 15a shows a single character placed at a random grid location. Figure 15b shows the same character after we applied weight and phase control according to the hint specifications and application rules described in Figure 7.

Figure 16 shows the details of grid fitting for the curved stem of "o." We used the optical correction value OptDist to place the curved stem in such a way that its exterior arc boundary had the same phase on the grid as the exterior horizontal arc at the optical correction position.

Figure 17 shows how we altered thin stroke ends to generate strong contrast. We made the ends darker using the same operations we applied to vertical and horizontal bars. We then widened the ends and placed their exterior parts at a pixel boundary.



15 Character "n" (a) before weight and phase control and (b) after.



16 How phase control is applied to round stems, in this case, the curved stem of the character "o." Before grid fitting, the horizontal extremum of the external arc has any phase with respect to the grid. After grid fitting, it has a phase that corresponds to the optical correction distance.

Character spacing

The character spacing provided by traditional filtering and resampling techniques is costly in terms of both processing power and memory. The reason is that subpixel spacing—spacing successive characters apart by fractions of a pixel width—is provided only if several versions of the same character are generated at different phase intervals.[2,4]

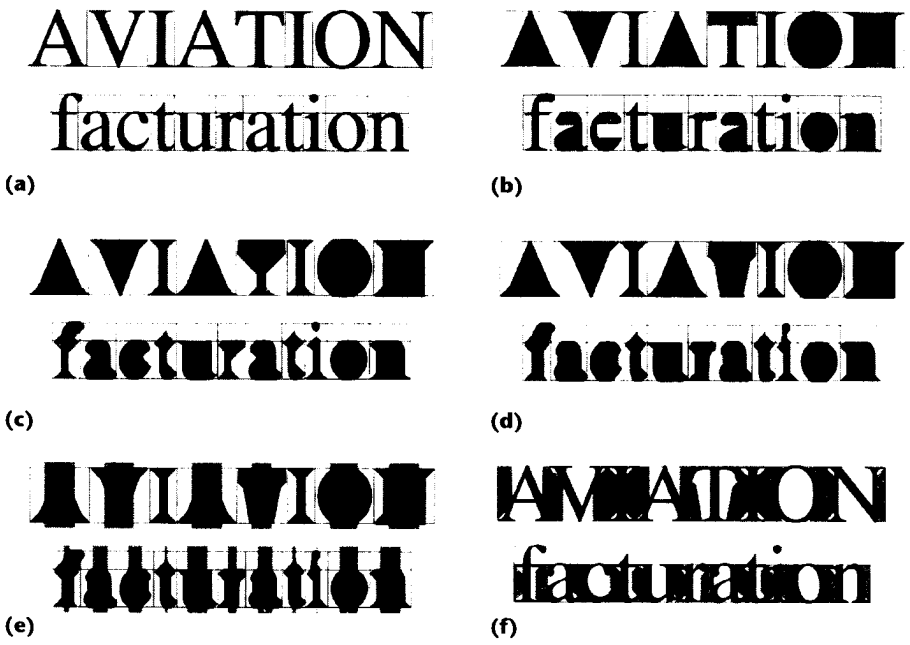
Our method completely defines the phase relationship between the weight- and phase-controlled contour and the grid and thus provides no opportunity for subpixel spacing. Instead, to provide visually pleasant intercharacter spacing, we developed an algorithm for automatically computing the best possible spacing value between two given characters. This optical character-spacing method is based on a model that lets us compute the space the human vision system is most likely to perceive between two characters from the geometry of their grid-adapted outline shapes.

Assumptions

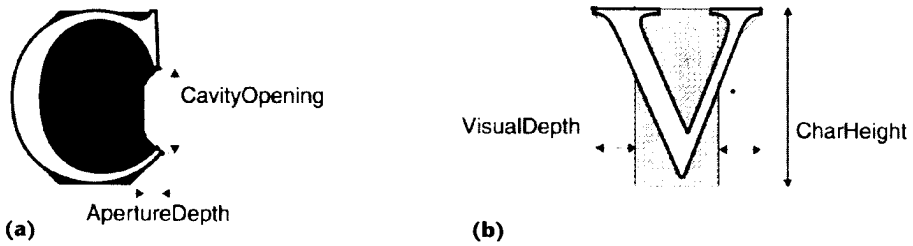
The model is based on the assumption that when we read a text line, our vision system recognizes each successive character by activating a spatial-frequency channel in a band one octave wide, ranging from one cycle to two cycles per character.[5] The perceived intercharacter space is therefore filtered and the shape details of the two character parts are smoothed out. Because the filtering process tends to close an open cavity, such as that in "c," we use the following criteria to model the perceptual space between two characters:

- The inner shape of a character does not have much influence on the perceived intercharacter space.
- Only the parts of cavities that can be seen from the exterior of the character influence the visual intercharacter space.
- The cavity space and exterior contour parts that are close to the next character's contour have the largest effect on the perceived intercharacter space.

17 Effect of phase control applied to stroke ends, or terminals: (a) Without weight and phase control, and (b) with it.



18 Area equivalent to the perceived intercharacter space obtained through a series of transformations applied to (a) the original letter shapes. The first transformation is (b) eliminating white space not directly connected to the intercharacter space. The second is (c) to smooth out contour parts. The third is (d) to limit the interior depth of cavities contributing to the intercharacter space. The fourth is (e) globally limit the interior space that contributes to the intercharacter space. The result is (f) an area equal to the perceived intercharacter space.



19 Depth parameters used to tune the perceived-spacing model. In (a), ApertureDepth = CavityOpening * RelativeDepth. In (b), VisualDepth = CharHeight * VisualDepthFactor.

Transformations

From these criteria, we derived a set of transformations that convert the geometric intercharacter space into its visually perceived equivalent. Figure 18a shows the original character shapes to which we applied the following transformations:

1. Eliminate white space interior to the character shape or hidden behind vertical character elements, as in Figure 18b.
2. Limit interior white concavities to a surface bounded by ±45-degree straight line segments, as in Figure 18c.
3. Limit the depth of the interior white space of cavities to a value proportional to the size of the cavity, as in Figure 18d.
4. Limit the depth of the contributing white space to a value given as a proportion of the character height. That is, for capitals the cap-height and for lowercase characters the *x*-height, as in Figure 18e.

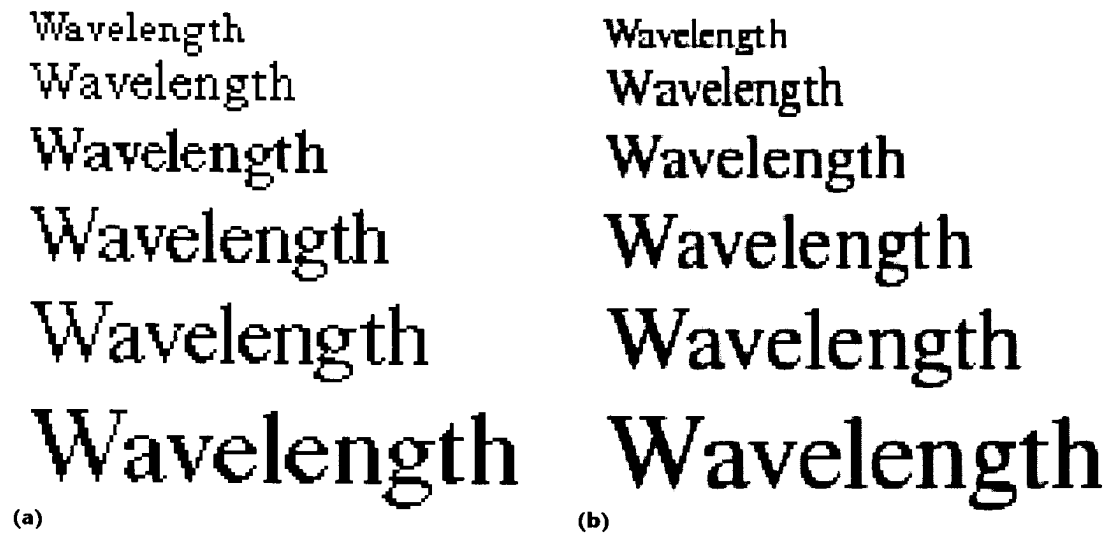
Figure 18f shows the area we obtained after applying the transformations. The area is equivalent to the perceived intercharacter space. To produce uniformly spaced characters, we equated the visual space computed by successive transformations with an ideal visual space. The ideal space is the visual space between two characters that a type designer has optimally spaced. For Latin characters, the ideal visual space is the value taken from the visually perceived intercharacter space of the "nn" character pair for lowercase and the "HH" character pair for capitals.

The parameters *RelativeDepth* and *VisualDepthFactor* limit the depth of the geometric white space that contributes to the visually perceived intercharacter space, as Figure 19 shows. We tuned these depth parameters to ensure that the computed spacing values of typical character pairs ("cu," "ov," "vu") produced the correctly spaced pairs of characters.

Figure 20 shows that these transformations produce correct optical spacing values for both bilevel and grayscale characters in the character string Wavelength set in the Times typeface.

Character pairs

Characters that have the same character elements on each side of the space between them (vertical bars, curved stems or one vertical bar and one curved stem) must be spaced in the same way. For such character pairs, we extracted the spacing value for all combinations of vertical bars and curved stems from the original character master and tabulated them as percentages of their body height. Figure 21 shows these pairs for the Times boldface font. The actual tabulated values depend

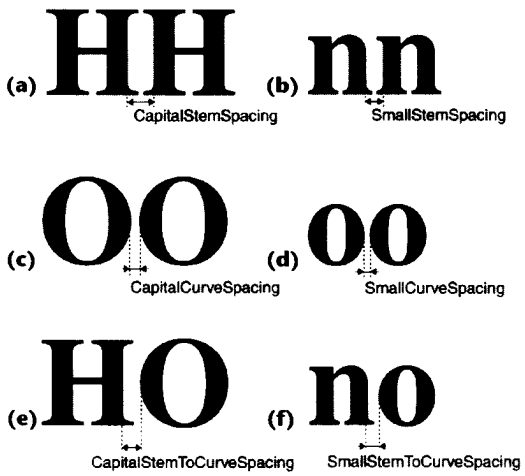


20 Automatic optical spacing of (a) Times bilevel characters and (b) Times grayscale characters.

on the font style, but there are always six tabulated values per font.

Computing the spacing

As Figure 22 shows, each weight- and phase-controlled character outline that has passed through the transformations possesses a left and right border. The space between the left reference line and the right border is the *complemented right visual space,* or *CRVS.* The space between the left reference line and the left border is the *left visual space,* or *LVS.* The *ideal optical space,* or *IOS,* is the value taken from the visual space of two characters optimally spaced by design, such as the visually perceived intercharacter space of the "nn" pair for lowercase and the "HH" pair for capitals.



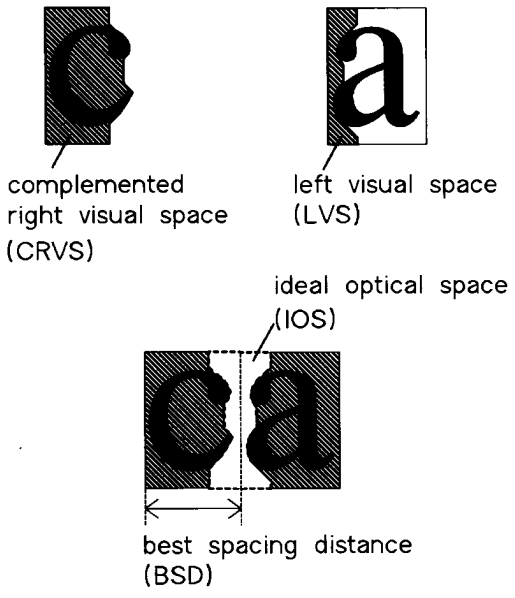
21 Character pairs for which we tabulated distance values: (a) space between capital bars, (b) space between lowercase bars, (c) space for capital curved stems, (d) space for lowercase curved stems, (e) space between capital bars and curved stems, and (f) space between lowercase bars and curved stems.

When spacing character pairs that have a capital and a lowercase character, the mean value of the capital's ideal optical space and the lowercase's ideal optical space is the ideal optical space. We computed the *best spacing distance,* or *BSD,* between the origin of the first character and the origin of the second character according to

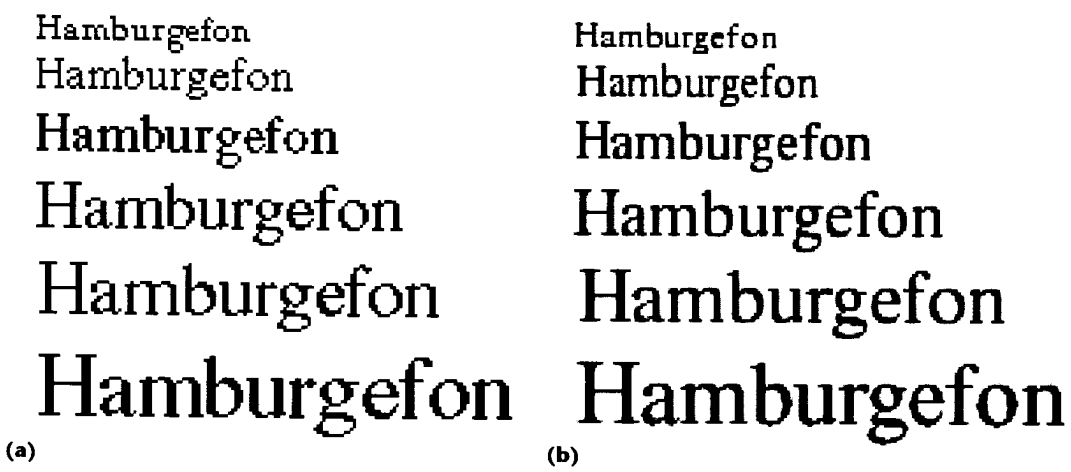
$$\mathrm{BSD}(1\mathrm{stChar},2\mathrm{ndChar}) = \frac{\mathrm{CRVS}(1\mathrm{stChar})}{\mathrm{CharHeight}} + \frac{\mathrm{IOS}}{\mathrm{CharHeight}} - \frac{\mathrm{LVS}(2\mathrm{ndChar})}{\mathrm{CharHeight}}$$

where CharHeight is respectively the cap height for capitals, the *x*-height for lowercase characters, or the mean value between the cap-height and *x*-height for the ideal optical space associated with a mixed pair of capital and lowercase characters.

We computed the best spacing distance in a similar way for character pairs whose respective right and left borders are either vertical bars or curved stems. In such cases, we define the borders of the characters by a vertical line either through the vertical bar's edge or through



22 Best spacing distance between the origins of two successive characters.



23 Display of grid-fitted bilevel characters and weight- and phase-controlled grayscale characters: (a) Automatically generated bilevel characters, and (b) weight- and phase-controlled grayscale characters.

the extreme of the curved stem. We replaced the ideal optical space with the corresponding tabulated spacing value, multiplied that by the character's height, and then computed the best spacing distance as before.

If necessary, we can increase the best spacing distance to avoid overlapping parts of the two character shapes. For grayscale characters, we request a minimum spacing value of 0.25 pixels between the character extremes located at similar vertical locations, such as between foot serifs. However, at the target resolution, we must round the so-obtained next character's origin to fit a pixel boundary. To avoid rounding similar character widths both up and down, we used an adaptive rounding process. This process ensures that most best-spacing values are either rounded up or rounded down. To accomplish this, we created a histogram that gives the distribution of the fractional values of the spacing distance for all character pairs, weighted by their occurrence probability. The histogram lets us define a rounding threshold, possibly different from 0.5, ensuring that the best visual distance of most character combinations will be rounded in the same way.

Rounding the best spacing distance in this way gives the integer spacing value in pixels from the character origin of the first character to the character origin of the second character.

Results

Figure 23 shows the results of generating "Hamburgefon" at several sizes using different character-generation techniques. We generated the grayscale characters in Figure 23b using 17 intensity levels selected among 256. We used a γ correction to produce visually equidistant gray-level steps.

We produced the gray-level characters without taking into account the modulation transfer function of the display device, such as the smearing effect produced by the CRT. Our experience shows that on a sharp monochrome display such as the Macintosh Portrait Display monochrome monitor, the characters look very much like the ones in Figure 23.

Lower quality and color displays often have a lower bandwidth in the horizontal direction and tend to filter out high-contrast transitions.[12] Therefore, differences in appearance among the various techniques for generating grayscale characters have a decisive effect only on high-quality high-bandwidth displays, where you can distinguish the intensities of individual pixels. LCDs are especially well-suited for the display of perceptually tuned grayscale characters because each pixel appears as a sharp black, white, or colored square.

As Figure 23b shows, applying weight and phase control produces characters with high-contrast regular bars and with a well-defined structure stressed by dark pixels distributed along the character shape. Because of phase control, instances of character structure elements such as vertical bars, curved stems and serifs located in different character shapes maintain an identical appearance. At an 8-pixel cap height, 10-point type on a 80-dpi display, weight- and phase-controlled grayscale characters still reproduce some of the characteristics of the original font, while remaining legible and well contrasted. As proof, compare the top character line in Figure 23b with the corresponding lines in Figures 1c and 23a. For those interested in seeing an actual display, pages with EPFL's perceptually tuned grayscale characters at different sizes appear on the World Wide Web at the address http://diwww.epfl.ch/w3lsp/pub/grayscale.

Conclusions

A visual comparison between filtered, weight- and phase-controlled, and manually tuned grayscale characters shows that our perceptually tuned weight-and phase-control mechanisms can generate grayscale characters whose appearance is quite close to the highest quality manually tuned characters.

Weight- and phase-controlled grayscaling represents a significant qualitative improvement compared with traditional techniques to automatically generate grayscale characters. Keeping high-contrast borders in vertical and horizontal bars and ensuring a minimal weight of thin parts guarantees visually pleasant characters on high-quality displays. Perceptually tuned grayscaling may find further application in flat-panel displays for laptop computers. By using a grayscale character cache, the perceptually tuned generation of grayscale characters may become a standard feature in personal computers and workstations. We also expect it to greatly enhance the text display capability of low-resolution displays such as LCDs. ■

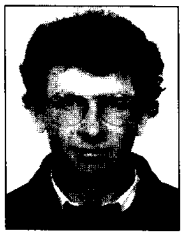
Acknowledgments

We thank Catherine André, who integrated the grayscale generating software into the RalPage PostScript previewer and generated the PostScript image files of most of the figures. Kaspar Mangolt designed most of the manually tuned grayscale characters. Shan HePing ported the grayscale rasterization software onto the Macintosh.

This research was funded by the Swiss Commission for the Promotion of Scientific Research under grant CERS 2077.1. The integration of grayscale characters into a PostScript previewer was funded as part of the Esprit II Spirit project under grant CERS 2130.1.

References

1. F. Crow, "The Use of Grayscale for Improved Raster Display of Vectors and Characters," *Computer Graphics* (Proc. Siggraph), Vol. 12, No. 3, 1978, pp. 1-6.
2. J. Warnock, "The Display of Characters Using Gray Level Sample Arrays," *Computer Graphics* (Proc. Siggraph), Vol. 14, No. 3, 1980, pp. 302-307.
3. J. Kajiya and M. Ullner, "Filtering High-Quality Text for Display on Raster Scan Devices," *Computer Graphics* (Proc. Siggraph), Vol. 15, No. 3, 1981, pp. 7-15.
4. A. Naiman and A. Fourier, "Rectangular Convolution for Fast Filtering of Characters," *Computer Graphics* (Proc. Siggraph), Vol. 21, No. 4, 1987, pp. 233-242.
5. G. Legge, et al., "Psychophysics of Reading — I Normal Vision," *Vision Research*, Vol. 25, No. 2, pp. 239-252.
6. M. Boschman and J. Roufs, "Comparison of Methods for the Evaluation of VDUs: The Effect of Bandwidth," *SID Digest of Tech. Papers*, Vol. 21, pp. 17-20.
7. W. Pratt, *Digital Image Processing*, John Wiley & Sons, New York, 1991.
8. R. Hersch, "Font Rasterization: The State of the Art," in *Visual and Technical Aspects of Type*, R. Hersch, ed., Cambridge University Press, Cambridge, 1993, pp. 78-109.
9. P. Karow, *Font Technology, Methods and Tools*, Springer-Verlag, Berlin, 1994.
10. R. Hersch and C. Bétrisey, "Model-based Matching and Hinting of Fonts," *Computer Graphics* (Proc. Siggraph), Vol. 25, No. 4, 1991, pp. 71-80.
11. A. Ginsburg, "Spatial Filtering and Visual Form Perception," in *Handbook of Perception and Human Performance: Volume 2*, K. Boff, L. Kaufman, and J. Thomas, eds., John Wiley & Sons, New York, 1986, pp. 34.1-34.41.
12. N. Lyons and J. Farell, "Linear Systems Analysis of CRT Displays," *SID Digest of Tech. Papers*, Vol. 20, pp. 220-223.



Roger D. Hersch *is a professor of computer science and head of the Peripheral Systems Laboratory at the Ecole Polytechnique Fédérale, Lausanne. The lab's main research interests are high-quality color reproduction, digital typography, and multiprocessor-multidisk systems for image and multimedia storage systems. Hersch and members of the lab contributed to several European projects, including ESPRIT's Spirit, which produced a prototype grayscale-character PostScript previewer. Hersch is editor of the book* Visual and Technical Aspects of Type *(Cambridge University Press, 1993).*

Hersch received an engineering degree from ETH Zurich and a PhD in computer science from EPFL. He is a member of the editorial board of Electronic Publishing, Origination, Dissemination, and Design, *published by J. Wiley.*

Claude Bétrisey *is a member of the technical staff at Microsoft Corp., where he is working on the development of font-production and hinting tools. He is the principal developer of the RastWare character-generation software produced by Ecole Polytechnique Fédérale, Lausanne.*

Bétrisey received a PhD in computer science from EPFL. His dissertation was on automatic hinting and optical spacing of bilevel and grayscale characters.

Justin Bur *is a research assistant at the University of Montreal, where he is working on the development of 3D cartographic software. His research interests include the development of tools for computer-assisted type design. While at Ecole Polytechnique Fédérale, Lausanne, he developed the grayscale-specific grid-fitting methods used in the RastWare software.*

Bur received a BS from the University of Toronto and an MS from the University of Montreal in computer science.

André Gürtler *is a professor of letterform design, calligraphy, and the history of letterform at the Basel School of Design in Switzerland. As a typographer and type designer, his current interests include research into quality type forms for low resolution laser printers as well as for bilevel and grayscale computer displays. He collaborates with Ecole Polytechnique Fédérale, Lausanne, and is a visiting lecturer at the University of the Americas in Puebla and at the University La Salle, Mexico.*

Readers may contact Hersch at the Peripheral Systems Lab. EPFL,CH-1015 Lausanne, Switzerland, hersch@di.epfl.ch; Bétrisey at Microsoft Corp., Redmond, WA 98052-6399, claudebe@microsoft.com; Justin Bur, 692 Filiatrault, St-Laurent, Quebec H4L 3V4, Canada, justin@crim.ca; Gürtler at Schule für Gestaltung (SfG-AGS), Vogelsangstrasse 15, 4021 Basel, Switzerland.