Font Rasterization: the State of the Art

Roger D. Hersch

1 Introduction

Outlines are becoming the standard way of storing character fonts. In the late seventies and early eighties, only fonts for photocomposers were stored by their outline description. Screen fonts and printer fonts were generally stored in bitmap form. The advent of resolution independent page description languages [Adobe85] and of outline grid fitting algorithms [Hersch87] provided the means of printing a given document page with the same appearance on middle-resolution laser printers as on high-resolution photocomposers. This concept has recently been extended to display devices thanks to interactive resolution-independent window interfaces such as NeWS [Gosling89] or DisplayPostScript [Holzgang90].

Due to competition on the marketplace, formats for the description of font outlines and hints have been published [Karow87], [Adobe90], [Apple90]. The *TrueType* format, designed by Apple Computers, provides a complete langage for the description and processing of hinting commands. In this language, font manufacturers are responsible for specifying outline fonts and associated hints. Therefore, creating hints for outline characters is no longer restricted to a few specialists. Anyone willing to create a *TrueType* description for his outline font will need either an automatic tool [Hersch91b] or will have to add the hints one by one to the outline description. This tutorial presents current outline character representation techniques, gives an overview of basic and advanced grid constaints and describes the philosophy behind the *Adobe Type 1* hinting system and the *TrueType* character hinting language defined by Apple Computers.

This survey relies on literature about splines [Farin88] and on previous publications related to grid-fitting. It also includes an original algorithm for converting cubic Bézier splines into quadratic splines, as required for the production of *TrueType* characters. For a comparison of the different industrial hinting and rasterization techniques, the reader is referred to [Deach92], [Karow92].

Rendering of typographic outline characters involves three main steps: outline grid fitting, outline scan-conversion and filling. Outline grid fitting is based on the piecewise deformation and grid adaptation of outline parts

Visual and Technical Aspects of Type, Cambridge University Press,
(Ed. R.D. Hersch), 1993, 78-109

[Hersch89]. Grid constraints or hints are rules which specify how a character outline should be modified in order to preserve features like symmetry, thickness and uniform appearance on the rasterized character. Basic grid constraints are responsible for keeping the characters aligned with the reference lines [Bétrisey89], for keeping the symmetry of stems and for producing discrete arcs of acceptable quality [Hersch89].

Advanced grid constraints include *snapping* and *dropout control* for producing regular and continuous characters at low resolutions (Fig. 1).

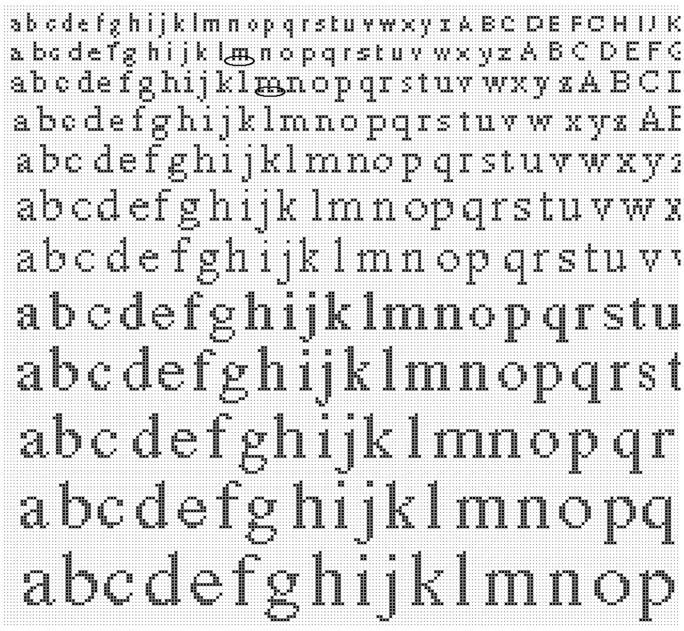


Figure 1. Character appearance with decreasing font size

2 Outline descriptions

Document description languages such as PostScript [Adobe85] require font descriptions to be invariant under affine transformations. Therefore, commercial font manipulation and rasterization systems describe character outlines using cubic splines or, to some extent, quadratic splines. In the past however, straight line segments and circular arcs were considered to be sufficient [Coueignoux81]. Several researchers advocate the use of conics [Pratt85] or conic splines [Pavlidis85].

Cubic splines are described by piecewise polynomial parametric curves. Since they are able to generate very smooth contour forms [Rogers76], we use them to describe character boundaries. Natural or clamped splines are given by two extrema and $n-2$ intermediate spline points. The different spline segments are smoothly connected at the intermediate spline points. Mathematically, smoothness is described by the continuity of the first and second derivatives. A set of $n-2$ equations [Rogers76, Farin88] can be established by requiring continuity of second derivatives. At the spline extrema, either the tangent vectors are given (clamped spline) or the second derivative is zero (natural spline). Solving this set of equations leads to the first derivatives (tangent vectors) at each intermediate spline point. Spline segments described by two points and their tangent vectors are defined up to their parametrization interval.

Cubic spline segments $P_i(t_i)$ have the following parametric equation:

$$\begin{aligned} x_i(t_i) &= a_{xi} + b_{xi} \cdot t_i + c_{xi} \cdot t_i^2 + d_{xi} \cdot t_i^3 \\ y_i(t_i) &= a_{yi} + b_{yi} \cdot t_i + c_{yi} \cdot t_i^2 + d_{yi} \cdot t_i^3 \,. \end{aligned} \tag{1}$$

Care must be taken when choosing the parameter t_i. On an ideal curve, t should be proportional to the arc length [Farin88]. To minimize computations, ideal parametrization is only applied on circular arc segments. In the case of cubic spline segments, the parameter range is generally chosen so that it is proportional to the chord length.

A character with outlines described by cubic splines and straight line segments is completely defined. For ease of scan-conversion and filling, the simple cubic spline description (1) is generally converted into an equivalent form based on the Bézier-Bernstein basis.

A spline segment with parameter t varying from 0 to 1, given by its interpolation points V_0, V_3 and by its tangent vectors T_0 and T_3 in V_0 and V_3 can be described in the following way as a Bézier spline segment. Two new control points V_1 and V_2 are computed:

$$V_1 = V_0 + \frac{1}{3} T_0 \tag{2}$$

$$V_2 = V_3 - \frac{1}{3} T_1 \,. \tag{3}$$

Points V_0, V_1, V_2 and V_3 are the control points of the Bézier control polygon (Fig. 2).

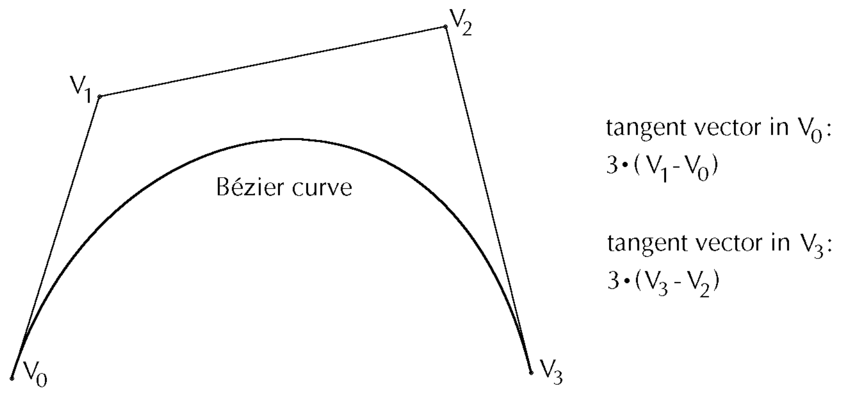


Figure 2. Interpolation points V_0, V_3, tangents and corresponding Bézier control points V_1, V_2

The corresponding spline segment in Bézier form is given by the following parametric equation for $P(u) = (x(u), y(u))$:

$$P(u) = V_0 \cdot (1-u)^3 + V_1 \cdot 3 \cdot u(1-u)^2 + V_2 \cdot 3 \cdot u^2(1-u) + V_3 \cdot u^3 \quad \text{with} \quad u \in [0,1]\,. \tag{4}$$

One can easily verify, by differentiating $P(u)$, that the tangents at the departure point $P(0)$ and at the arrival point $P(1)$ correspond to equations (2) and (3).

In order to convert a spline segment from (1) to (4), it is necessary to convert first the original spline segment with arbitrary parametrization $(0..t_k)$ into an equivalent description with uniform parametrization. For this purpose we introduce the parameter transformation $u = \frac{t}{t_k}$. The intermediate spline equation will be

$$P(u) = P(\frac{t}{t_k})\,. \tag{5}$$

The tangent $P(u)$ is :

$$\frac{dP}{du} = \frac{dP}{dt} \cdot \frac{dt}{du} = \frac{dP}{dt} \cdot t_k\,. \tag{6}$$

Parameter normalization multiplies the length of the original tangent vector by a factor t_k. This is natural, since by reducing the available time (parameter t) by a factor t_k, an object flying along the curve needs to travel t_k times as quickly to get from departure point P_0 to arrival point P_1.

Once spline segments with uniform parametrization have been obtained, it is easy to describe each of them in Bézier form by applying equations (2) and (3).

Quadratic Bézier splines are given by a Bézier triangular control polygon (Fig. 3).

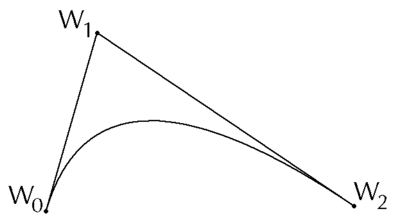


Figure 3. A quadratic Bézier spline segment given by its control polygon

$$W(u) = W_0 \cdot (1-u)^2 + 2 \cdot W_1 \cdot u \cdot (1-u) + W_2 \cdot u^2 \\ \text{with} \quad u \in [0,1]\,.$$

One can check that its tangents at the spline departure and arrival points are:

$$\begin{aligned} W'(0) &= 2 \cdot (W_1 - W_0) \\ W'(1) &= 2 \cdot (W_2 - W_1)\,. \end{aligned}$$

In the *TrueType* format, outlines are described by quadratic B-splines. Curve segment support points are either *off the curve* or *on the curve*. Off the curve points belong to the B-spline control polygon. On the curve points are tangential locations where the quadratic B-spline curve touches its B-spline polygon made up of two phantom vertices at the ends [Bartels87] and the intermediate *off the spline* vertex sequence (Fig. 4).

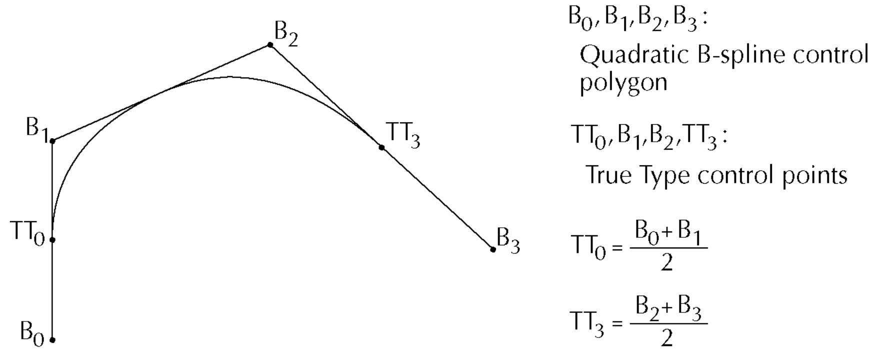


Figure 4. TrueType curve given by points (TT_0, B_1, B_2, TT_3)

Quadratic B-splines with a given parametrization (knot sequence) can easily be converted into series of quadratic Bézier splines having continuous first derivatives [Farin88]. For example, a quadratic B-spline with control points (TT_0, B_1, B_2, TT_3) having a uniform knot sequence produces the following two quadratic Bézier splines:

First quadratic Bézier spline: Second quadratic Bézier spline:

$$\begin{aligned} S_0 &= TT_0 \\ S_1 &= B_1 \\ S_2 &= \frac{1}{2}\cdot(B_1+B_2) \end{aligned} \qquad \begin{aligned} T_0 &= \frac{1}{2}\cdot(B_1+B_2) \\ T_1 &= B_2 \\ T_2 &= TT_3\,. \end{aligned} \tag{7}$$

The following algorithm converts one cubic Bézier spline segment (V_0, V_1, V_2, V_3) into a quadratic B-spline with four control points. The resulting quadratic B-spline will have at its extremity a tangent (first derivative) which is very close to the tangent of the original cubic Bézier spline segment. Therefore, the proposed algorithm almost keeps continuity of first derivative at extremity points. The resulting quadratic B-spline given by its *TrueType* control polygon (TT_0, B_1, B_2, TT_3) can be considered as two consecutive quadratic Bézier splines (S_0, S_1, S_2) and (T_0, T_1, T_2) having first-order continuity between them (Fig. 5).

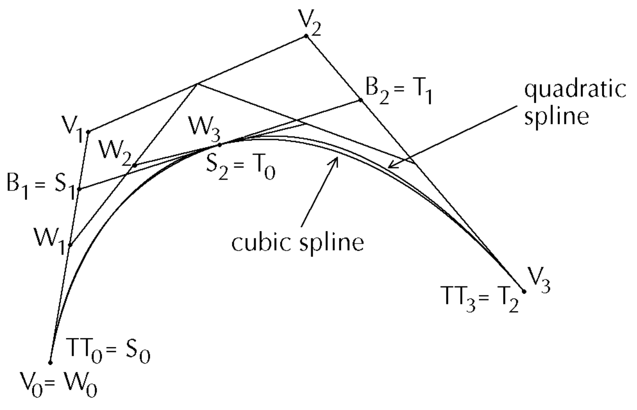


Figure 5. Conversion of cubic Bézier spline into quadratic B-spline

Quadratic Bézier spline support points S_2 and T_0 are identical and they lie on the straight line segment $\overline{S_1T_1}$. Due to (7), the *TrueType* control points of the desired quadratic B-spline are:

$$\begin{aligned} TT_0 &= S_0 \\ B_1 &= S_1 \\ B_2 &= T_1 \\ TT_3 &= T_2\,. \end{aligned} \tag{8}$$

The unknown quadratic Bézier spline support points S_1 and T_1 are computed so that the tangents at the departure and arrival points of the cubic Bézier spline and of the quadratic Bézier spline become similar. When seeking the quadratic Bézier spline support point S_1, one should consider the first cubic Bézier polygon (W_0, W_1, W_2, W_3) obtained by the DeCasteljou subdivision (see section 4) of the original cubic Bézier polygon (V_0, V_1, V_2, V_3). From this cubic Bézier spline we know that:

$$\begin{aligned} W_0 &= V_0 \\ W_1 &= \frac{1}{2} \cdot (V_0 + V_1)\,. \end{aligned} \tag{9}$$

This cubic Bézier polygon obtained by subdivision describes a spline segment having approximately the same length as the unknown quadratic Bézier spline (S_0, S_1, S_2). Therefore, their parametrization intervals can be considered as identical and their tangents made equal. The tangent at departure point W_0 is

$$W'(0) = 3 \cdot (W_1 - W_0) = \frac{3}{2} \cdot (V_1 - V_0)\,.$$

The tangent of the unknown quadratic Bézier spline (S_0, S_1, S_2) at departure point S_0 is

$$S'(0) = 2 \cdot (S_1 - S_0)\,.$$

By making tangents $S'(0)$ and $W'(0)$ equal, S_1, the intermediate control point of the first quadratic Bézier control polygon becomes:

$$S_1 = \frac{3}{4} \cdot (V_1 - V_0) + V_0 = \frac{3}{4} \cdot V_1 + \frac{1}{4} \cdot V_0\,. \tag{10}$$

Applying similar considerations, one obtains the intermediate control point of the second quadratic Bézier control polygon T_1:

$$T_1 = \frac{3}{4} \cdot (V_2 - V_3) + V_3 = \frac{3}{4} \cdot V_2 + \frac{1}{4} \cdot V_3\,. \tag{11}$$

The support point B_1 of the resulting quadratic B-spline is identical to S_1 and the support point B_2 is identical to T_1 for uniform parametrization intervals.

The deviation of the quadratic spline segments from the original subdivided cubic spline segments can be computed at the center of the parametrization intervals. In order to lower this deviation, one can further subdivide the original cubic Bézier spline (see section 4) and convert each new subdivided cubic Bézier spline segment separately into one B-spline having four control points.

3 Scan-conversion and filling: the basics

The outline scan-conversion and filling algorithm developed for character generation is an extension of the well-known flag fill algorithm [Ackland81]. It is based on the assumption that any pixel whose center lies within the continuous border of a shape is to be considered as an interior pixel. This assumption is derived from the fact that shape boundaries are relatively smooth. The shape boundary part which intersects a pixel can generally be approximated by a straight line segment. Therefore, pixels are selected as interior pixels if their surface coverage is more than 50% (Fig. 6).

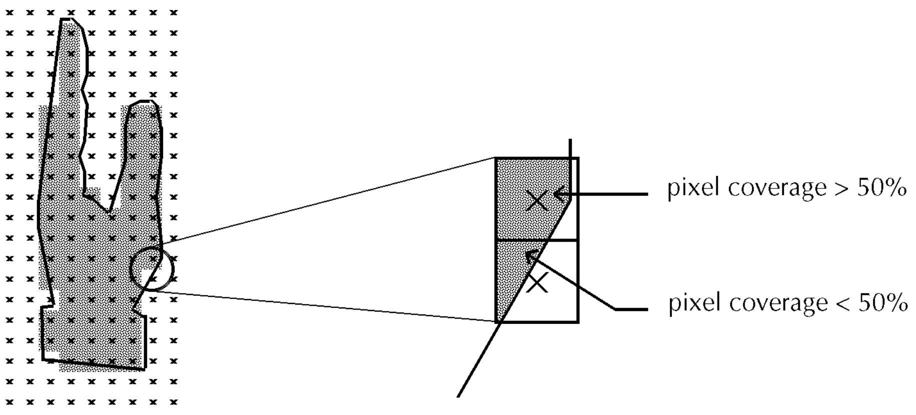


Figure 6. Interior of filled shape

The bitmap which will be generated by the flag fill algorithm can be considered as a set of black horizontal spans for the inside of the outline and as white horizontal spans for the outside. The first pixel of each span is marked by a flag (Fig. 7). Once all the flags corresponding to an outline have been set, the flag fill algorithm scans the flag image memory from left to right. Each flag encountered indicates the start of a new horizontal interior or exterior span.

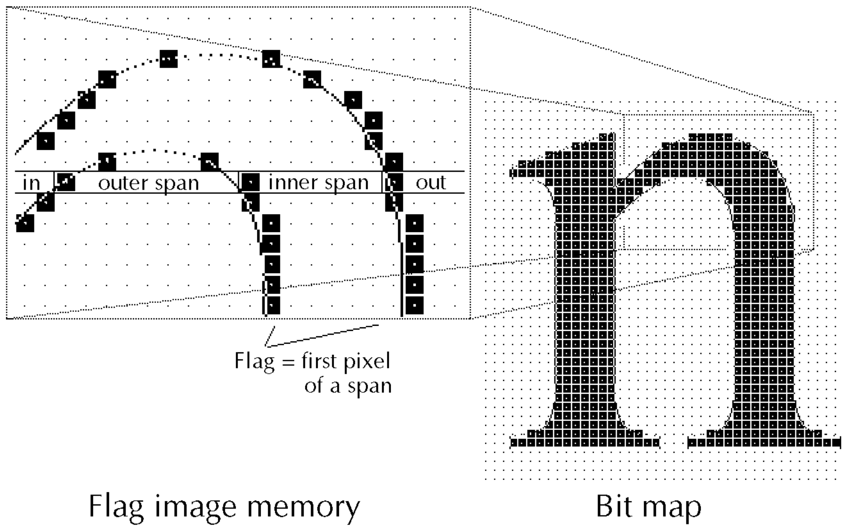


Figure 7. Example of the flag fill algorithm applied to a character

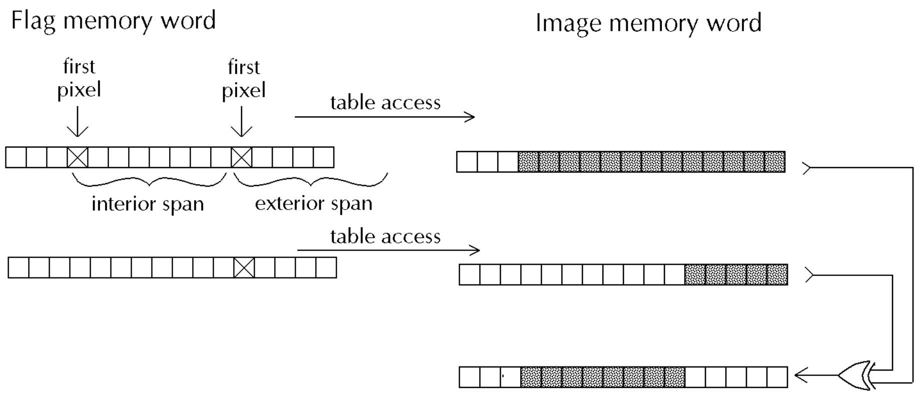


Figure 8. Illustration of parity flag fill

Filling horizontal spans lying between starting pixels can be carried out very efficiently with the help of table accesses. Each word of image memory is checked for the existence of one or more starting pixels. Each starting pixel has the effect of reversing the colour of the following horizontal scanline part. Therefore, an image memory word containing a starting pixel will be put back into memory with the run from the starting pixel to the last pixel of the word written in a complementary colour. Each subsequent starting pixel within the current word will have the effect of inverting the colour of the remaining horizontal pixel run (Fig. 8). The same rules apply to starting pixels lying in the next words of the same image memory scanline.

4 Outline scan-conversion

Rasterization algorithms described in computer graphics books [Newman79] are inadequate for the rendering of raster characters. They suggest rounding segment coordinates to integer grid values before scan-conversion. Shapes can be so rendered, but rasterization effects cannot be adequately controlled. Intermediate approaches suggest overlaying a higher resolution grid over the basic pixel grid [Pratt85]. High-resolution grid overlay may provide better rasterization control but it requires more scan-conversion steps to generate the same graphic primitive.

The last and in our eyes most successful approach is to scan-convert character contour segments with a digital differential analyzer [Rogers85] working with real fixed-point numbers [Hersch88].

4.1 Vertical Scan Conversion

The Bézier splines and line segments which make up an outline have to be converted into flags for the filling algorithm. Two strategies can be adopted to scan-convert a Bézier spline: recursive subdivision and forward differencing [Newman79]. Both strategies have been developed in order to reduce the number of required operations without reducing the precision of the scan conversion.

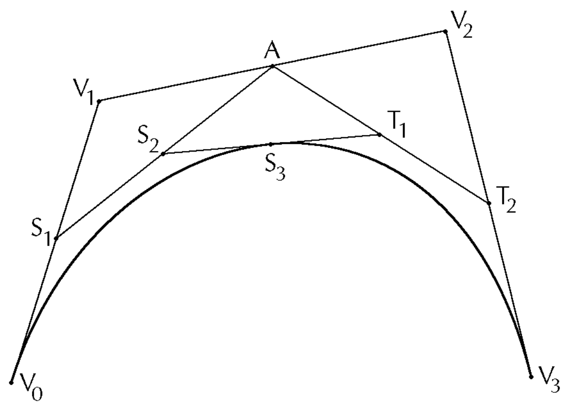
Ordinary forward differencing has one main drawback: the incremental step of the parameter used to describe the curve is a constant. Adaptive forward differencing (AFD) corrected this problem [Lien87]. AFD ensures that most of the points which are generated will be used to trace the curve. Integer AFD further improved the algorithm by using fixed- or pseudo floating-point arithmetic instead of floating-point arithmetic [Lien89] [Gonczarowski89]. The resulting algorithm is even faster.

Recursive subdivision has also been optimized [Hersch91a]. It presents several advantages over adaptive forward differencing. First, computation errors aren't amplified in the same way as in AFD. In order to get the same quality result, recursive subdivision requires a significantly smaller amount of precision bits than AFD [Morgan91]. Second, recursive subdivision can be carried out with the control points of a Bézier curve rather than its polynomial equation. This allows for a better understanding and monitoring of the algorithm. On the other hand, the recursive aspect of subdivision has to be implemented with a stack. Stack access will slow down subdivision. This problem can be partially eliminated by working with an iterative version of the DeCasteljou subdivision algorithm, where the Bézier polygon control points obtained by subdivision [Hersch91a] are explicitly stored on a dedicated data stack. If this data stack resides in cache memory, recurrent subdivision of Bézier control polygons will be as fast as adaptative forward differencing.

4.2 Scan conversion subdivision of Bézier splines

Recursive subdivision of Bézier splines is based on the DeCasteljou's theorem [Farin88]. As Figure 9 shows, a Bézier spline represented by its control polygon (V_0, V_1, V_2, V_3) can be subdivided into two smaller Bézier splines, (V_0, S_1, S_2, S_3) and (S_3, T_1, T_2, V_3).

The smaller splines will have their control polygons closer to the spline. Therefore, if a spline is subdivided often enough, the resulting control polygons can be assimilated to the spline.



Subdivision of (V_0, V_1, V_2, V_3) into (V_0, S_1, S_2, S_3) and (S_3, T_1, T_2, V_3):

$S_1 = (V_0+V_1) / 2$

$T_2 = (V_2+V_3) / 2$

$A = (V_1+V_2) / 2$

$S_2 = (S_1+A) / 2$

$T_1 = (T_2+A) / 2$

$S_3 = (S_2+T_1) / 2$

Figure 9. DeCasteljou's subdivision of Bézier splines

One of the delicate points of the algorithm is the criterion for stopping subdivision. It is based on the convex hull property of Bézier curves: a Bézier curve always lies within the convex hull formed by its control polygon.

Repeated subdivision of Bézier splines can result in three types of Bézier splines:

- splines which don't intersect any scan line, and which can be discarded since they won't generate any flag (Fig. 10a),
- splines which don't intersect any vertical grid lines, and can be assimilated to vertical line segments (Fig. 10b),
- splines which still intersect a scan line and a vertical grid line (Fig. 10c).

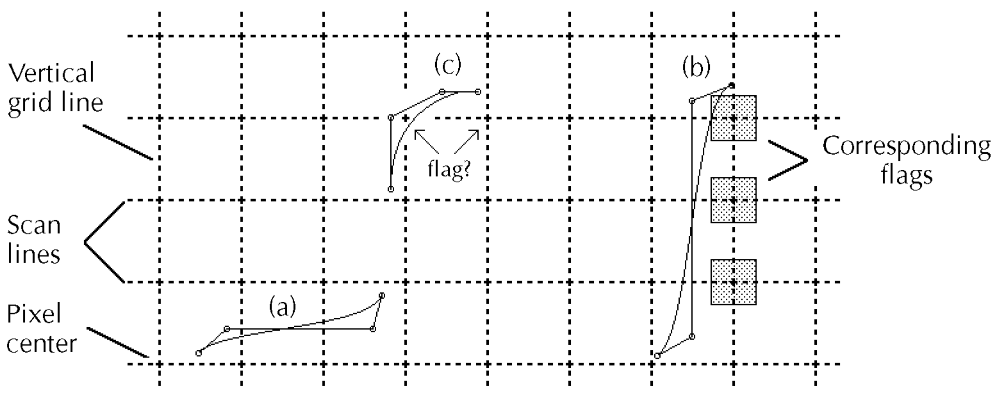


Figure 10. The three possible final outcomes of a Bézier subdivision

The third case raises the problem of knowing when to stop subdividing a spline and how to choose the position of the corresponding flag. The choice of stopping a subdivision has to take into account the level of error tolerated and the precision used in the computations.

The criterion we chose is to have the spline's bounding box smaller than the parameter *MinFracLength* along both the x and y axis. When this criterion is met, the small spline is assimilated to the three line segments which make up its control polygon. Experience shows that a good trade-off for *MinFracLength* is 1/8 of a pixel. Such a precision is generally sufficient for outline character generation, since most hinting algorithms work with a grid precision lower then 1/16 of a pixel.

5 Basic grid constraints

Rasterization algorithms working with real fixed- or floating-point numbers allow us to predict the discrete outlook of the rasterized shape. Fractional displacements of the continuous shape in respect to the grid will produce different discrete shape boundaries (Fig. 11).

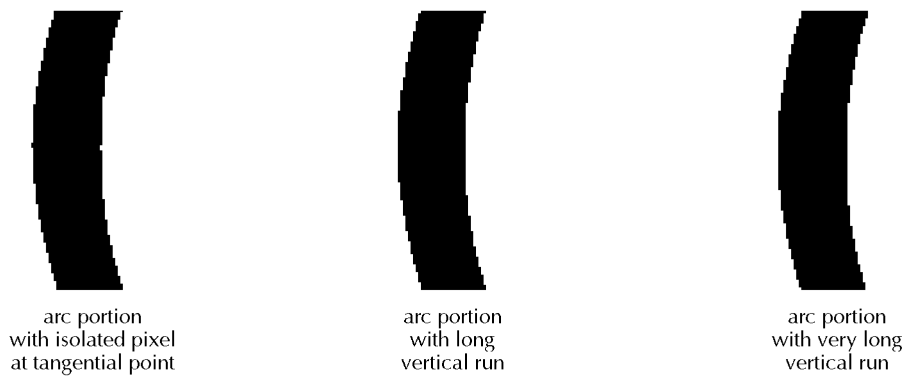


Figure 11. Rasterized curved shapes having different phase at horizontal or vertical tangential points

On vertical bars, phase determines the discrete width of the bar. In order to be able to respect original relationships between different horizontal and vertical bars, it is necessary to fix their phase relative to the pixel grid. The phase is computed so as to minimize the difference between the width of the continuous bar and the width of the resultant discrete bar (Fig. 12).

Basic grid constraints are sufficient to produce normal quality characters at 300 dpi. Each grid constraint contains a definition part and an application part. The definition part consists of a pair of points and of parameters specifying a given character part which will have to be fitted in the allowed phase range. Grid-fitting rules for fitting stems and bowls require two outline support points specifying the stem or bowl width.

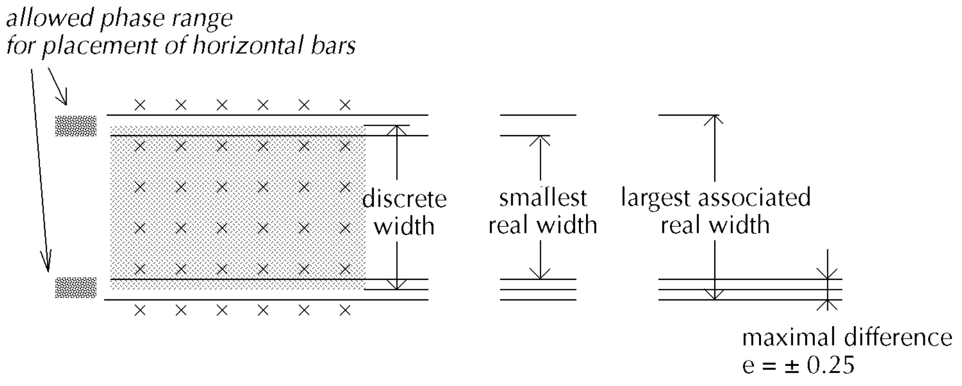
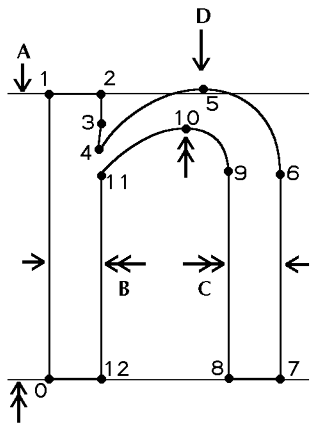


Figure 12. Phase placement of bars



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

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

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

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

Figure 13. Support points for the specification of grid constraints

The evaluation of a given hint produces a subpixel displacement which must be applied to certain parts of the outline. The application part specifies the character parts on which the computed displacement is applied by giving their starting and ending outline support points (Fig. 13).

Proportional deformation of curved outline parts can be produced by specifying a fixed point and a point on which the full displacement is applied (Fig. 14).

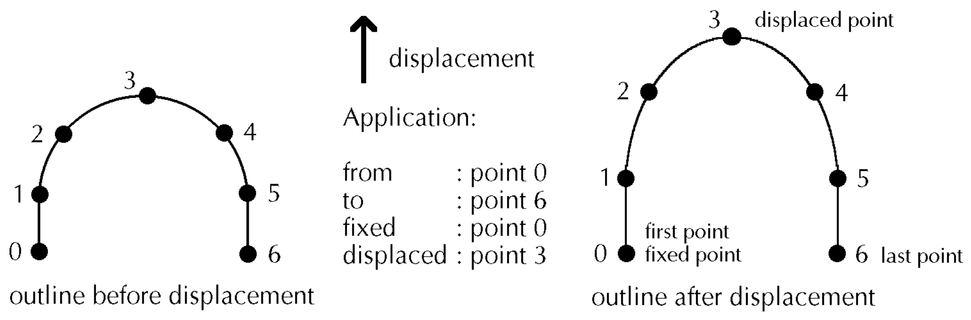


Figure 14. Proportional application of a computed subpixel displacement

Basic constraints provide facilities to center pairs of horizontal reference lines (base line, x-height line, caps line) on the grid (Fig. 15).

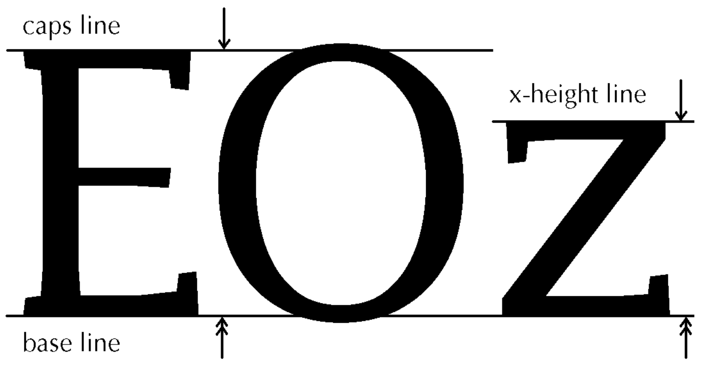


Figure 15. Phase control of reference lines

These constraints produce a vertical displacement which is used to center the character between the base line and the given reference line. The character is slightly rescaled in order to place the horizontal reference lines halfway between pixel centers.

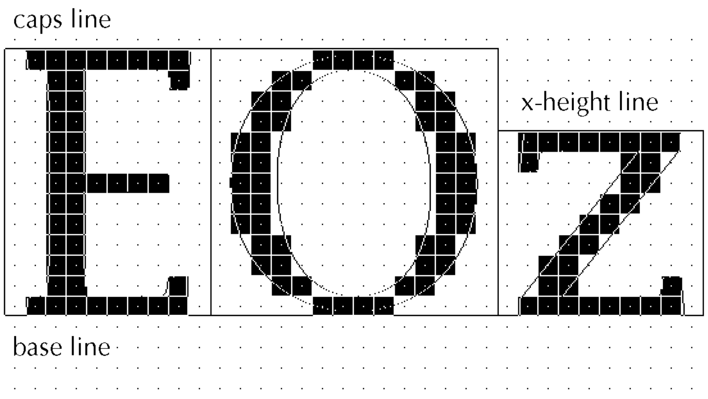


Figure 16. Result of reference lines phase control application

The initial horizontal positioning of characters on the grid is also highly important. Completely different rasterizations will be obtained if the center of a symmetric character is placed on a pixel center or half-way between two pixels (Fig. 17).

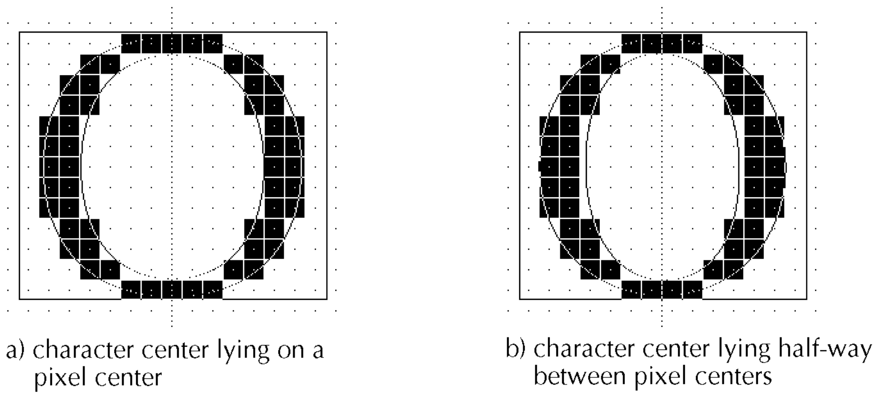

a) character center lying on a pixel center
b) character center lying half-way between pixel centers

Figure 17. Different phase positioning for the center of symmetry of a character

Horizontally, curvilinear letters are centered between their two extremal points. For characters like “C” having a curve only on one side, the width of character “O” found in the standard width table is used in order to guess the position of the character center.

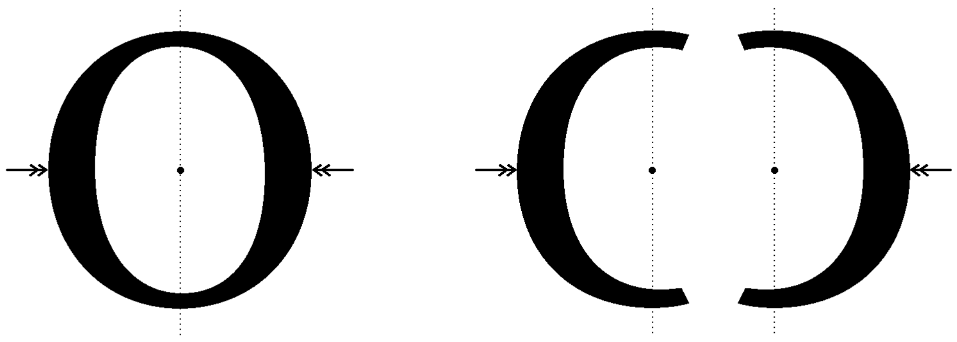


Figure 18. Horizontal centering constraints

These constraints use the width of character “o”, or “O” respectively, in order to center in the same way the curvilinear letters “o”, “c”, “p”, “d”, “b”, “q”, or characters “O”, “C”, “D”, “Q”. They should, at small sizes, provide a uniform appearance.

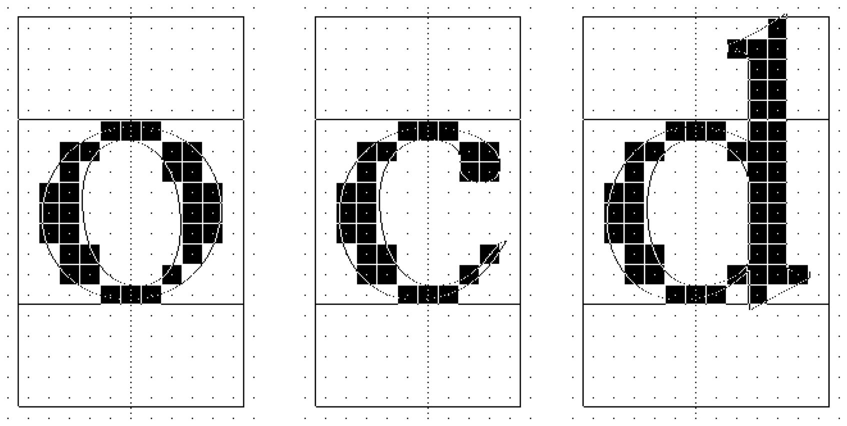


Figure 19. Uniform appearance of similar characters at small sizes

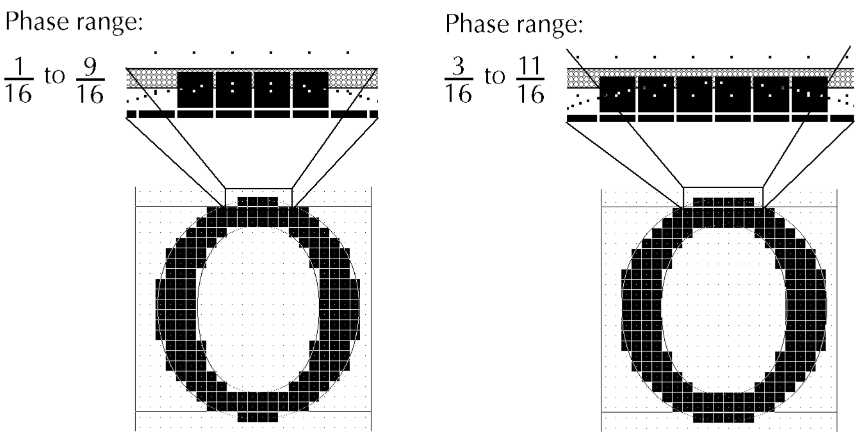

Phase range:
$\frac{1}{16}$ to $\frac{9}{16}$
Phase range:
$\frac{3}{16}$ to $\frac{11}{16}$

Figure 20. Specifying an allowed phase range for the bowl's extremities

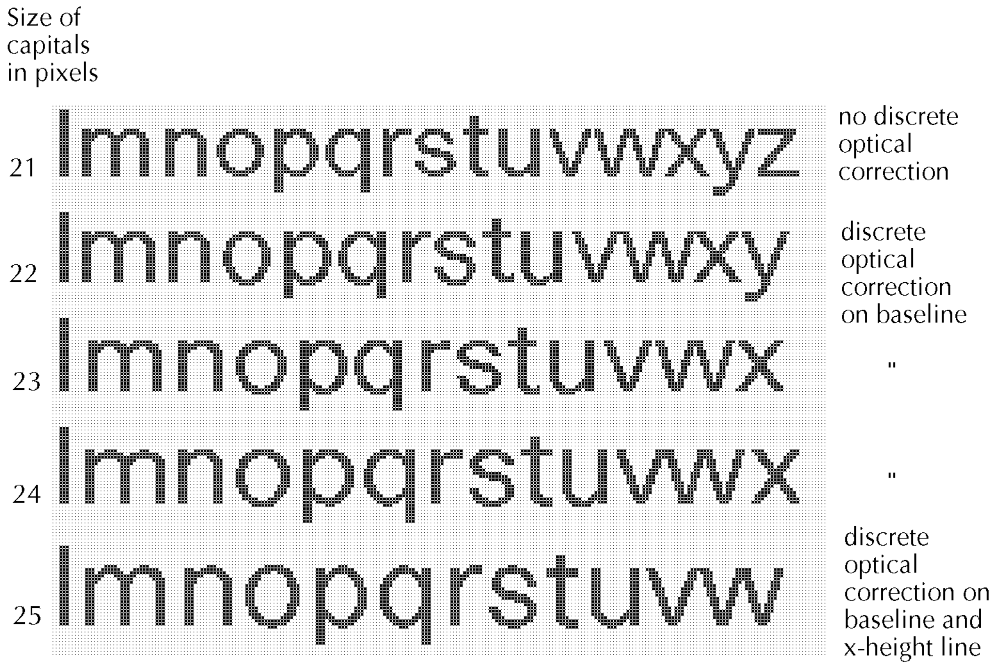

Size of capitals in pixels
21
lmnopqrstuvwxyz
no discrete optical correction
22
lmnopqrstuvwxy
discrete optical correction on baseline
23
lmnopqrstuvwx
"
24
lmnopqrstuvwx
"
25
lmnopqrstuvw
discrete optical correction on baseline and x-height line

Figure 21. Rasterized Haas Unica: optimal corrections at different sizes

Horizontal stems are controlled individually or in relation to a given reference line. Bowls and curved character parts are controlled by keeping the phase of the vertical or horizontal extremity of arcs within a given phase range. The phase range influences the flatness of the produced discrete arc (Fig. 20).

Bowls can be further constrained to produce a discrete optical correction from a given size [Bétrisey89]. Separate control is provided for optical correction on baseline, x-height line and caps line (Fig. 21).

6 Advanced grid constraints

The previously shown basic constraints are sufficient for generating raster characters at 300 to 800 dpi as long as highly regular character outlines are provided. Manual or automatic digitizing of master artwork does not provide highly regular outlines. The width of stems may be subject to some variations; curves at stem junctions may also be slightly different. At rasterization time, slight outline variations sometimes produce important variations on digitized raster characters (Fig. 22).

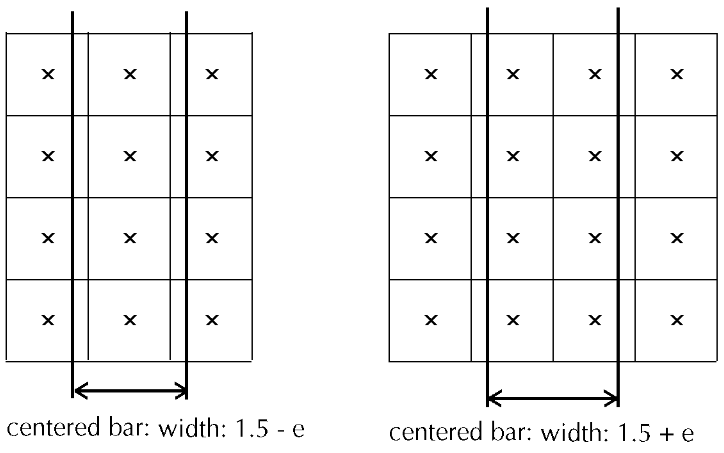
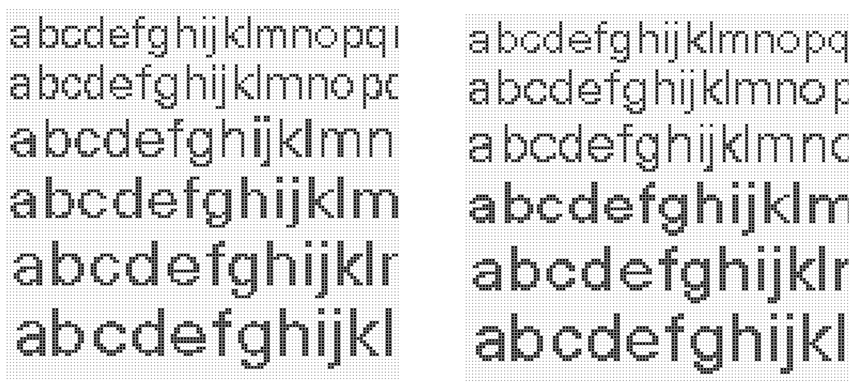


Figure 22. Small stem width variation produces different rasterizations

Therefore, for systems incorporating only basic hints, a work-intensive interactive character regularization step is required in order to produce highly regular outlines. The designers of *Ikarus* [Karow87] and of the *TrueType* character hinting language [Apple90] provided a solution to the problem of slight shape differences: they introduced a set of reference values called *snapping values* for important character metrics like stem widths. When grid contraints applied on the original outline are liable to produce an ambiguous rasterization (Fig. 22), instead of using the width specified by the constraint parameters, the reference values are consulted and the corresponding snapping value is taken.

Centering stems having slightly different stem widths will therefore produce the same rasterization, since at each hint displacement computation, identical stem widths will be taken from the reference value table (Fig. 23).



a) Without snapping b) With snapping

Figure 23. Rasterization of unregularized Haas-Unica

Snapping can be generalized for controlling serif appearance and diagonal line width. At digitizing time, serifs should start to appear on characters larger than a given font size. At smaller sizes, all serifs should disappear. Control of serif appearance can be mastered by snapping half-serif widths (Fig. 24) to predefined values. At very small sizes, the half-serif width can snap to zero and the serif will disappear.

Phase control of diagonal bars is necessary in order to ensure a constant bar width. At small sizes, snapping will also help to maintain identical thicknesses of diagonal and vertical bars. In order to be effective, phase control of diagonal bars must also ensure that bars are given by a pair of strictly parallel lines.

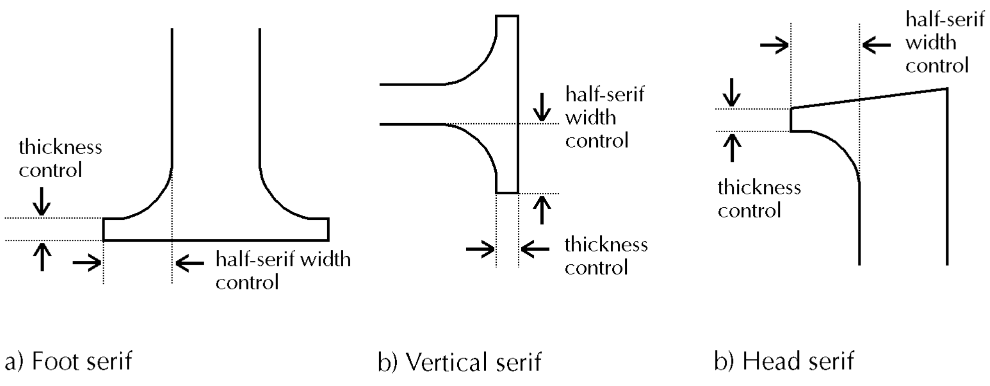


Figure 24. Control of half-serifs

Parallelizing diagonal bars given by their four extremities implies a slight rotation of one of the bar's border lines (Fig. 25). At small character sizes, the bar width in the horizontal direction can be snapped with a predefined value taken from the reference values table. This bar width is used to apply a horizontal translation to one or to both borders in order to obtain an integer horizontal bar width, which ensures that both border lines have the same phase. This means that the produced discrete bar will be of constant width.

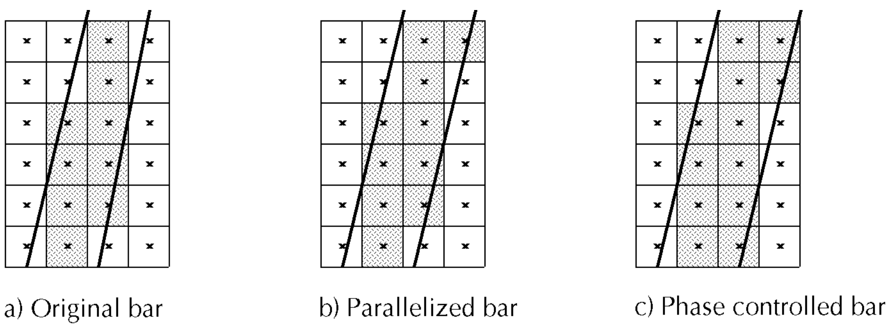
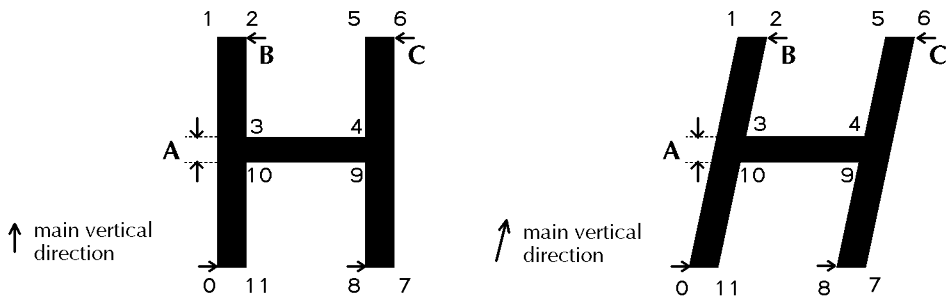


Figure 25. Control of diagonal bars: parallelization, snapping, phase control

Control of diagonal bars is only effective if the border lines are straight line segments. In many fonts, border lines are defined by flat curves. Since diagonal control is not effective on flat curves, either no control is applied at all, or flat curves must be replaced by staight line segments in an off-line process.

Italic characters can be hinted in the same way as normal characters. The rasterizer is informed by the font header whether or not the current font is italic. It will interpret standard hints found in italic characters in a slightly different, but appropriate, way. For the vertical phase control of horizontal bars, hint specifications of italic characters remain essentially the same: the current displacement direction will follow the direction of the vertical stems (Fig. 26).



Hint A:

hint specification:

vertical phase control of horizontal bar;
bar width given by Pt10, Pt3

hint application:

vertical displacement of horizontal bar along main direction;
displaced points: Pt3, Pt4, Pt10, Pt9

Hint C: similar to hint B

Hint B:

hint specification:

horizontal phase control of vertical stem;
stem support points given by Pt0, Pt2

hint application:

displacement of stem borders $\overline{\text{Pt0Pt1}}$, $\overline{\text{Pt3Pt2}}$ and $\overline{\text{Pt10Pt11}}$
if main direction vertical: horizontal phase control only
if main direction oblique: horizontal and diagonal phase control

Figure 26. Common hints for upright and italic characters

Support points used for horizontal phase control of vertical or italicized stems can be defined in such a way that the same hints produce acceptable rasterizations in both cases.

Using the same hints for upright and italic typefaces gives acceptable results (Fig. 27).

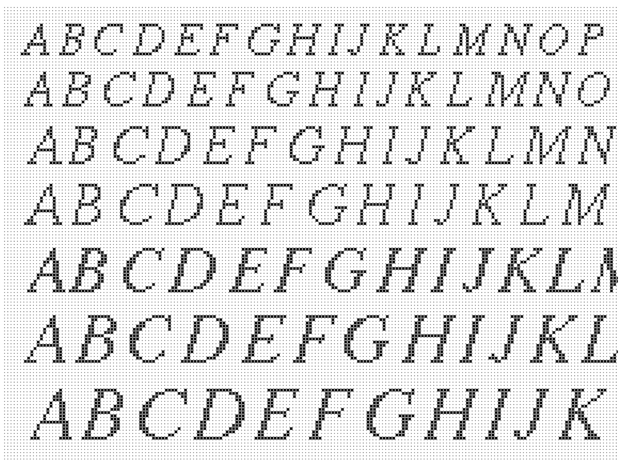


Figure 27. Rasterization of automatically hinted italic outline characters

7 Dropout control

At screen resolution it is difficult to render outline characters. Thanks to snapping techniques, it is possible to ensure that the appearance of the characters throughout different font sizes remains quite regular. But since some stroke or bowl parts are thinner than one pixel, the produced raster character may have some holes (drops). Since holes disturb the perception of the character, an artifact called *dropout control* is used in order to detect the location of drops and to insert one dot at the site of the drop. Dropout control is executed at rasterization time. The rasterization algorithm mentioned in section three is able to detect dropouts: a dropout may occur if the scan-conversion of two contour lines leads to the selection of the same span starting pixel.

Depending on the direction of the scan-converted contour lines and on their respective intersection location with the current scanline, the dropout is either an interior null-span or an exterior null-span segment (Fig. 28). An interior null-span will produce an active dot at its nearest pixel location. The inserted dropout pixel has its pixel center closest to the middle of the two contour scanline intersections responsible for the null-span. At rasterization time, fractional values of contour scanline intersections are stored in auxiliary pixmap memory locations.

Dropout control must be applied horizontally and vertically. For fast application of vertical dropout control, one can rotate the original outline by 90 degrees, rasterize it with horizontal dropout control and add the new set of dropout points to the original rasterized shape.

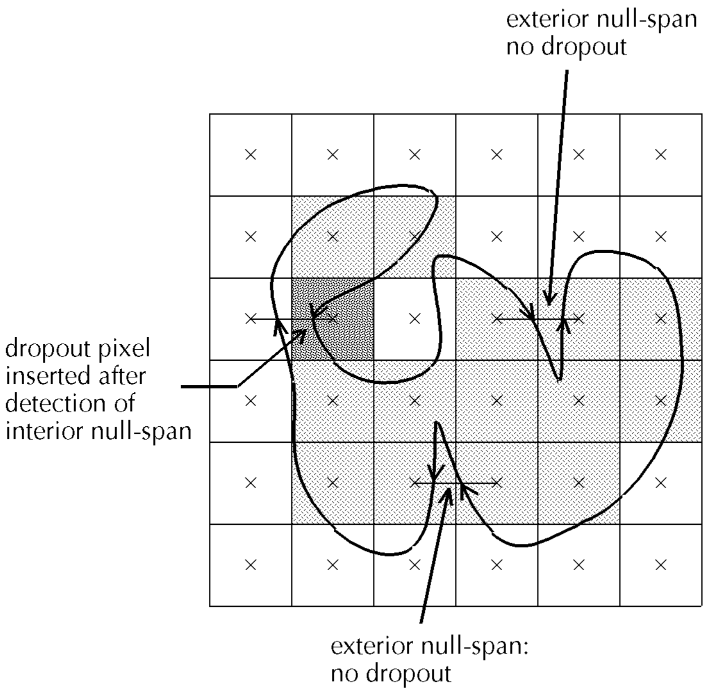
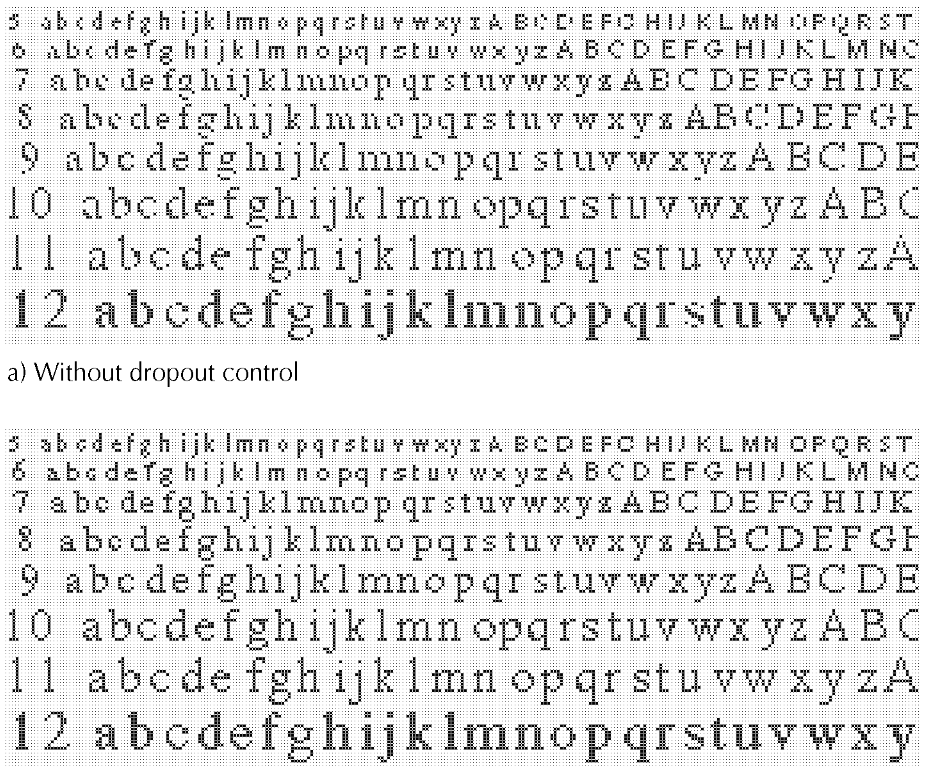


Figure 28. Dropout control

8 The "TrueType" hinting language

TrueType is a character description and hinting language [Apple90] which provides a general-purpose framework for the definition of outline fonts and grid-fitting rules. A *TrueType* interpreter has to apply the grid-fitting rules associated to the character description by deforming and adapting its outline to the grid. After grid-fitting, the *TrueType* interpreter carries out scan-conversion and filling as described in the previous sections.

In addition to normal filling, the interpreter is capable of detecting and correcting *dropouts* which occur when some stroke or bowl parts are thinner than one pixel width. Without dropout control, characters rasterized at screen resolution may have holes (Fig. 29).


a) Without dropout control

b) With dropout control

Figure 29. Characters at low resolution with and without dropout control

Fonts described in the *TrueType* language provide information about metrics, reference and overhang lines as well as snapping values. The original outline description (current glyph) can be completed by adding some positional information about reference points (shadow glyph). At grid-fitting time, the current grid-fitted glyph and the original glyph remain available.

Outline descriptions are given by points defining quadratic B-splines and straight line segments. *TrueType*-spline segment extremities lie on the outline. Intermediate control points define the spline behavior. Outline points are numbered according to the contour orientation (Fig. 30).

Glyph element pointers are used as pointers to given outline support points. The hint description language is a stack-based binary-encoded language incorporating instructions to initialize glyph element pointers, to compute distances between outline points pointed by glyph element pointers, to apply mathematical operations on values previously pushed onto the stack, and to displace individual or series of outline support points by values found on the stack.

Outline support points can be displaced vertically, horizontally or in any direction given by the current *freedom vector*. Furthermore, the language includes instructions for measuring values along a given projection vector, for adjusting the angle of diagonals, for snapping to values stored in the control value table and for defining character size ranges for certain special instructions.

TrueType is a relatively comprehensive hinting language, but it does not provide any solution for the hinting of outline characters. Font makers must develop their own hinting strategy. They can either apply their hints manually on each font, or resort to an automatic tool for producing hints [Hersch91b].

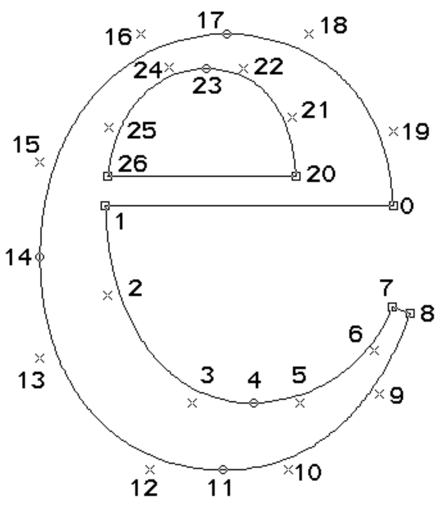


Figure 30. Example of TrueType outline description

9 The “Adobe Type 1” hinting philosophy

The Adobe approach to hinting is quite different from the hinting strategies described in earlier sections. The previously described grid-fitting techniques are based on deforming and adapting the outline to the grid. Adobe [Adobe90] and Bitstream [Appley87] apply techniques which are based on optimal distribution of grid rows and columns within the space of an outline character.

In *Type 1* terminology, hints are merely specifications of horizontal and vertical bands over the character space. Such bands (Fig. 31) are given for example by pairs of reference lines (*BlueValues*) such as *BaseLine* and *BaseLineOverhang*, *XHeightLine* and *XHeightLineOverhang*, *CapsLine* and *CapsLineOverhang*. Further parameters (*BlueScale, BlueShift*) define the limit in point size or arc depth from which optical correction affects the rasterized character.

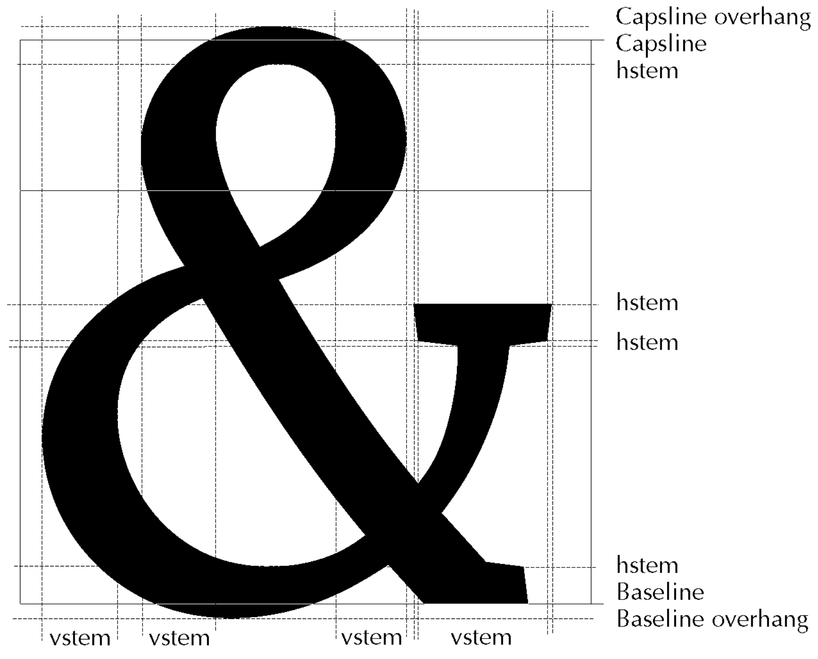


Figure 31. Bands defined by reference lines and individual stem hints

Individual hints required to produce regular straight and curved stems (bowl parts) are specified by a pair of x-coordinates for horizontal stems (*hstem*) and a pair of y-coordinates for vertical stems (*vstem*). For a given set of hints, the bands defined by pairs of horizontal or vertical coordinates should not overlap.

The *Type 1* rasterizer is capable of optimizing, for each separate character, the distribution of rows and columns of discrete pixels among the bands specified by font reference lines and individual hints.

If a given character requires overlapping hints, a first set of hints is described and applied to the character. Afterwards, a new set of hints with bands overlapping those described by the first set of hints can be used to further improve the rasterized discrete shape.

This hinting philosophy is not as flexible as the hinting philosophy offered by the Apple *TrueType* language. Hints contain only information about vertical and horizontal bands within the character glyph space. It is up to the rasterizer to use this information in an intelligent way and to optimize the distribution of pixel rows and columns along the glyph space. Distribution of pixel rows and columns is equivalent to stretching or compressing character outlines within given horizontal or vertical bands.

Due to its extended analyzing and decision capabilities, the *Type 1* rasterizer will perform more operations than a rasterizer which has to grid-fit outlines along predefined rules. But since *Type 1* hints are very simple, it is easy to develop an automatic hinting package capable of recognizing vertical and horizontal stems and bowl parts [Andler90].

10 Remaining problems

Basic and advanced grid constraints are nice tools for controlling the rasterization of simple shape parts such as straight line and diagonal bars, bowls and serifs. However, they are inadequate for solving more complex shape rasterization problems, such as the rasterization of slightly curved character parts (Fig. 32).

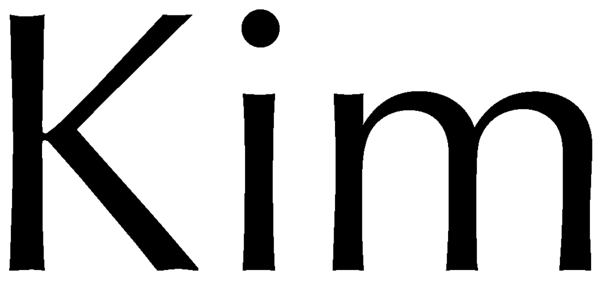


Figure 32. Characters in Optima font, given by slightly curved shape parts

As already mentioned, many shapes incorporate straight line segments for vertical and horizontal bars, but slightly curved arc segments for diagonal lines. At small sizes, and especially for bi-level screen characters, regularized straightened outlines are required. A preprocessing step is needed to generate regularized and straightened low-resolution outlines. This regularization program can make use of *a priori* information about location of nearly straight outline parts [Hersch91b]. These locations can be checked and low-curvature curves can be replaced by straight line segments.

At low resolution, many of the original font features are lost by grid-fitting and scan-conversion (Fig. 33). Essential features disappear and important geometric relationships are destroyed. Completely different fonts appear to be very similar.

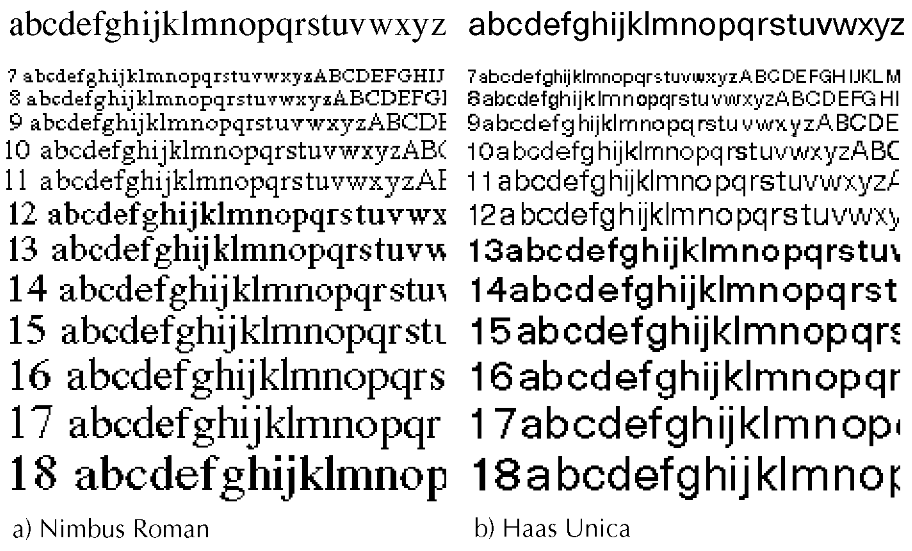


Figure 33. Nimbus Roman and Haas Unica rendered at low resolution (size: caps-height in pixels)

Since most workstations have colour displays capable of displaying 256 colours or gray levels out of 24 million, current hardware provides the opportunity to use gray levels for rendering outline fonts at low resolution.

Grayscale characters produced by filtering a large size character master [Naiman87] have not found wide acceptance in the computer industry: the characters look too fuzzy to be used for interactive work. They may however find applications for proofing purposes. Recent research has shown that it is possible to regularize the appearance of grayscale fonts and to remove some of their fuzzy appearance [Abe91].

11 Conclusions

Basic and advanced grid constraints are needed for controlling the rasterization of outline characters at low resolution. Research efforts led by industry have brought solutions for the generation of outline characters on page printers and displays. At low resolution, grid-fitting introduces significant shape distortion. Some features of the original font can hardly be recognized. Grayscale displays and variable dot size printers provide the technology for improving the appearance of fonts at low or middle resolution. Grayscale character rasterization uses hints to provide perceptually uniform stems, serifs and bowls across characters. Hints also ensure that hairlines do not disappear completely.

Further research efforts are required in order to produce more legible characters automatically at small sizes (optical scaling) and to compensate certain printer effects such as ink traps.

12 Acknowlegments

Claude Bétrisey made important contributions to the hinting techniques described in this tutorial. The author would also like to acknowledge the contributions made by André Gürtler and John Brillon, from The School of Design, Basel. They contributed to the esthetic evaluation of basic and advanced hints and to their improvements.

References

[Abe91] H. Abe, Y. Yamamoto, Y. Ohno, High-Quality Grayscale Kanjii Font Generation Using Automatic Stroke Displacement, in Morris, André (eds.), *Raster Imaging and Digital Typography II*, Cambridge University Press, 1991, 147–155.

[Ackland81] B.D. Ackland, N.H. Weste, The Edge Flag Algorithm - A Fill Method for Raster Scan Displays, *IEEE Trans. on Computers*, Vol. 30, No. 1, January 1981, 41–48.

[Adobe85] Adobe Systems Inc, *Postscript Language Reference Manual*, Addison-Wesley, 1985.

[Adobe90] Adobe Systems Inc, *The Type 1 Format Specification*, Addison-Wesley, 1990.

[Andler90] Andler, Automatic Generation of Gridfitting Hints for Rasterization of Outline Fonts or Graphics, *Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, September 90, (R. Furuta, Ed.) Cambridge University Press, 221–234.

[Apple90] Apple Computer, *True Type Spec - The True Type Font Format Specification*, July 1990.

[Appley87] Philip Apley, Automatic Generation of Digital Typographic Images From Outline Masters, *ACM SIGGRAPH'88 Tutorial on Digital Typography*.

[Bartels87] R.H. Bartels, *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, 1987.

[Bétrisey89] Bétrisey, R.D. Hersch, Flexible Application of Outline Grid Constraints, in André, Hersch (eds.), *Raster Imaging and Digital Typography*, Cambridge University Press, 1989, 242–250.

[Coueignoux81] Philippe Coueignoux, Character Generation by Computer, *Computer Graphics and Image Processing*, Vol. 16, 240–269.

[Deach92] S. Deach, Outline Font Hints and Rasterization: A Technology Primer, *The Seybold Report on Desktop Publishing*, Vol. 6, No. 7, March 9, 1992, 21–32.

[Farin88] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press, 1988.

[Gonczarowski89] J.Gonczarowski, Fast Generation of Unfilled and Filled Outline Characters, *Raster Imaging and Digital Typography*, Cambridge University Press, 97–110. (1989).

[Gosling89] J. Gosling, D.S.H. Rosenthal, M.J. Arden, *The NeWS Book*, Springer Verlag, 1989.

[Hersch87] R.D. Hersch, Character Generation under Grid Constraints, Proceedings SIGGRAPH'87, *ACM Computer Graphics*, Vol. 21, No .4, July 1987, 243–252.

[Hersch88] R.D. Hersch, Vertical scan-conversion for filling purposes, Proceedings CGI'88, in Thalmann (ed.), *New Trends in Computer Graphics*, Springer Verlag, 318–327.

[Hersch89] R.D. Hersch, Introduction to font rasterization, in André, Hersch (eds.), *Raster Imaging and Digital Typography*, Cambridge University Press, 1989, 1–13.

[Hersch91a] R.D. Hersch, Efficient Rendering of Outline Characters, *Proceedings of The Society for Information Display (SID)*, Vol. 32, No. 1, 1991, 55–58.

[Hersch91b] R.D. Hersch, C. Bétrisey, Model-based matching and hinting of fonts, Proceedings SIGGRAPH'91, *ACM Computer Graphics*, Vol. 25, 71–80.

[Holzgang90] David A. Holzgang, *Display PostScript Programming*, Addison-Wesley, 1990.

[Karow87] Peter Karow, *Digital Formats for Typefaces*, URW Verlag, Hamburg, 1987.

[Karow92] Peter Karow, *Digitale Schriften, Darstellung und Formate*, Springer Verlag, Berlin, 1992.

[Lien87] S.L. Lien, M. Schantz, V. Pratt, Adaptive Forward Differencing for Rendering Curves and Surfaces, *Proceedings SIGGRAPH'87*, ACM Computer Graphics, Vol. 21, No. 4, 111–118.

[Lien89] S.L. Lien, M. Schantz, R. Rocchetti, Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing, Proceedings SIGGRAPH'89, *ACM Computer Graphics*, Vol. 23, No. 3, 157–166. (1989).

[Morgan91] M. Morgan, R.D. Hersch, An Asic for Outline Character Generation, *Proceedings SID Symposium 1991*, Anaheim, SID Digest, Vol. 22, 201–204.

[Naiman87] A. Naiman, A. Fournier, Rectangular convolution for Fast Filtering of Characters, Proceedings SIGGRAPH'87, *ACM Computer Graphics*, Vol. 21, No. 4, 233–242.

[Newman79] W.M. Newman, *Principles of Interactive Graphics*, McGraw-Hill, 1979.

[Pavlidis85] Theo Pavlidis, Scan-Conversion of Regions bounded by Parabolic Splines, *IEEE Computer Graphics and Applications*, Vol. 5, No. 6, June 1985, 47–53.

[Pratt85] V. Pratt, Techniques for Conic Splines, Proceedings SIGGRAPH'85, *ACM Computer Graphics*, Vol. 19, No. 3, 151–159.

[Rogers76] David F. Rogers, *Mathematical Elements for Computer Graphics*, McGraw-Hill, 1976.

[Rogers85] D .F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, 1985.