Two Decades of Typographic Research at URW: a Retrospective

Peter Karow

Kreienhoopsberg 26, 22399 Hamburg, Germany

1 Background

In 1971, URW was founded as **U**nternehmensberatung **R**ubow **W**eber by Gerhard Rubow and Jürgen Weber. I joined as a third partner in early 1972 and didn't ask for a name like UKRW. I was engaged as president (CEO) together with the other two founders until the sequestration of URW in January 1995. URW++ was founded in 1995 as a follow-up company by former employees.

In 1972, we started to program the IKARUS system which became the world's leading program for producing digital typefaces. In 1975 together with Hermann Zapf, first results and achievements were presented to the participants of the annual ATypI-meeting in Warsawa, Poland. The first buyers of IKARUS systems were the companies Hell (1976) in Kiel, the former Compugraphic (1978) near Boston, and Autologic (1979) near Los Angeles. Among 70 major installations world-wide, we sold three in the USSR and two in the DDR. Today at URW++, the business with IKARUS systems is coming from Japanese newspapers and publishing companies.

From 1974 to 1977, we programmed computer-controlled output for the embroidery industry (company Gunold near Frankfurt). In a very early but simple approach, we programmed automated kerning for text stitching.

During the years 1979 – 81, we developed the concept for improving the quality of bitmap fonts produced from our outline format, mainly for Hell and Autologic. URW programmed the first intelligent font scaling package.

In 1981 at Stempel/Linotype in Frankfurt/New York, we programmed the LINUS system, the first auto-tracing system for scanned input. It had the task of automating digital font production.

Since 1976, URW itself used the IKARUS system for offering services to its customers and to build up the world's largest font library (3500 fonts). Today, this IK-lib consists of Latin (95%), Chinese, Korean, Japanese, Thai, Russian (Cyrillic), Greek, Hebrew, Arabic, and Devanagari typefaces. In respect to Latin fonts, it comprises the major part of ITC typefaces.

In 1983, we started the SIGNUS system for producing signs. SIGNUS was and is used in the signmaking industry. URW achieved a market share of about 60% in Germany around the year 1990. The SIGNUS system has been further developed to become an advertisement layout system called Admaster which is used by newspaper companies. It uses masters to automate the layout of standard advertisements.

In 1984 and 1993 I had a private collaboration with Karl Gerstner. According to his ideas, I had to program and plot many iterations and interpolations of astroids/hypercycloids (yellow), diagons/rhombuses (red), circles (blue) and supercircles (green) in

order to get outlines helping to illustrate Karl's *color form model* in a two- and three-dimensional manifold.

During the years 1985 until 1994, together with Margret Albrecht, we developed programs for improving the quality of justified text. We were encouraged by Hermann Zapf and received many good advises from him. Together, we programmed the *hz*-program for typesetting (named after him) in order to achieve advanced typography. We developed tools for the following application programs:

- *jp* for justification per paragraph,
- *kf* for kerning on the fly,
- *ek* for typographic expanding/condensing,
- *Kρ* for optical scaling,
- *In* for interpolating the weight of typefaces, and
- ScreenMaster for the grayscaling of text on display screens.

What did we experience and learn? In the following two chapters I will first summarize some key achievements and the lessons I learned as *results*, and then emphasize a few front edge topics which deserve investing some energy.

2 Milestones at URW

2.1 Digital outlines

In 1972 we wanted to automate the font production for photo-typesetting equipment. Usually, one took printouts of typeset characters, enlarged them, drew their outlines, and hand-cut new "originals" (Fig. 1).

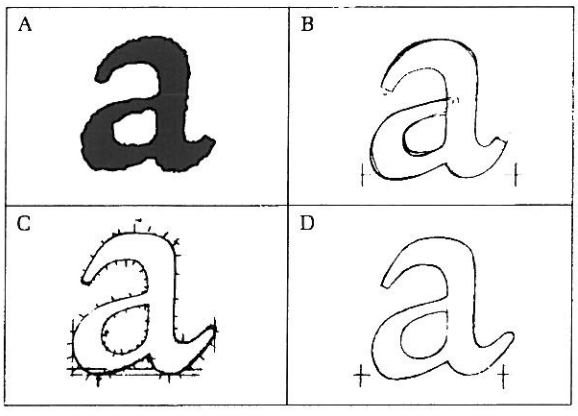


Fig. 1. A: enlarged reproduction of printed character. B: partially redrawn with interpretation. C: first digitization of the corrected outline. D: final drawing.

We decided to use a flat plotter for cutting outlines of fonts into vinyl (rubylith) in a size of 15 cm (about six inches) per em in order to get an accuracy far beyond that of

hand-cutting. At that time, plotters could only draw lines and circles. Therefore, the IKARUS-format had to be output as straight line and circular arc segments. When necessary, they had to respect continuity conditions. Typically, the outer outline of the character O consists of 24 arcs. That kind of data was neither suited to be stored nor to be digitized and hand-tuned (edited). We decided to take the input specification directly as the storage format. We defined four kinds of points: start, edge (corner), curve and tangent as in Fig. 2. We used absolute coordinates which facilitate the calculation of Delete, Shift or Insertion of points (for more arguments, see [1], page 185 ff.).

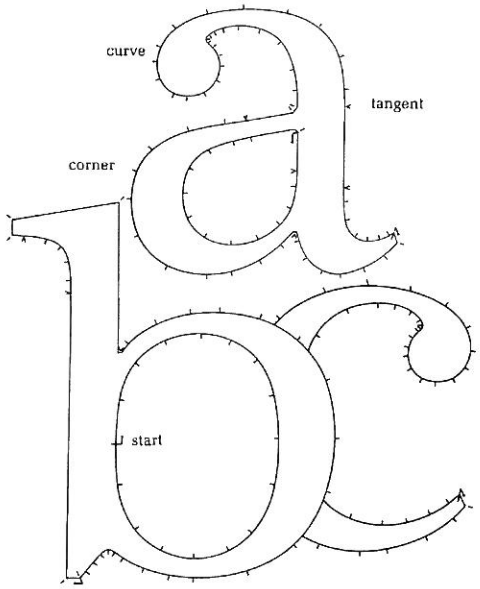


Fig. 2. The IK- format has only four different kinds of points.

Results are the following:

1. The key to success was to define the in- and not the output as the standard format.
2. Companies are never really happy when you come and tell them that they can save time and money. Perhaps, you are talking with people proud of having a big department which may have to shrink as a result of automation!
3. You give them ideas which they try to implement themselves possibly in order to climb up the company's internal hierarchy.
4. Some of your clients are so clever that they use your system in the reverse manner, believe it or not.
5. IKARUS succeeded a little bit in cutting vinyl for a while, but became successful as the first soft scanner by accident.
6. I did not consider the format as an invention; therefore I never applied for a patent. Later I learned the lesson. Please remember: *Experience is a good teacher, but a very expensive one.*

2.2 Rasterizing

In 1976, most of the people in the field of typesetting equipment believed that only a scanner could rasterize the image of a character. Mergenthaler (Linotype) bought flying spot digitizers for $ 0.5 million each, and their managers were reluctant to use software. We sold IKARUS as a soft scanner to all those who didn't have a real hard scanner. Fortunately, a lot of companies wanted to make large laser printers, many of them invented their own bitmap-formats — more or less by accident — which gave us a lot of programming jobs.

Company Hell asked us what should be controlled (Fig. 3), and how to develop the first intelligent scaling of characters. Because at university, we were previously active in pattern recognition, we first solved the problem by automated stem-, curve-, and serif-detection — this was the first auto-hinter — and then we modified the character elements according to the font guide-lines to get intelligently scaled bitmaps at the various required point sizes (Fig. 4). With one program, called PA, the production took 0.5 char/sec on a VAX computer. Hell was happy. Autologic wasn't because one had to learn how to set the parameters for a font. This could be only done by skilled people, but not by their font people hired from the street. For several hours we gave them a course and explained them for example the meaning and use of the %-notation, but without success.

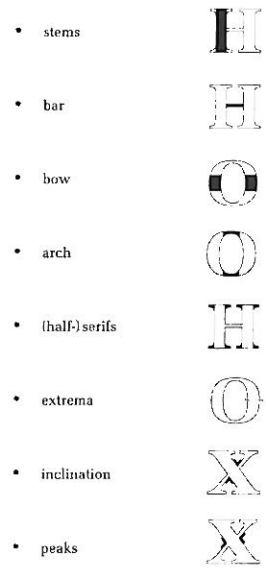


Fig. 3. Overview of important character parts to be controlled for laser printing.

At Palo Alto in 1983, John Warnock showed us his PostScript output on a large Canon laser printer. We pointed to the changing stem widths of a "m" within the 12 pt output, asked them to buy IKARUS and PA, and said it could be demonstrated at Autologic. In 1984, John told us at the Hannover fair that they are not going to use IKARUS and any kind of stem control. But Adobe had hired Sumner Stone (Autologic's manager

of digital font production). In 1985 they separated the rasterizing task into two parts: hinting by hand, and execution of hints in a rasterizer (10 char/sec). Adobe bought their first 250 outline fonts from us. After 1986, four patents were applied for an outline-format with hints and given to Adobe, Folio, Bitstream, and Compugraphic. It took very long until "auto-hinting" was invented a second time.

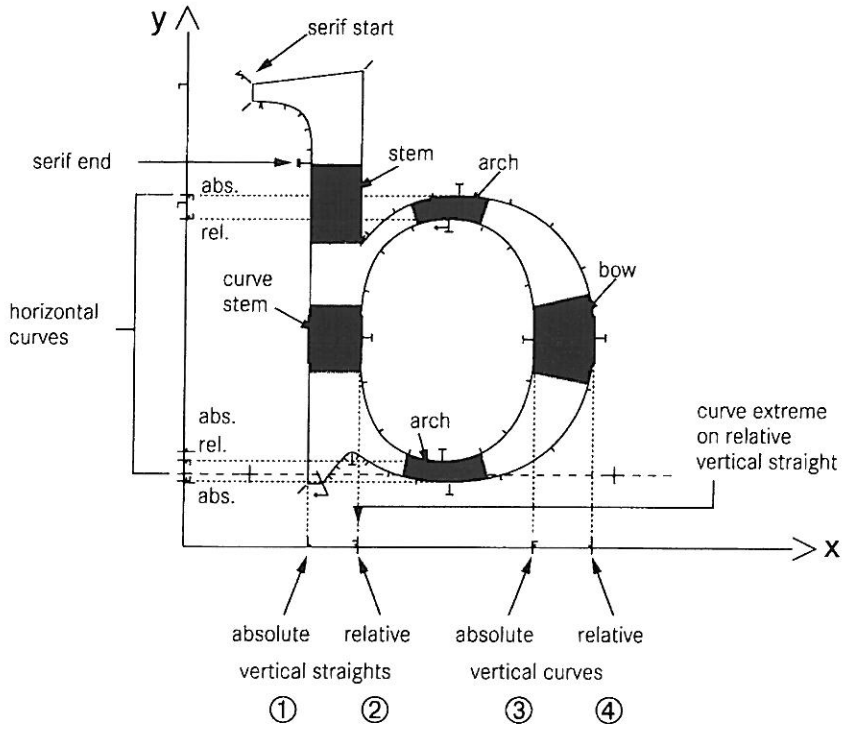


Fig. 4. Illustration of an outline with hints.

In 1988, several people from Apple Computers Inc. came to Hamburg to get a course on intelligent font scaling. They didn't want to buy our rasterizer, but to pay good money for lessons. So, we helped them to develop TrueType. Believe it or not, later it was programmed by one person, Sampo Kaasila, hired from Folio, who left Apple after completion of the first version. The results are the following:

1. Hand-hinting succeeded because auto-hinting is too complicated. I still doubt it.
2. Key technology could never be kept in one company, especially if the company doesn't see its importance.
3. Too many hints are made and executed daily. At 600 lpi, it is enough to do just stem-control for vertical stems and baseline-control. For displays, no hints are necessary, see next section.
4. In any case, there is a resolution funnel for bitmapping (see Fig. 5). At coarse resolutions only a few fonts can be represented distinguishably.

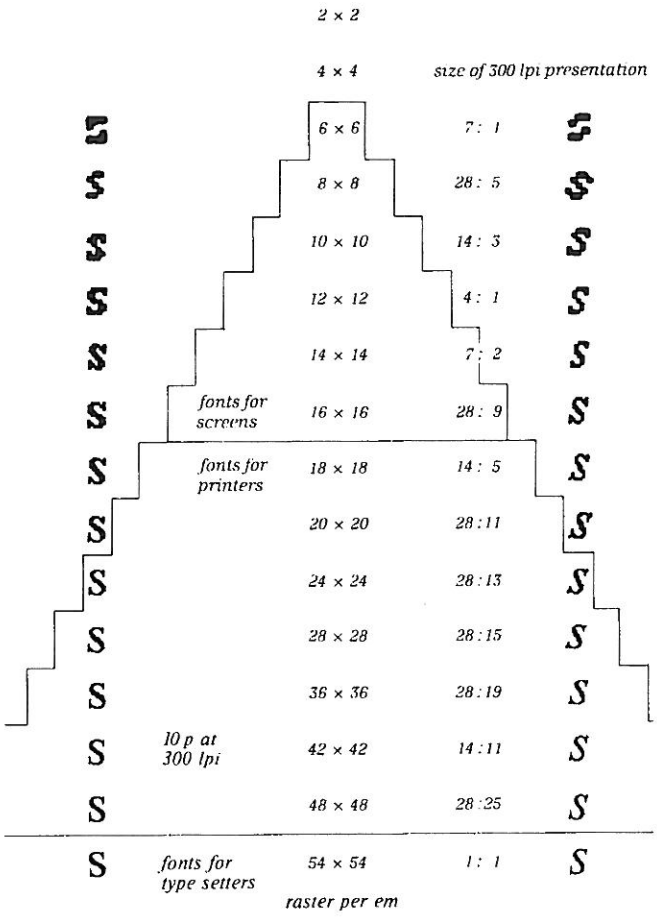


Fig. 5. The resolution funnel shows that beyond 6 × 6 pixel per em there are no readable characters to be made. At 9 × 9 there are about 6 distinguishable fonts, at 16 × 16 about 20.

2.3 Grayscaling

In 1986, URW ordered several grayscale screens from AEG in Ulm according to our specifications in order to develop a text input terminal for prepress. We obtained the best results with sub-pixel positioning for character form *and* position (8 × 4 subpixel per screen pixel are optimal) and with unhinted fonts. Unfortunately, the direct-rasterizer was to slow, about 10 char/sec. At that time, cache memory was not large enough to store 8 (32) graymaps per point size instead of one bitmap, and Hell didn't want to invest into large memories and new bus structures.

In 1993, once again, we started grayscaling of fonts for color screens using sub-pixel positioning, and no hints. This time, we were fast enough: 200 char/sec into cache, one master character in memory for 32 graymaps per character, and 4000 char/sec out of cache into video memory on a Macintosh at 32 MHz! We called it ScreenMaster.

Working at Adobe in 1995, I learned that hints were the major invention with respect to digital typefaces and that they belong to them due to the typefaces' digital nature. Therefore, in general, (gray) scaling without using hints would be a sacrilege despite the comparisons and tests made by their font people which showed that the best results were obtained without hints.

I admit that one can do special bitmap fonts which are very readable, e.g. the 12pt-versions of Geneva and New York as system fonts. These bitmap fonts however have their individual spacing values. Therefore they can't represent WYSIWYG text written in Helvetica or Times Roman (Fig. 6).

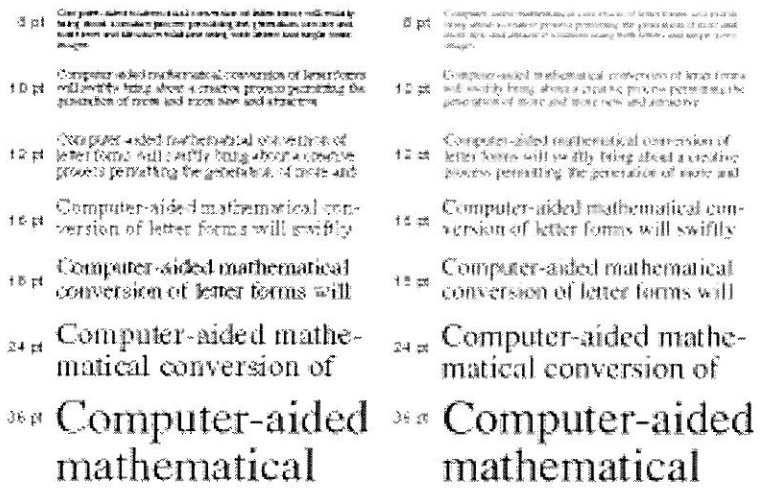


Fig. 6. Left: Bitmapped characters are less readable at lower point sizes, the color of the text sways through the "quantum jumps" between 16 and 18 pt. Right: Non-hinted grayscaling is more readable, faster by a factor 1.8 at 9×9 pixel per em, and is able to represent more than 100 distinguishable fonts at coarse resolutions (6 – 12 pt at 72 dpi).

Regarding the grayscaling implementation in Windows'95, it is a progress compared with bitmaps, but a mistake compared with that what could be achieved already. Hint interpretation makes it too slow and no subpixel positioning is done for characters. Therefore, word images are still aliased.

The results of the our studies are the following:

1. During reading, human beings focus just at 15% of all text, the rest is perceived as gray image. Therefore we have experience with gray and are very tolerant.
2. Correct word images are more important for fast reading than correct character forms.
3. Bitmaps could be used as system fonts.
4. Graymaps should be used for WYSIWYG in connection with subpixel-positioning.
5. Computer and TV people still believe to belong to a different world with respect to their needs for the representation of typefaces on displays. I believe the TV people are right when they use grayscaling with subpixel-positioning.

6. Reading is the point, not inspection of characters.
7. Bitmapping can't reproduce script or black letter typefaces, see Fig. 7.

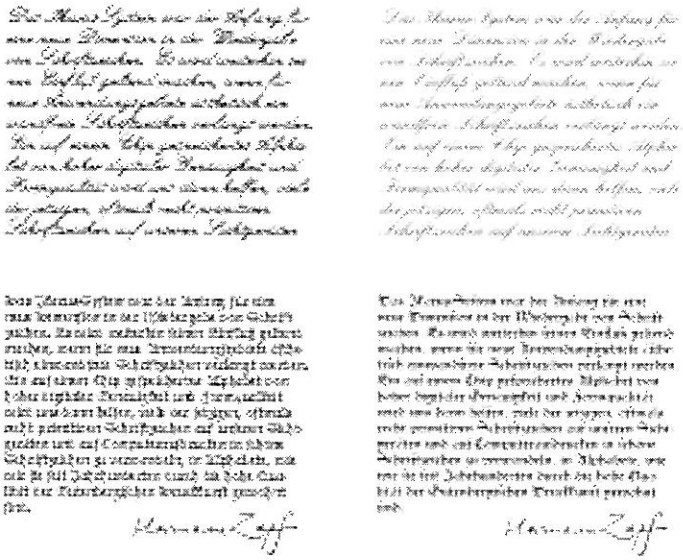


Fig. 7. Left: At 16 × 16 pixel per em bitmapping can't reproduce script and black letter typefaces as well as signatures. Right: Grayscaling does it at 16 × 16 quite well.

2.4 Auto-tracing

Between 1981 and 1987, we tried to improve our auto-tracing algorithms. It is a typical case for pattern recognition and requires several hierarchical steps. Unfortunately, this problem could not get enough scientific sex-appeal to attract somebody at a university in order to study it in more depth.

In a first step, we followed all pixels along the contour which were transition points from black to white in order to get the so-called "short-vectors" of the scanned figure. We took 3 × 3 matrices with the actual transition point as center in order to look up

1. where to go and which pixel to take as next coordinate of the short-vector contour,
2. how to improve the position of the actual center point in terms of quarters of the length of the pixel area according to the pattern of the 3 × 3 matrix, and
3. used three consecutive matrices to decide whether an actual white or black contour pixel should be regarded as noise and deleted.

The next step was edge detection. We used to fit either straight lines or parabolic arcs piecewise to the edges. These simple geometric shapes can be quickly fitted onto about 5 to 20 pixels of a scanned black figure. The critical point is to distinguish narrow curves from corners. It is also difficult, but less important, to separate flat curves from straight lines.

In a last step, we were able to fit the IK-format, PostScript cubic splines, or TrueType quadric splines to the chain of short straight line and parabolic arc segments. We called our auto-tracing software LINUS. To my opinion, it is the best auto-tracer and superior to Illustrator for example, but still not good enough. The results are the following:

1. Too many auto-tracing programs are sold, and often they have a poor performance.
2. Perfect auto-tracing is very difficult, it belongs to the discipline of artificial intelligence. Unfortunately, human beings are very, very good in this art; and therefore it seems that scientists either are too fearful to compete or are just misled to look at it with interest because it is "too easy for us".
3. Probably, a fourth step is necessary which takes into account that the scanned figure belongs to a certain class of objects like characters, animals, trees, buildings, clouds etc.
4. In most of the cases, hand-digitizing is more efficient than today's auto-tracing methods (see Fig. 8).

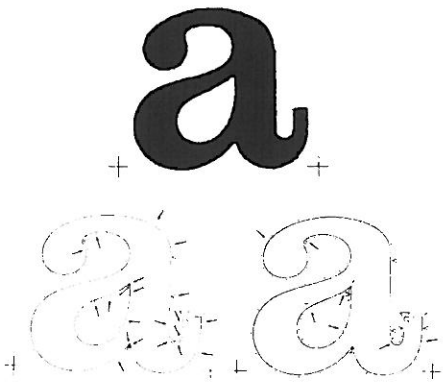


Fig. 8. Above: Possible input for digitizing. Left: An auto-traced character has to be fine-tuned. Right: A hand-digitized characters leaves less points for fine-tuning of the outline.

2.5 Auto-kerning

In typeset text, the space between the characters is used to form words. During the early nineties, we could hear people saying — especially in the surrounding of Apple Computer Inc. during the period of TrueType GX — that the space between two characters is depending on more than just these two characters. Statistically at URW, we could not find any evidence for it. We tried several times to approach the problem. For example we looked at triples and quadruples of characters in 19 languages. As a very interesting side effect from statistics, we obtained the following fact. All European languages written with Latin characters have one big point in common: about 1200 kerning pairs are enough to set 98% or more of ordinary text in one of these languages. Traditional kerning tables consist out of many useless pairs (about 30% from 200 – 300 pairs). They were included by type people who found them "sexy" to be kerned, but they don't occur frequently and sometimes they do never occur in a text.

My personal view is that there are three levels to approximate the ideal spacing of characters in order to form words:

1. traditional spacing (without any kerning),
2. pair-kerning depending on pairs and point size (Fig. 9), and
3. word-kerning depending on individual words.

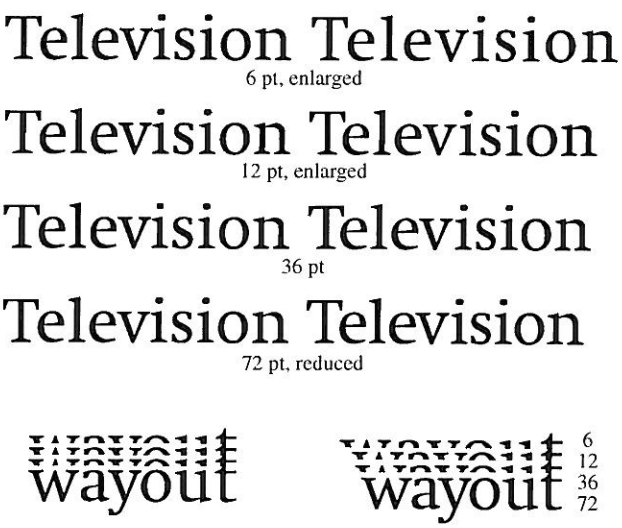


Fig. 9. Left: Traditional spacing without kerning. Right: Spacing with auto-kerning is dependent of point size.

The conclusion is the following: pair-kerning could be supplied with the font as general data. Word-kerning requires hand-tuning at the client's site because it depends very much on his taste and on the specific font and its size.

URW concentrated on automated pair-kerning and the influence of point size. We called it optical spacing. After a lot of trials, we were led to a model which assumes that the white space between two characters behaves like a viscose liquid (Fig. 10) which doesn't like to spread easily into narrow areas, for example into the inner counter on the right side of the character "c" [2]. We left aside the atom model where two characters are attracted as long as they are too far from each other, and get repulsed if they are put together too closely. We tried hard to make the viscose liquid model fast and achieved a speed of about 1000 pair-kernings per second in 1994 (*kf* for kerning on the fly).

At Adobe, we made a comparison between the results of hand-tuned and automatic pair-kerning and found that deficiencies could be observed with both methods. The deficiencies were 120% more frequent with hand-tuned kerning pairs. The comparison showed also that test persons are influenced by specific words which contain a certain pair of characters, due to word-kerning. The results are the following:

1. Kerning tables should have a length of about 1200 – 1500 pairs. The selection of pairs should be made according to their frequencies in the desired language, not by type designers.

2. Pair-kerning should and can be automated. It belongs into the computer's operating system. It must also be made dependent of point size.
3. Spacing of characters is not only dependent of point size, but also of the normal reading distance for given target objects (books, screen displays, posters, windows, signs/directories etc.), see [3], pages 193ff.
4. Kerned typesetting with mixed fonts can only be achieved by automated pair-kerning.
5. Also, Kanji could be kerned automatically both for vertical and for horizontal setting.

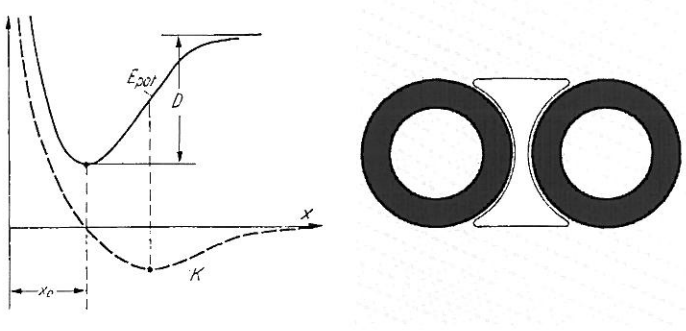


Fig. 10. Left: The Atomic model could not describe a "molecule" consisting out of two characters. Right: The white space between characters behaves like a viscose liquid.

2.6 Micro typography

We should be aware that computer publishing did nothing else than copying photo typesetting, and that photo-typesetting itself was a copy of hot metal printing.

We know that nearly everybody excuses himself for his ordinary and sometimes ugly handwriting. Therefore, we as consumers are very tolerant against bad writing. We did not punish the publishing companies when they escaped from Gutenberg's standards in order to make hot metal printing faster.

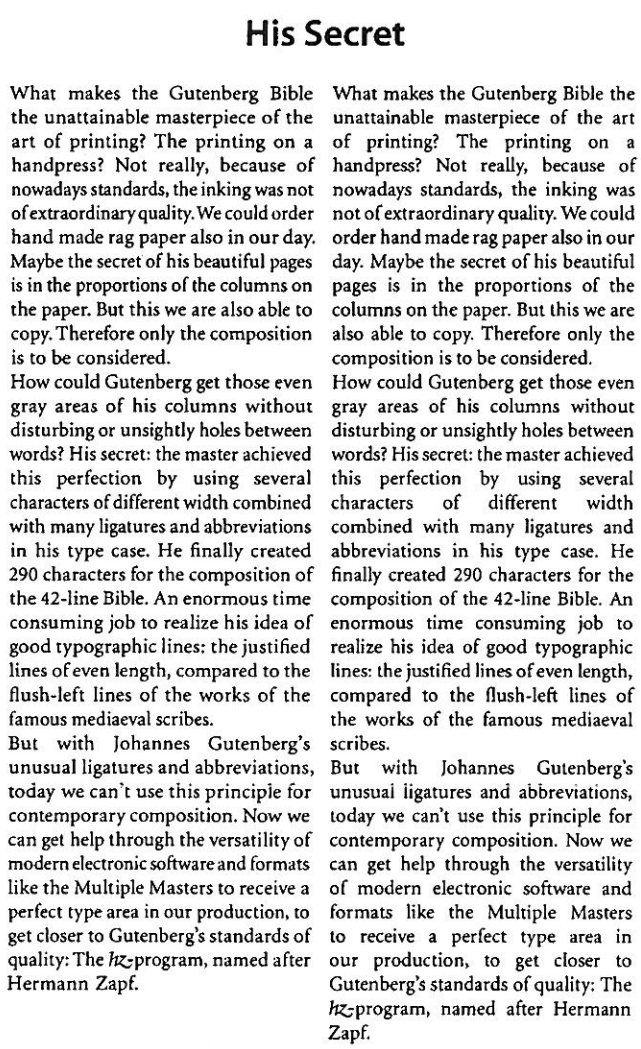
Photo-typesetting tried to replace hot metal printing. Typesetting was as good as previously; characters could even touch and overlap each other. This new technology enabled a linear scaling based only on one font for all the many possible point sizes. This became the dominant feature of photo-typesetting compared with hot metal.

While this happened and made people excited about it, they forgot the advantages of specially cut point sizes, which integrated optical scaling. Optical scaling vanished before most of us became aware of it and had a chance of sharing its reading comfort.

Desktop publishing reproduced involuntarily historical typesetting constraints. Copying and stepping into the boots of the predecessor technology reminds the beginning of the automobile industry. The first mobiles were coaches! The idea was to replace the horses. Only later, people recognized that they were able to do mobiles independently from former models.

Typographically, we have the fine tuning of fonts which was achieved by Multiple Masters, but also have more in our hands: the automated fine tuning of other ingredients such as point size, expanding/condensing, leading, spacing, kerning, and hyphenation. We can do a lot of virtual trials of typesetting which let us choose from the best and most convenient solutions for a given page of text.

We can apply subtle changes which the readers can't see; meaning that they are not disturbed during reading. There are two examples: justified text with optical margins as achieved by the *hz*-engine (see Fig. 11), and the procedure of chapter-fit.



His Secret

What makes the Gutenberg Bible the unattainable masterpiece of the art of printing? The printing on a handpress? Not really, because of nowadays standards, the inking was not of extraordinary quality. We could order hand made rag paper also in our day. Maybe the secret of his beautiful pages is in the proportions of the columns on the paper. But this we are also able to copy. Therefore only the composition is to be considered.

How could Gutenberg get those even gray areas of his columns without disturbing or unsightly holes between words? His secret: the master achieved this perfection by using several characters of different width combined with many ligatures and abbreviations in his type case. He finally created 290 characters for the composition of the 42-line Bible. An enormous time consuming job to realize his idea of good typographic lines: the justified lines of even length, compared to the flush-left lines of the works of the famous mediaeval scribes.

But with Johannes Gutenberg's unusual ligatures and abbreviations, today we can't use this principle for contemporary composition. Now we can get help through the versatility of modern electronic software and formats like the Multiple Masters to receive a perfect type area in our production, to get closer to Gutenberg's standards of quality: The *hz*-program, named after Hermann Zapf.

What makes the Gutenberg Bible the unattainable masterpiece of the art of printing? The printing on a handpress? Not really, because of nowadays standards, the inking was not of extraordinary quality. We could order hand made rag paper also in our day. Maybe the secret of his beautiful pages is in the proportions of the columns on the paper. But this we are also able to copy. Therefore only the composition is to be considered.

How could Gutenberg get those even gray areas of his columns without disturbing or unsightly holes between words? His secret: the master achieved this perfection by using several characters of different width combined with many ligatures and abbreviations in his type case. He finally created 290 characters for the composition of the 42-line Bible. An enormous time consuming job to realize his idea of good typographic lines: the justified lines of even length, compared to the flush-left lines of the works of the famous mediaeval scribes.

But with Johannes Gutenberg's unusual ligatures and abbreviations, today we can't use this principle for contemporary composition. Now we can get help through the versatility of modern electronic software and formats like the Multiple Masters to receive a perfect type area in our production, to get closer to Gutenberg's standards of quality: The *hz*-program, named after Hermann Zapf.

Fig. 11. Left: *hz*-program, hyphenation turned-off, 38 lines, last lines OK. Right: Today's software, hyphenation turned-off, 40 lines, last lines of paragraphs either too long or too short.

The basic feature of the *hz*-engine is to regard all lines of a given paragraph at once - realizing the "justification per paragraph". At first, all words or syllables are distributed to the lines altogether in a manner that each line gets a line length nearest to its given individually parametrized width (the default column width) [4, 5]. This optimization is controlled by minimizing the typographical demerits which are obtained from a function of the actual deviations from the given line widths and tolerances of the layout parameters [6].

The initial idea of the chapter-fit is to apply the same kind of automation of typography to chapters of text as is available for paragraphs. Whereas, the *hz*-engine handles and optimizes the layout of paragraphs/lines/words/characters, the justification per chapter handles the layout of chapters/columns/paragraphs/lines in order to optimize the presentation of text at a higher level. This is obtained by a procedure which scans all lengths of paragraphs and lengths of columns and balances them altogether in a manner that each individual column gets an optimal amount of paragraphs which could be managed and fitted with a minimum of demerits into its given layout. Here are the results:

1. It is not our ignorance, but our tolerance against our bad writing which is the greatest enemy of typography. Nevertheless, we enjoy reading comfort and consume typography like any other luxury.
2. Changes to sizes of the magnitude of $2-3\%$ are invisible to an unbiased reader.
3. There is a need for the European languages to get digital dictionaries providing hyphenation with increasing demerits, preferably at least the following three:
   - *easy* like "type-face",
   - *normal* like "para-graph", and
   - *hard* like "eas-y".
4. The Internet needs typography, at the time being it is ignoring it.
5. We have an enormous computing power available. We can calculate the size of books, pages, and paragraphs in advance and make decisions on the results, e.g. whether to shorten or to lengthen them in order to achieve better fits. Nobody could do this before. Even, after having done a composition, we can renew it without losing money or much time in order to optimize our messages.

3 Front edge

Don't replace horse carriages, build new mobiles.

3.1 Kerning, expanding/condensing and scaling on demand

The methods for kerning, expanding/condensing and scaling should be installed in a way that they give information on demand. With respect to kerning, storage and access should be provided as a matrix (i, j) where i is the preceding and j the following character. Both are used to address and call the kerning value. Most of the applications of today look up a kerning value from a list which makes them very slow when they should handle kerning tables of lengths larger then 1000.

3.2 Friendly Fonts

At first, one should provide a bundle of helpful statistics [7] and new typeface utilities because most of the end users are angry about having the possibility of using many fonts, but no help of the foundries on how to apply them.

There are nine possible utilities:

1. FontSetting (large sets of helpful pre-produced layouts)
2. FontInfo (*helpful* info about a font like setwidth, stemwidth, class...)
3. FontSearcher (find a font via geometric info)
4. FontDirector (find alternatives via a data base)
5. FontUser (statistical information about the use of fonts
6. FontSuitor (find fonts via attributes like male, hot, fashioned)
7. FontMixer (find fonts which work together)
8. FontInspector (compare and check fonts with own user-library)
9. FontFinder (auto-trace a word and compare the result with a library to find the best match with loaded user fonts)

Secondly, friendly fonts should have

- new kerning tables of about 1200 values
- data from above nine utilities
- East European layouts for all fonts

Could the wavelet transform give the characteristic data which one needs to distinguish them automatically?

3.3 GrayDraw

GrayDraw is thought to be an enlargement of QuickDraw. Instead of displaying bit-mapped lines and curves, it would represent them grayscaled. This could be a significant improvement of the look & feel of drawing programs like CorelDraw, Illustrator and the professional CAD-applications.

Basically, it needs a video refresh memory having a resolution which is four times higher (16 times larger memory space). Possibly, a video refresh memory could be used such as the graphic board proposed in the next section.

3.4 New graphic board

Graphic boards for color screens are wasting memory whenever they have to display black & white pixels for the representation of texts, especially, if graphic boards have 16 bit or more per pixel.

For such kinds of graphic boards (having more than 16 bit per pixel) we propose the following improvement which could be designed and constructed easily and which would not increase production costs.

One adds a flag bit per pixel which could be set to text-mode or to color-mode. All areas of the screen which show color will be supplied by 16 bit-pixels as nowadays,

triggered by the color-mode flag. But all areas of the screen which show text in black & white, are refreshed by 16 bit-pixels which are set in text-mode. These pixels are filled with the 4×4 black & white resolution of a 300 lpi-device (like color-laser printers or other color printers). It is assumed that the screen has 75 lpi, and therefore 4×4 bits are sent to a LUT (look-up table) in order to get on the fly the appropriate gray-valued pixel for the screen.

3.5 MultiType

MultiType contains also the methods for expanding/condensing according to established typographic rules. MultiType should also include optical scaling and auto-kerning.

Regarding the immense efforts for the production of MultipleMasters (0.5 – 1.0 Man year per MM) [8], I propose either an automated production of MMs or the addition of an extension to the MM-interpreting software.

Automated production would mean using IK or another program for digitizing and interpolating typeface weight. In any case, the production of two weights is unavoidable. Automated production would also mean automated production of expanded and condensed versions, automated production of three masters for optical scaling (6pt, 12pt, 72pt), and using F for the automatic generation of on demand kerning.

Extension to a type manager. The methods for expanding and condensing as well as the methods for optical scaling can be implemented into an Type Manager. It would then be possible to load a MM-font consisting of two weights only, and still be able to make the typical three dimensional interpolation. It would also be possible to make two dimensional interpolations on the basis of just one font (width and point size).

3.6 Page description

Why shouldn't we carry x-coordinates and some layout parameters in front of each character? Think about the problems which arise on the receiver's side if a specially requested font is not available. Basically, all receivers have some sort of core fonts. Using one of them according to a general information like regular/italic, Roman/Grotesque or Script could achieve at least a proper resemblance with the original layout.

3.7 Masters

Masters are layouts for diverse documents or document parts like templates or forms. At the same time, they actively guide the user's actions and enable them to focus on the few additional needed steps and parameters. Below a few examples:

LetterMaster is a program which can be used in order to write letters at home. It uses *hz* and grayscaling for display. It has a master-driven GUI, except for the editing process. It

takes a Master out of a large set in order to layout and "individualize" the letter. The user just selects the following three features: *Master, number of pages, typeface.*

AddressMaster is an application which fits an address of given text and number of lines in an optimal manner into a given rectangle. Automated features for expanding/ condensing, optical scaling and plus/minus kerning could be used without destroying some possible general rules of typography within a customer's layout.

LineMaster could be a stand-alone product and provide headlines of text with geometric distortions (circle, cylinder, globe, ellipse), contouring, shadowing, rounding, and intersection (hidden-line technology, also called set-theory for characters).

AdMaster comprises LineMaster. The objects are positioned in a hierarchical order relatively to each other. This gives a production orientated, very practical application.

VideoMaster works on master layouts for various kinds of titles (e.g. closing sequences with title and credits for clips/movies), it provides a preview in real-time. The fast grayscaling enables to generate a title of a length of 10 TV-frames in 20 seconds using subpixel positioning in order to avoid flickering on interlaced 50 lpi screens.

References

1. Peter Karow: Digital Typefaces; Springer-Verlag, Berlin Heidelberg New York London Paris Tokyo, 1994, ISBN 3-540-56509-4
2. URW: IKARUS brochure, Hamburg, 1983
3. Peter Karow: Font technology; Springer-Verlag, Berlin Heidelberg New York London Paris Tokyo, 1994, ISBN 3-540-57223-6
4. Donald E. Knuth, Michael F. Plass, "Breaking paragraphs into lines", Software-Pratice & Experience, 11/11), Nov. 1982, 1119–1184.
5. Donald E. Knuth, The TEX Book, Addison-Wesley Publishing Company, Reading, Massachusetts, 1984, reprinted as Vol. A of Computers & Typesetting, 1986
6. URW: *hz*-program brochure, Hamburg, 1993
7. Peter Karow: Typeface Statistics; URW Verlag, Hamburg, 1993, ISBN 3-926515-08-2
8. MultipleMasters are a product of Adobe Systems Inc., San Jose, California.