Introduction to iKern

iKern is a tool for letter-fitting (spacing and kerning) fonts.

iKern is a mathematical model of the interaction between separate generic shapes conceived for the task of letter-fitting.

iKern is a theory that describes possible ways of interaction that could be used to calculate final sidebearings and kerning values.

Letter-fitting can be considered as the the sum of a set of *choices* and a set of consequent repetitive procedures.

iKern isolates the set of choices and automates all the rest. The set of choices is exposed in a way that could make sense from a microtypographic point of view.

Choices are made by means of parameters. These parameters tune the way the iKern engine executes the instructions foreseen by the model.

This way, it is possible to test key combinations of letters and immediately have a visual response.

Parameters get adjusted in an iterative way, and with visual feedback, until the result thus achieved coincides with the one anticipated.

This letter-fitting paradigma differs from the traditional one:

- sidebearings and kerning values are *found* rather than created one by one;
- rhythm and texture in a typeface have to be *designed* at the beginning rather than determined at the end of a (long) process.

From this point of view iKern may be considered as an *interface* for a system of global decisions where the machine is put in charge of achieving:

- coherence: the initial choices are globally and indiscriminately inherited by glyphs and kerning pairs;
- consistency: the same algorithms and the same parameters produce both autospacing and autokerning;
- accuracy.

The speed of the process also allow to prepare different versions to be evaluated making the whole process even more efficient.

Development of iKern began in 2002.

At that time I had completed the digitazion of the *Fell Types* and had to begin with kerning.

My career in manual kerning lasted for only a few hours because I quickly realized that it would have taken me years to kern 15 fonts.

So I thought about automation:

> I was sure it would have been easy because I knew some math models from structural analysis that seemed to me usable by analogy.

I tried and failed. Tried variations on the theme and failed again.

Tried different ideas and failed again. For one year and a half.

Until I realized why I failed:

- I was trying to find an answer without knowing the question;
- I was trying to mathematically describe something I didn't understand.

I needed a theory to base the model on. And there were no serious theories around.

So I began to adopt a philosophical shifting:

Instead of thinking of letters moving in a *static theater* called *white space*, I began to describe mechanisms I imagined happening in a *living entity* called *white space*, disturbed by the presence of static letters.

In the end, these turned out to be the mechanisms that moved the letters from where they were to where I would have liked them to be:

> where the *eye* judges them to be correctly placed.

So iKern was conceived not as a substitute of the *eye* but as a model based on the *expectations* of the eye. My eye.

An eye that probably wasn't trained too much at the time, but that surely got some training in the years that followed.

Building a theory and a model at the same is like travelling whitout a map.

A *theory* begins *complex* because it has to take into account many phenomenons that seem unrelated. It has to end *simple* because it has to become as much as possible *general*.

A *model* begins *simple* because ideas come simple but has to become *complex* to be able to approximate as much as possible the *theory*.

The *complexity* of a model is a function of the *amount of information* that the theory can produce. The *development* of a *theory* depends on the reach, accuracy and extent of the experiments that a *model* makes possible.

It's a never-ending feedback, one needs to be clear about what one is doing, and one needs to understand when to *simplify* and when to *complicate*.

It's true in every creative field:

Ideas are born, grow and die generating new ideas.

It's a fundamental compass.

So iKern was born not as a description of the *white space* as a *geometric* object – but as the description of a set of *potential events* inside of it. Like a *physical* object.

This paradigm change can be described like this: The relative geometric position of letters is not considered as a matter of *geometry.* It is considered as merely being *triggered* by geometry.

This paradigm change became necessary because simple *geometry* alone would have provided a very poor vocabulary for describing the *homegeneity* of what is *geometrically dishomogeous*:

the *white space* between letters of every possible different shape.

I was more familiar with *Continuum Mechanics* and *forces*, *tensions*, *deformations*, *energy*, ... which are *concepts* far more powerful than *distances* and *areas* alone, and more appropriate for modelling non-linearities.

I'm not saying that those are the only valid concepts that lead to success.

I'm just saying that those were the words I learnt in my mother tongue.

In 2004 I published a first version of the *Fell Types* reworked with iKern and so I also went public with iKern itself.

In 2008 I began to officially offer my services.

In 2010 I left my job of engineer and became *type engineer* full time.

Today I still keep on working on the develpment of iKern with a new stable build every 1-2 weeks.

The *service* aspect is fundamental to iKern: it offers *test cases* that bring new ideas.

Working on different shapes and styles, it is possible to make the system more general.

iKern works not only with Latin, Cyrillic, Greek script fonts but with fonts of any script whose letter-fitting is ruled by a concept of *white space*.

I worked on Hebrew, Arabic, Armenian, Ethiopic, Inuktikut, Glagolitic and various ancient scripts. Invented scripts too.

And the *dialogue* with different designers, every one of them with a peculiar *point of view*, helps achieve *generality* more and more.

A few notes on the iKern model and theory

Spacing and kerning happen at the same time.

The relative position of two letters is just one and it indissolubly contains both *spacing* and *kerning*.

In iKern, there are concepts of *portion of space that is owned by a glyph* and of *compenetration space*.

These are similar, related, but are not identical with final *sidebearing* and *kerning*. (Sidebearings and kerning is just one way of storing letter-fitting. It is just how current font formats store letter-fitting.)

Spacing and kerning operations are performed in sequence but:

- The same set of parameters is used for both spacing and kerning;
- The same algorithms are used for calculating sidebearing and kerning values. In case of kerning, there are *two facing outlines*. In case of spacing the other outline is *virtual* and modeled as a *distortion* opportunely imposed on the *medium* the *white space* is thought to be made of;
- During the *spacing* phase, sidebearing values are produced. But this is the relatively less important happening:

  when spacing, every side of the glyph is analyzed to extract informations necessary for the *kerning* phase.

That's why iKern can only produce spacing and kerning together and at the same time.

Anyway, applying a certain kerning strategy to a set of glyphs spaced with a different one, would create kerning values mixed with a component of attempted numerical corrections without any possibility to separate them. And in very great quantity.

Kerning would try to *get rid of spacing*. Just like an immune reaction. No matter how good the spacing may be.

In iKern there is *congruence* between spacing and kerning and the number of kerning values eventually ends up being the minimum possible.

Distributed as in a *Gaussian bell* centered on the zero. As a *continuous* phenomenon.

The algorithms iKern is made of may be considered as part of two separated subsystems:

- the first tries to optimize *legibility*;
- the second tries to optimize *readability*.

Legibility means *separation* of shapes. So that glyphs are more *recognizable*. As a consequence, the white space must have the ability to keep the glyphs *separated*.

Readability means *equilibrium* of shapes. The concept of equilibrium implies a *principle of symmetry*: what happens to the left has to be the same of what happens to the right because every glyph has to *appear* to be *centered* in the middle of a triplet. As a consequence, the white space must have the ability to *seem homogeneous*.

The first subsystem *separates* the glyphs.

It does so by trying to impose a certain *distance* between every point in every direction in the boundary outlines.

This distance is the main parameter in iKern and it's called *Width*.

The mechanism tries to move points towards each other if their distance is larger than the expected *Width*, and tries to move points away from each other if their distance is smaller than the expected *Width*.

The medium that the *white space* may be thought to be made of is a *non-linear elastic body* which is subject to *distortion*.

This subsystem is called *Proximity model*.

The second subsystem *balances* spaces.

On the left and right of every glyph and between every pair of glyphs.

The aim is to *induce* a *perceived homogeneity* of the various white space islands between the letters.

Object of model, in this case, it's not only a set of mechanisms that may do that.

It's also the definition of a way to *evaluate* a difference between *perception* and *measure*. Between *optical space* and *geometric space*.

Generally, the *Proximity Model* will create uneven *white spaces* because all the possible shapes of the letters will compenetrate one each other in different ways and of different amounts.

These *spaces* need to be *balanced*.

And if the *white space* gets *distorted* by the presence of the glyphs then the *space* may be considered as *not uniform* because not made anymore of parallel lines like the warp and weft of a flat fabric.

It can be imagined that the *boundary outlines* project themselves in the *white space* forming *structures of forces*, inducing *tensions* and *deformations*. The *shape* propagates itself in the *white space* in form of *perturbation*.

The concept of *non-uniformity* of a space propagating *perturbations* may be combined with a concept of *distribution of densities*.

The medium, the *white space*, may be thought of consisting of *air*.

Because *balanced* as a whole. Because being *dense*. Because *moving*.

This subsystem is called *Air model*.

Air may reflow. The same amount of *air* may occupy different areas depending on the *density*.

So a *first* possible way to model *perception* is:

- redistribution of *geometric* spaces;
- Definition of a different way to *measure* the *space:*

    an *optical* space.

Perturbations break the *inner symmetry* of the *uniform space* creating the substrate for *something that can happen*.

The combined *perturbations* in a *white space* formed by two different glyphs create a new whole configuration.

A configuration may be *stable* or not.

Glyphs may *move* to resolve *instability*.

A resolution can only happen in the direction of an increased *symmetry*.

This is a possible *second* way to model *perception*.

The *non uniformity* in the *white space* may also be considered as the *fingerprint* of the shape that generated it. Because every one is as unique as its shape.

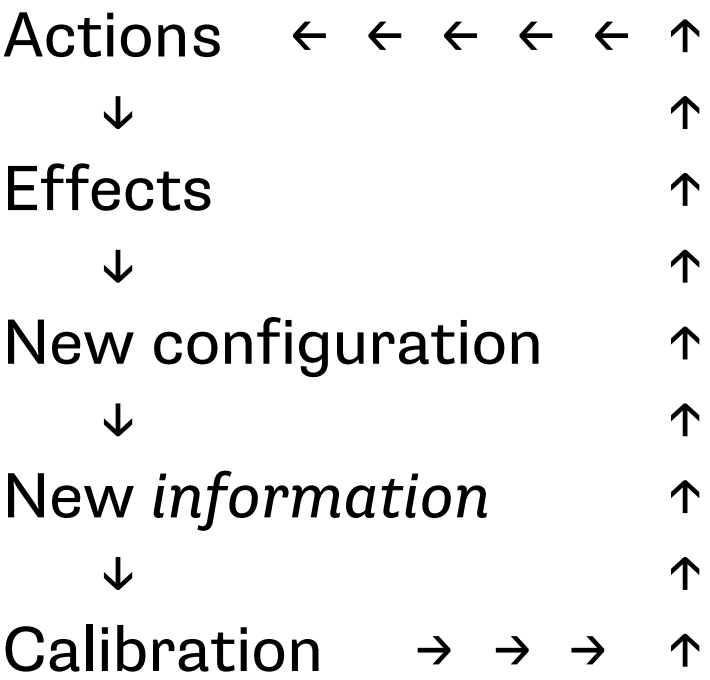
From this point of view, the *white space* may be thought of as a medium that not only accomplishes a *function* but also carries *information* itself.

[*Gestal psychology* tells us that *perception* may be the result of an efficient simplification of so-called *reality* so that *structures* may emerge from an otherwise unmanageable amount of data. *Black and white* may consequently be seen also as a *whole* sometimes. A whole that is more than the sum of the parts. This *more* would just be *information*.

I like to think that the *fingerprint information* may have some kind of relation or correspondence with that.]

In any case, the *study* of the *(non) uniformity* of the *white space* allows to extract the *information* that makes the mechanisms work. The fuel to the engine.

There's always a *recursive feedback* in the inner working of the mechanisms:

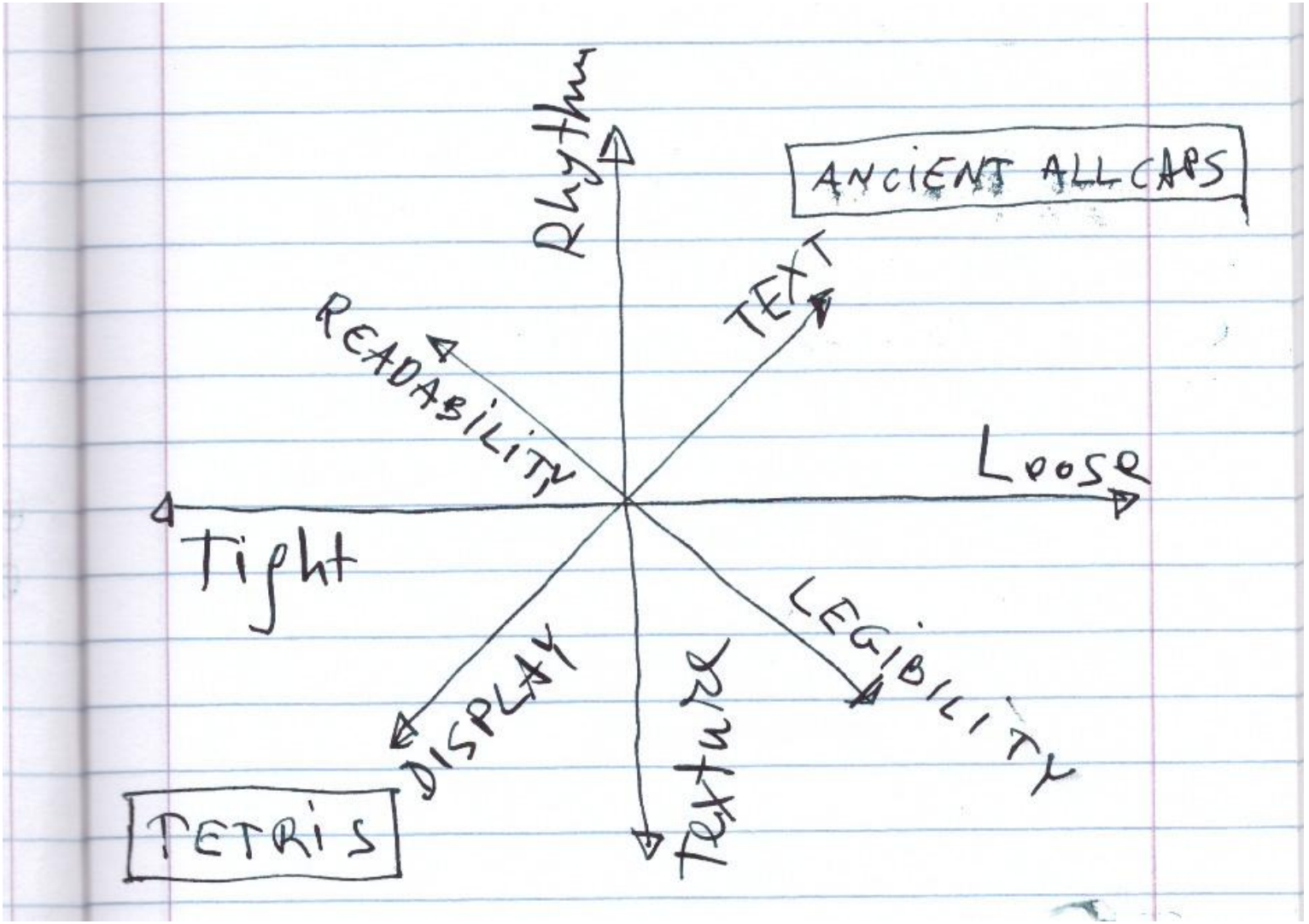


With the proper *information*, *mechanisms* not only describe *behaviours*, but also *approximate* the *transition of behaviours* even for minimal variations in the geometry of the outlines.

Instability and *recursion* become the foundation for *non-linearity*.

Context and Compromise

iKern can produce an infinite number of different letter-fitting configurations. What makes them different is the *context*.



Context, microtypographically speaking, may be defined as a point in a bidimensional diagram with two axes: *Distance of observation* and *Choice of a specific relation between legibility and readability*.

The *distance of observation* is a measure of how the *white space* between the letters is seen by an observer.

The larger the distance, the more the white space is seen as a whole, so that its predominat feature is its *extension*.

The smaller the distance, the more the boundaries are detectable, so that its predominat feature becomes the *minimum distance* between the letters.

Distance of observation may be microtypographically defined by the pair: *Text* & *Display*.

The *context* space may also be converted to a *microtypographical* space where the axes may be: *tight-loose* and *rhythmic-textured*.

Rhythm, i.e. the *beat* of the white spaces while reading, is tightly connected to the concept of *equilibrium*: if the setting is large enough so that letters may not interfere one each other, *rhythm* and *equilibrium* would exactly be the same feature.

Texture refers to a well-ordered, compact sequence of well-recognizable objects.

But as tightness comes into play, reducing the *breathing* white space and triggering interactions between letters, things change:

to preserve *rhythm,* letters may collide;

to preserve *texture,* letters may compenetrate more.

The tighter the setting is, the more *readability* and *legibility* need to be sacrified to pass from an *ideal* to a *working* setting.

Eventually, even *kerning* may be a side effect of this process.

The tighter the setting is, the larger the amount of kerning is, being a consequence of the need for more *compenetration*. Compenetration is required to prevent that the glyphs may come too near one each other.

Conversely, the looser the setting is, the less kerning is required. If the distance between letters is sufficient, they do not interfere, they are not trying to enter the space of their neighbor.

So kerning may indirectly be a *measure* of the interaction.

The choice of a *context* is, in general, a *compromise*.

Fitting decisions are, more or less, the choice of a compromise that minimizes the damage.

Here, different points of view come into play. Everyone may have a different strategy.

In the end iKern is a tool that helps finding the perfect *equilibrium* as well as disrupting it in a controlled way.

My personal strategy is to consider all the possible pairs inside every possible triplet, quadruplet or larger sequence, and having them to work averagely the same in all the cases.

It is better to have good combinations throughout, even if sub-optimal, than some optimal combinations alongside others with visible flaws.

I call this strategy:

A font is not a logo

Operatively

In a font there are different groups of glyphs that require to share the same input parameters. For example: lowercase letters may form a group, uppercase letters another one, small caps yet another one, ...

Ikern uses different sets of input data for every group.

I talked about the engine and about the driving, but still haven't talk about the car. →

Glyphs can be recognized in different ways: by index, name, or Unicode codepoint. Index is not reliable because the glyph set may be re-sorted. A Unicode codepoint may not be assigned to every glyph. So iKern relies on *names*. But not the real names.

iKern uses *working names* that are mapped to real names using Unicode codepoints or the names themselves via dictionaries.

General dictionaries' entries can be overriden or added locally for every font or family of fonts.

Names are analyzed to create associations between glyphs that share the same name but not necessarily the same suffixes. Like in 'a' and 'a.smcp'. This way, information e.g. about base glyph, composition or type can be gathered from the Unicode database.

Pair lists are built using standard lists and expanding them to glyphs having a *suffix in the name* in an opportune and programmable way.

That's why it is important that:

- *Unicode codepoints* are correctly stored whenever possible;
- the use of *suffixes* in glyph names is consistent.

There's no need to set classes before letter-fitting a font.

Classes would be overriden. In iKern, classes are defined using databases and/or automatically.

Whether glyphs get included in a class or not is decided by superimposing outlines and, based on a variable that defines a tolerance, how much they deviate.

iKern produces 2 kinds of output data:

- a pair consisting of a TrueType font for the sidebearing values and an AFDKO compliant 'kern' feature including class definitions.
- a proprietary format that contains the usual data and also the variation of sidebearing for every glyph.

These data can be imported into new fonts. There are different ways to assign sidebering values: directly, preserving the width, using variations, ... So it is possible, for example:

- to make *grades*. Every font is spaced but only the median one kerned. New width values from the median font are imported and glyphs centered. The same kerning is imported to all. This way there won't be text reflow.
- to work on *layered fonts*. The variation of sidebearing and width values coming from one of the layers are imported into all other fonts, thus preserving the relative position.

iKern output data can be imported into .vfb or UFO fonts.

Testing is done on hi-res displays, looking at test words. For a final test, a *slide show* system shows words taken from database files.

Words can be displayed in random order.

A temporary TrueType font can be installed on the system during the work for testing it in an external application. Using the *suffix in the name* system, it is also possible to map glyphs differently. For example: small caps in place of the lowercase letters.

There's an option to modify the way that outlines are analyzed after discretization, rotating the system of reference. It's used to handle *rotalic fonts*.

That's all.