Meta-designing parameterized Arabic fonts for AlQalam

Ameer M. Sherif, Hossam A. H. Fahmy
Electronics and Communications Department
Faculty of Engineering, Cairo University, Egypt
ameer dot sherif (at) gmail dot com, hfahmy (at) arith dot stanford dot edu
http://arith.stanford.edu/~hfahmy

Abstract

In this paper we discuss how parameterized Arabic letters are meta-designed using METAFONT and then used to form words. Parameterized Arabic fonts enable greater flexibility in joining glyphs together and rendering words with imperceptible junctions and smoother letter extensions. This work aims to produce written Arabic with quality close to that of calligraphers. Words produced using our parameterized font are compared to other widely used fonts in a subjective test and results are presented.

1 Introduction

The Arabic script is used for a multitude of languages and is the second most widely used script in the world. However, due to the inherent complexity [3, 6] of producing high quality fonts and typesetting engines, the support for Arabic digital typography has been very weak.

OpenType is currently the de facto standard font technology. It has many features to support a wide variety of scripts, yet has its limitations for Arabic [7]. The most significant limitations are probably the following two.

1. The concept of letter boxes connecting together via other boxes of elongation strokes is not suitable for highest quality Arabic typesetting. When connecting glyphs to one another, the junctions rarely fit perfectly because adjacent letter glyphs usually have different stroke directions at the starting and ending points.
2. The use of pre-stored glyphs for different ligatures is limiting. The number of possible ligatures is far greater than what can be made available.

In order to achieve an output quality close to that of Arabic calligraphers, we modeled [7] the pen nib and its movement to draw curves using METAFONT. In this paper, we use the pen stroke macros that we have defined to meta-design the primitive glyphs needed for a good quality Arabic font. So far, we are working with the Naskh writing style and we provide a fully dynamic and flexible design leading to smooth junctions between letters. We also developed a simple algorithm to perform kerning in the case of letters that do not connect to what follows them. According to a survey we conducted, our design surpasses the widely used fonts.

Our work is not yet finished. In the future, we need to provide for the automatic placement of dots and diacritic marks and complete the rest of the required shapes.

2 Strokes in Arabic glyphs

The Arabic alphabet, although consisting of 28 different letters, depends on only 17 different skeletons. The dots added above or below some of these skeletons are the means of differentiating one letter from another. For example the letters *ǧīm* (ج) and *ḫā'* (خ) have the same shape as the letter *ḥā'* (ح), but *ǧīm* has a dot below, and *ḫā'* has a dot above. When we discuss a primitive we mention its use in the group of letters having the same skeleton, not individual letters, and this further simplifies our designs.

2.1 The Arabic measurement unit

Over a thousand years ago, Ibn-Muqlah, one of the early theorists of Arabic calligraphy, was probably the first to make the choice of the *nuqṭā* (Arabic for dot) as a measurement unit for letter forms [3]. He chose it in order to have some fixed measurements between different letter forms. For example, in the Naskh writing style, the height of *'alif* is 4 *nuqṭās*, and the width of an isolated *nūn* is 3 *nuqṭās*. The *nuqṭā* or dot we refer to is that made by the pen used to write the letter, i.e., it is not a constant like the *pt*. The horizontal width of the *nuqṭā* in Naskh (where the pen is held at an inclination of 70 degrees to the horizontal) is approximately equal to the diagonal of a dot drawn by the pen as shown in Fig. 1. Since the dot is a square then the *nuqṭā* width is slightly less than the pen width multiplied by the square root of 2. In our work we take it as $1.4 \times$ *pen width* and in our METAFONT programs it is simply abbreviated as `n`.

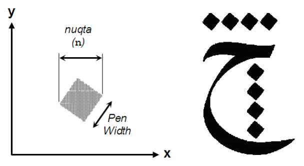


Figure 1: The *nuqta* as a measurement unit.

2.2 Stroke point selection

We define a calligrapher's pen stroke as a continuous movement of the pen. The location where the calligrapher pauses defines the end of a stroke and the start of a new one. Thus, a circular path may be considered as only one stroke because the start and end points are defined by the movement of the hand and not by the appearance.

The first step in the process of meta-designing any primitive or letter is to select the points through which the pen strokes pass. This is not an easy choice. When designing outline fonts, the solution is usually to scan a handwritten letterform, digitize its outline, then make the necessary modifications. We did not adopt this approach because Arabic letters do not have fixed forms but rather depend on the calligrapher's style. Since we are *meta*-designing, we are more concerned with how the letter is drawn and not just a single resulting shape. Hence, instead of capturing the fine details of a specific instance of the letter by one calligrapher, we wanted to capture the general features of the letter. To help us accomplish this, we based our design on the works of multiple calligraphers.

The letter *'alif* is shown in Fig. 2 with three different possibilities of point selection. The leftmost glyph requires the explicit specification of the tangential angles at points 1 and 2. In the middle glyph, just connecting the points 1–3–4–2 with a Bézier curve can produce the same curve without explicitly specifying any angles: `z1..z3..z4..z2`.

Theoretically, we can specify the path using an infinite number of points, but the fewer the points, the better the design and the easier to parameterize it. Adding more points that also lie on the same path can be done as in the rightmost glyph, but point 5 is redundant because the stroke is symmetric, and can be produced without explicitly specifying any angles or tension.

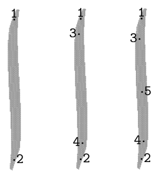


Figure 2: Selecting points to define the path of the letter *'alif*.

This *'alif* example shows that the minimum number of points to choose for any stroke is two, and their locations are at the endpoints of the stroke. These are the easiest points to select. Intermediate points are then chosen when curve parameters such as starting and ending directions and tensions are not enough to define the curve as needed for capturing important letter features. Hence more points are usually needed in stroke segments with sharp bends or in asymmetric strokes.

2.3 Stroke point dependencies

In our design, we model the direction of the stroke as it is drawn by the calligrapher, i.e., the stroke of the letter *'alif* is drawn from top to bottom. The points in our designs are numbered in order according to the pen direction. So for the letter *'alif*, the stroke begins at point 1 and ends at point 2.

However, a calligrapher chooses the starting point of the *'alif* stroke depending on the location of the base line. This means that point 1 is chosen relative to point 2, so we define 1 based on 2. Since METAFONT is a declarative language, not an imperative one, the two statements: `z1 = z2 + 3;` and `z2 = z1 - 3;` evaluate exactly the same. Yet we try to make the dependencies propagate in the natural logical order, which then makes editing the METAFONT glyph code an easier job; hence, the first expression is the better choice.

3 Meta-designing Arabic letters

Several characteristics of the letter shapes discovered during our design process were not mentioned explicitly in most calligraphy books. Calligraphers do not measure their strokes with precise rulers and their descriptions are only approximate. Detailed features of the letters are embedded implicitly in their curves as they learned them by practice. However, in our design, we represent the stroke mathematically and require accurate descriptions. The following sections show a couple of examples.

We start by studying the letter shapes and noting the fine variations that might exist between 'similar' shapes. Then we select the stroke points, decide on the pen direction at each point based on the letter shape and stroke thickness, and finally draw the strokes using our `qstroke` macro [7].

3.1 The concept of primitives

Meta-design enables us to break the forms of glyphs into smaller parts, which are refered to as primitives. These primitives may be whole letters or just parts of letters that exist exactly as they are or with small modifications in other letters. We can save design time and increase meta-ness by reusing primitives.

In Knuth's work on his Computer Modern (CM) fonts [4], primitives were not explicitly defined as black boxes and then reused. The use of primitives requires more parameterization than what CM deals with. There, the use of primitives was worthy only in small and limited flexibility shapes such as serifs and arcs, for which Knuth wrote subroutines. The difference between his work and ours is that he parameterized letters to get a large variety of fonts, while we parameterize primitives to make the letters more flexible and better connected, not to produce different fonts. Indeed, as mentioned earlier, our current focus is the Naskh writing style only.

Our classification of Arabic primitives consists of three categories:

- **Type-1 primitives** are used in many letters without any modifications.
- **Type-2 primitives** are dynamic and change shape slightly in different letters.
- **Type-3 primitives** are also dynamic but much more flexible.

3.2 Type-1 primitives

This category includes diacritics and pen strokes common in many letters. The *nuqṭā*, *kāf*'s *shāra*, and the *hamza* are Type-1 primitives. Fig. 3 shows the *shāra* of the letter *kāf* and the *hamza*. These glyphs are drawn using a pen with half the width of the regular pen. Other diacritics like the short vowels *fatḥa* and *kasra* are dynamic, and do change length and inclination angle.

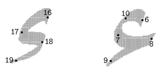


Figure 3: The *shāra* of *kāf* and the *hamza*.

The 'tail' primitive is another example of a Type-1 primitive. It is used as the ending tail in letters like *wāw*, *rā'*, and *zāy* in both their isolated and ending forms. Fig. 4 shows the tail designed using METAFONT on the left and its use in *wāw* and *rā'* on the right.

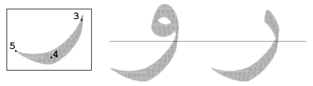


Figure 4: The tail primitive.

Even in cases with kerning where the tail may collide with deep letters that follow it, many calligraphers raise the letter as a whole without modifying the tail's shape. Fig. 5 shows an example of kerning applied to letters with tails. We follow the same approach in our design and the tail requires no flexibility parameters.

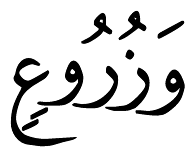


Figure 5: Four consecutive tails in a word as written in the Qur'an [1], [26:148]. Notice the identical tails despite the different vertical positioning.

Listing 1 shows the code for the tail. The natural direction of drawing is from point 3 to 4 to 5. But in fact, the stroke is only between 3 and 4; the last part of the tail is called a *shāẓya* and calligraphers usually outline it using the tip of the pen nib then fill it in. We use the METAFONT `filldraw` macro for that purpose. We use `filldraw` instead of `fill` in order to give thickness to the *shāẓya* edge at point 5. Also note the coordinate points dependencies: `z4` depends on `z3` and `z5` depends on `z4` not `z3`. This makes modification of the glyph much easier by separating the definition of the stroke segment and that of the *shāẓya*, i.e., if we modify the stroke, the *shāẓya* is not affected, unlike if `z5` was a function of `z3`.

```
z4 = z3 + (−1.7n, −2n);
z5 = z4 + (−2n, .36n);
path raa_body;
raa_body = z3{dir −95} .. tension 1.3 ..
    z4{dir −160};
qstroke(raa_body, 85, 100, 0, 0);
path shathya;
shathya = (x4, top y4){dir −160} ..
    {dir 160}z5{dir −38} .. tension 1.3 ..
    (rt bot z4){dir 15} -- cycle;
pickup pencircle scaled 1.2;
filldraw shathya;
```

Listing 1: METAFONT code for the tail primitive.

The code first defines the points in relation to each other using the *nuqṭā* (`n`) as the unit of measurement. Then a path variable is created which holds the path definition of the stroked part of the tail. The `qstroke` macro [7] is used to draw the stroke. The second path defines the *shāẓya* outline.

3.3 Type-2 primitives

This class of primitives has few parameters that enable only slight variations in the primitive's shape to facilitate its use in different letters. We will discuss two primitives of this type: the '*wāw* head' and the '*'alif* stem'.

3.3.1 The *wāw* head primitive

This primitive is a circular glyph used in the starting and isolated forms of the letters *wāw*, *fā'*, and *qāf*. It consists of two parts: the head and the neck. In most calligraphy books, the head is described as being exactly the same in all three letters. However small differences exist between the heads due to the connections with different letter skeletons. Fig. 6 shows the letters *fā'* and *qāf* as drawn in three books. Note how the circular head does in fact look slightly different in both letters, yet none of these books mention that there are variations in the head.

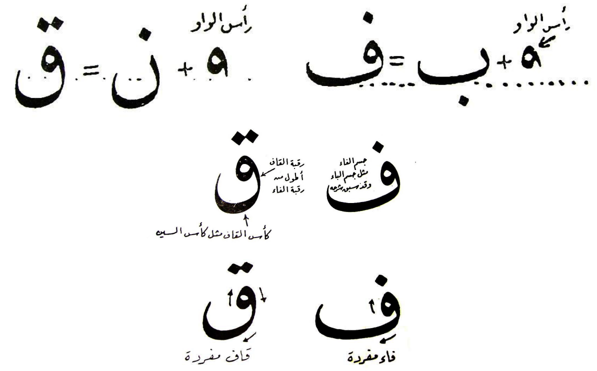

رأس الواو
رأس الواو
رقبة القاف أطول من رقبة الفاء
جسم الفاء مثل جسم الباء وقد سبق شرحه
كأس القاف مثل كأس السين
قاف مفردة
فاء مفردة

Figure 6: The letters *fā'* and *qāf* as drawn by three calligraphers, from top to bottom: Afify [2], Mahmoud [5], and Zayed [8].

Fig. 7 shows that the *wāw* head primitive consists of 2 strokes, one between points 1–2, the other between points 2–3–4. We approximate the differences between the *wāw*, *fā'*, and *qāf* by altering the 3–4 segment. Thus the same primitive may be used for the three letters in their isolated form. This same primitive is used in their ending forms by moving point 1 down and to the right, to connect to a preceding letter or *kashīda*.

3.3.2 The *'alif* stem primitive

The stem of the *'alif* (Fig. 2) is used in many letters: *lām* (all forms), *kāf* (isolated and final forms), *mīm*

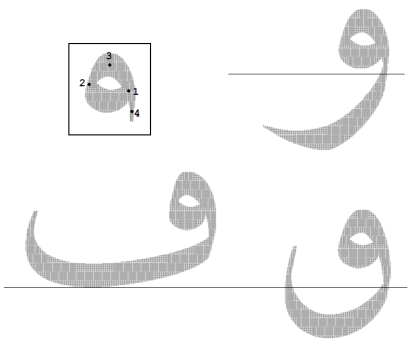


Figure 7: The *wāw* head primitive.

(final form), and *ṭā'* (all forms). Most calligraphers describe the straight stroke in the *lām*, *kāf*, and *ṭā'* as being identical to the *'alif*. Fig. 8 shows a calligrapher's description [5] stating that the form of the vertical stroke in the different letters is exactly the same as the *'alif*. This is a crude approximation because there are differences in the thickness, curvature, inclination, and height (in case of the *ṭā'*) between the isolated *'alif* and the modified form used in other letters.

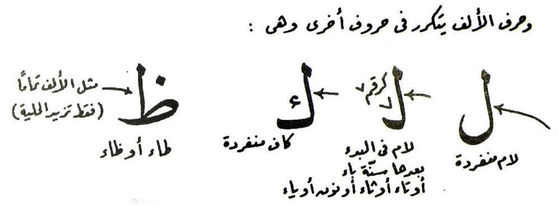

وحرف الألف يتكرر فى حروف أخرى وهى :
مثل الألف تماماً (فقط تزيد الحلية)
طاء أو ظاء
كاف منفردة
لام فى البدء بعدها سنة باء أو تاء أو ثاء أو نون أو ياء
لام منفردة

Figure 8: Approximate directions in calligraphy books.

Fig. 9 shows our design: on the far right the isolated *'alif* and to its left the modified *'alif* that is used in *lām* and *kāf*. The modified *'alif* is thinner with less curvature near the middle, or in other words more tension, together with more overall inclination. Listing 2 shows that they both have the same height and the thickness of the stem is achieved by increasing the pen nib angle.

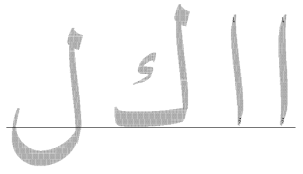


Figure 9: The *'alif* primitve.

```
% Description for isolated 'alif
curve := −100; incline := 70; height := 4.5n;
z2 = z1 + (0, −height);
path saaq;
saaq = z1{dir curve} .. tension 1.4 ..
    z2{dir curve};
qstroke(saaq, incline, incline, 0, 0);

% Description for 'alif used in lam and kaaf
curve := −95; incline := 79; height := 4.5n;
z2 = z1 + (0.3n, −height);
path saaq;
saaq = z1{dir curve} .. tension 1.4 ..
    z2{dir curve};
qstroke(saaq, incline, incline − 3, 0, 0);
```

Listing 2: METAFONT code for the *'alif* primitive.

3.4 Type-3 primitives

Type-3 primitives are glyphs that have a wider dynamic range, and greater flexibility. In this section, we discuss the skeleton of the letter *nūn*, called the *kasa* (Arabic for cup) and the *kashīda*. Calligraphers often use the great flexibility of these primitives to justify lines.

3.4.1 Kasa primitive

The body of the letter *nūn* is used in the isolated and ending forms of *sīn*, *šīn*, *ṣād*, *ḍād*, *lām*, *qāf*, and *yā'*. Fig. 10 shows the *kasa* in five letters. The *kasa* has two forms, short and extended. The short form is almost 3 *nuqṭās* in width in the case of *nūn*, one *nuqṭā* longer in *yā'*, and slightly shorter in *lām*. This difference between the *kasa* of the *lām* and the *nūn* is not well documented in calligraphy books, where most calligraphers mention that both are the same and only few state that in the *lām* it is slightly smaller.

An important property of the *kasa* is that it can be extended to much larger widths. In its extended form, it can range from 9–13 *nuqṭās*. Fig. 11 shows the short form together with three instances of the longer form generated from the same METAFONT code. Note that its width can take any value between 9 and 13, not just integer values, depending on line justification requirements. Also note how the starting *senn* (vertical stroke to the right) of the letter is shorter in extended forms.

3.4.2 Kashida primitive

Another very important primitive for justification, the *kashīda* is used in almost all connected letters. As an illustrative example, Fig. 12 shows the letter *ḥā'* in its initial form with two different *kashīda* lengths, differing by 3 *nuqṭās*. The parameter `tatwil` controls this length by varying the distance between points 3 and 4, both the horizontal and vertical components, as shown in this line of code:

$$z_3 = z_4 + (1.74n, 0.116n) + (0.5\mathit{tatwil}, 0.025\mathit{tatwil}) * n;$$

As `tatwil` increases, point 3 moves further from point 4 both to the right and upward. This vertical change helps maintain the curvature in the *kashīda*. If no vertical adjustment is made, longer *kashīdas* look like separate straight lines with a sharp corner at their intersection with the surrounding letters. Calligraphers, on the other hand, draw curved lines rather than straight ones producing aesthetically better shapes. For these reasons, in our definition of the stroke, the tangential direction at point 3 is left free depending on the distance between 3 and 4. We will see in the next section how *kashīdas* are adjusted to join letters together smoothly.

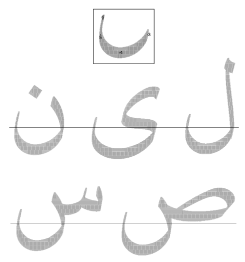


Figure 10: The *kasa* primitive.

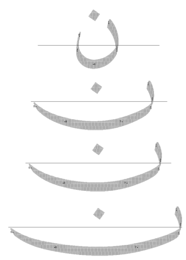


Figure 11: The letter *nūn* shown with *kasa* widths of 3, 9, 10 and 13 *nuqṭās*.

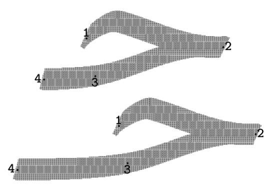


Figure 12: The initial form of the letter *ḥā'* with two different *kashīda* lengths.

4 Forming words

The combination of primitives to form larger entities is the final step towards producing Arabic script that is as cursively connected and flexible as calligraphers' writings. The parameterization of the glyphs allows us to piece them together perfectly as if they were drawn with just one continuous stroke.

4.1 Joining glyphs with *kashīdas*

The most widely used glyph to connect other letters is the *kashīda*. In this section we will explain the mechanism we use in order to make the junction between letters as smooth as possible. In current font standards, such as OpenType and TrueType, *kashīdas* are made into fixed glyphs with pre-defined lengths, and are substituted when needed between letters to give the feeling of extending the letter. But because that design for the *kashīda* is static, as are the rest of the surrounding letters, they rarely join well. It is evident that the word produced is made of different segments joined by merely placing them close to each other.

In our work, the *kashīda* is dynamic and can take continuous values, not just predefined or discrete values. We believe that when a *kashīda* is extended between any two letters, it does not belong to just one of them; instead, it is a connection between them both. This belief is the result of experimenting with different joining methods.

Let us take the problem of joining the two letters *ḥā'* and *dāl* as an example to illustrate the *kashīda* joining mechanism we have developed. The solution we propose is to pass the `tatwil` parameter to the macros producing the two glyphs, and the *kashīda* length is distributed between both glyphs. This enables us to fix the ends of the glyphs to be joined at one angle, which is along the x-axis in the Naskh style, since any *kashīda* in that style must at one point move in this direction before going up again. To accommodate long *kashīdas*, each glyph ending point is moved further from its letter and slightly downwards. Long *kashīdas* need more vertical space in order to curve smoothly, sometimes pushing the letters of a word upwards.

Other than affecting the ending points, the parameter also affects the curve definition on both sides by varying the tensions, while keeping the direction of the curves at the intersection along the negative x-axis (since the stroke is going from right to left). The resulting word at many different *kashīda* lengths is shown in Fig. 13.

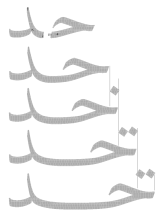


Figure 13: Placing a *kashīda* between the letters *ḥā'* and *dāl* with different lengths: 2, 3, 5 and 7 nuqtas.

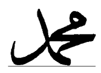


Figure 14: The word *Mohammed* as an example of vertical placement (Thuluth writing style).

4.2 Vertical placement of glyphs

In written Arabic, the existence of some letter combinations may force the starting letter of a word to be shifted upwards in order to accommodate for the ending letters to lay on the baseline of the writing. A very simple example of that property is the name *Mohammed* when written with ligatures, where the initial letter *mīm* is written well above the baseline, as shown in Fig. 14.

It is hence obvious that the starting letter's vertical position is dependent on the word as a whole. It might then be thought that it is easier to draw the words starting from the left at the baseline and then move upwards while proceeding to the right. But this has two problems: one is that the horizontal positioning of the last letter depends on the position of the first letter on the right and on the length of the word, and the second is that a left to right drawing would be against the natural direction of writing and may result in an unnatural appearance.

The solution is then to walk through the word till its end and analyze each letter to know where to position the beginning letter vertically, and then start the actual writing at the right from that point going left. This process is what a calligrapher actually does before starting to write a word. So, for a combination of letters, we benefit from the declarative nature of METAFONT. The following rules are applied:

- The horizontal positioning starts from the right,
- the vertical positioning starts from the left at the baseline, and
- writing starts from the right.

To illustrate this better, see Fig. 15. The ligature containing the letters *sīn*, *ǧīm*, and *wāw* is traced from left-to-right as shown, going through points 1–2–3, the starting points of each glyph, until

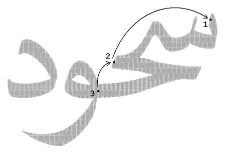


Figure 15: Tracing a word from left-to-right to know the starting vertical position.

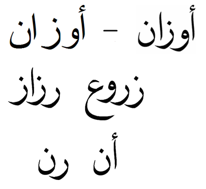


Figure 16: Kerning: letters with 'tails' need special attention. The first line shows our output to the right and the current font technologies to the left. The last two lines show our output.

the vertical position of point 1 is known. The next step is to start writing the word starting from point 1. Isolated letters like *dāl* are not taken into consideration, because they do not affect the preceding letters vertically.

4.3 Dealing with kerning

To imitate what calligraphers do as in Fig. 5, we had to invent a special kerning algorithm. Fig. 16 shows our output. We raise the level of any 'deep letter' following a 'tail letter' to kern correctly. Any non-deep letter holds the level steady. The end of the word restores the baseline.

4.4 Word lengths

In reality, word lengths are not selected by the calligraphers word by word, but instead, are chosen based on the justification requirements of a whole line. When a word length is decided according to the line it exists in, this length should be passed to a main macro that calls the glyph macros in order to form the word. This main macro should decide the length of *kashīdas* to be added depending on the minimum length of each letter. For example, that the word under consideration is the one shown in Fig. 13, and that the desired total length of the word is 10 *nuqṭās*. In order to calculate the extension or the `tatwil` parameter between the letters, it subtracts all the minimum lengths of the individual letters. In our example, the head of the *ḥā'* is 4 *nuqṭās* wide, and the base of the *dāl* is 3 *nuqṭās*, hence the word has a minimum length of only 7 *nuqṭās*. In order to stretch it to 10, the added *kashīda* is 3 *nuqṭās* wide.

4.5 A final example

This section describes a more illustrative example shown in Fig. 17. This example, showing four instances of the word *sujud*, demonstrates the many properties and benefits of our parameterized font. First, it shows flexibility in stretching and compressing words for line justification purposes. This flexibility is due to two capabilities of the font: dynamic length *kashīdas* and glyph substitution. For a very small line spacing, the *sīn* is written on top of the *ḥā'*, and the *kashīda* after the *ḥā'* is almost zero.

When more space is available, the *kashīda* after the *ḥā'* is stretched and the *senn* connecting the *sīn* and the *ḥā'* is also made slightly longer. Further elongation is made possible by breaking the ligature between *sīn* and *ḥā'*. And finally, the maximum length is obtained by elongating the *kashīda* between the two letters. Theoretically, we could get more stretching of this *kashīda* and even add another one after the *ḥā'*, but calligraphic rules ultimately limit the stretching.

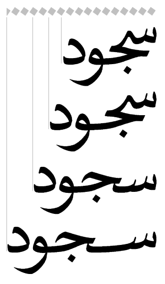


Figure 17: The word *sujud* written with different lengths ranging from 10.5 to 16.5 nuqtas.

5 Testing the output

5.1 A GUI to simplify tests

To facilitate the task of testing our ideas, we made a simple graphical user interface (GUI) to write words using our METAFONT programs. This allows us to make changes to parameters and draw words faster than having to edit the code manually. Fig. 18 shows the block diagram describing the operation of the GUI and Fig. 19 shows the different window components of the GUI.

First, the user types in a word or sequence of words in the input word text box, then presses the start button. This parses the input word(s) into a string of letters, removing any space characters. Each character can be selected from a list and its shape determined from the letterform list at the bottom of the screen. Also a length extension can be input in the length text box (default is zero).

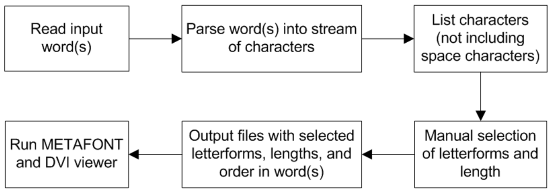

Read input word(s)
Parse word(s) into stream of characters
List characters (not including space characters)
Manual selection of letterforms and length
Output files with selected letterforms, lengths, and order in word(s)
Run METAFONT and DVI viewer

Figure 18: Block diagram describing the operations of the graphical user interface.

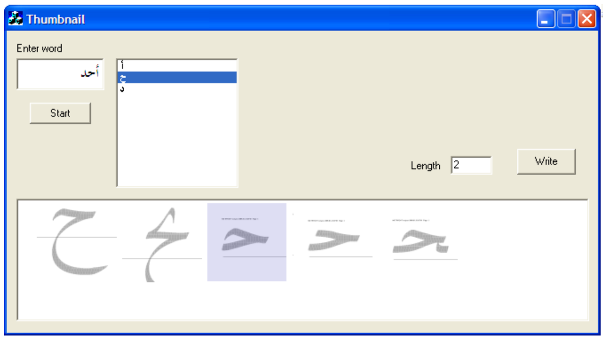


Figure 19: Screenshot of our simple GUI.

When the write button is pressed, the letterforms selected and their extra lengths are written in files, then METAFONT executes the glyph programs. Finally, a DVI previewer opens the resulting output as seen in Fig. 20.

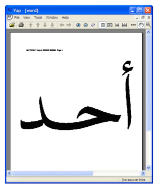


Figure 20: Screenshot of the DVI previewer displaying the output word.

5.2 The survey

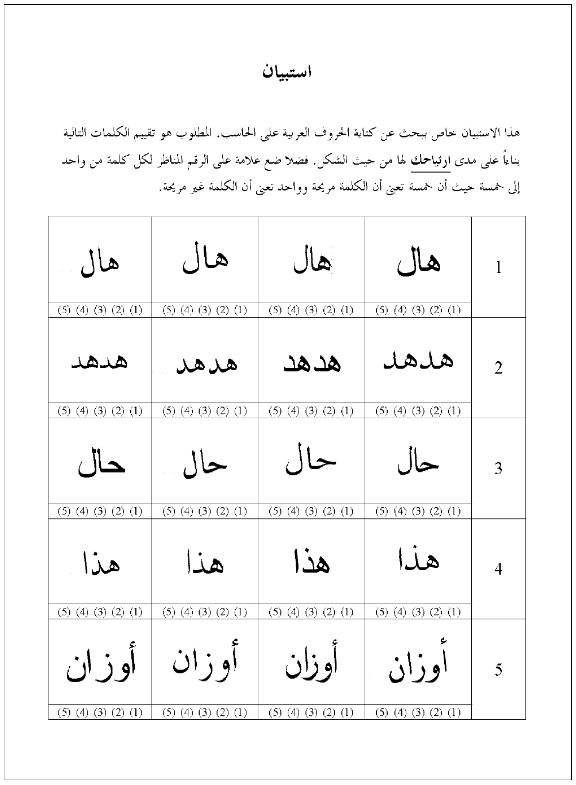
In order to test if readers are comfortable with the way our parameterized font looks as compared to other Naskh fonts, we made a list of 16 words, each written in four different Naskh fonts:

- Simplified Arabic,
- Traditional Arabic,
- DecoType Naskh, and
- AlQalam parameterized font.

For a more reliable and unbiased test, the order of fonts used is varied in consecutive rows and all words are set to approximately the same sizes although nominal point sizes of the different fonts are not exactly equivalent. We selected the 16 words to test three main features:

- connections between letters,
- extension/tatwil of letters, and
- kerning.

Fig. 21 shows the first page of the test.



استبيان

هذا الاستبيان خاص ببحث عن كتابة الحروف العربية على الحاسب. المطلوب هو تقييم الكلمات التالية بناءاً على مدى **ارتياحك** لها من حيث الشكل. فضلا ضع علامة على الرقم المناظر لكل كلمة من واحد إلى خمسة حيث أن خمسة تعني أن الكلمة مريحة وواحد تعني أن الكلمة غير مريحة.

هال (5) (4) (3) (2) (1)	هال (5) (4) (3) (2) (1)	هال (5) (4) (3) (2) (1)	هال (5) (4) (3) (2) (1)	1
هدهد (5) (4) (3) (2) (1)	هدهد (5) (4) (3) (2) (1)	هدهد (5) (4) (3) (2) (1)	هدهد (5) (4) (3) (2) (1)	2
حال (5) (4) (3) (2) (1)	حال (5) (4) (3) (2) (1)	حال (5) (4) (3) (2) (1)	حال (5) (4) (3) (2) (1)	3
هذا (5) (4) (3) (2) (1)	هذا (5) (4) (3) (2) (1)	هذا (5) (4) (3) (2) (1)	هذا (5) (4) (3) (2) (1)	4
أوزان (5) (4) (3) (2) (1)	أوزان (5) (4) (3) (2) (1)	أوزان (5) (4) (3) (2) (1)	أوزان (5) (4) (3) (2) (1)	5

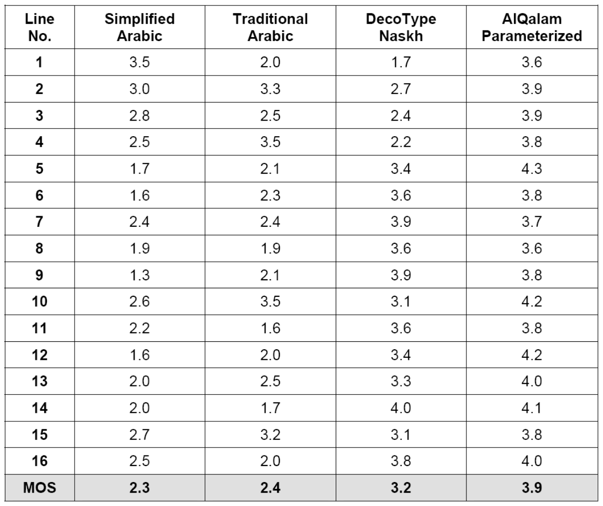
Figure 21: First page of the test where we ask the readers to rate their comfort.

We then surveyed 29 Arabic readers (14 males and 15 females) with ages ranging from 10 to 70 years. In the survey, readers were asked to evaluate and rate words in terms of their written quality and how comfortable they feel reading the words. A rating of 1 is given to low quality and uncomfortable words, and a rating of 5 is given to words of high written quality and most comfort to the reader.

This test methodology is known as the Mean Opinion Score (MOS), a subjective test often used to evaluate the perceived quality of digital media after compression or transmission in the field of digital signal processing. The MOS is expressed as a single number in the range of 1 to 5, where 1 is lowest quality and 5 is highest quality. In our case, the MOS is a numerical indication of the perceived quality of a written word. Finally, the total MOS is calculated as the arithmetic mean of all the individual scores.

Table 1 shows the MOS scores for individual words of each font and the total average.

Table 1: MOS results for each font.



Line No.	Simplified Arabic	Traditional Arabic	DecoType Naskh	AlQalam Parameterized
1	3.5	2.0	1.7	3.6
2	3.0	3.3	2.7	3.9
3	2.8	2.5	2.4	3.9
4	2.5	3.5	2.2	3.8
5	1.7	2.1	3.4	4.3
6	1.6	2.3	3.6	3.8
7	2.4	2.4	3.9	3.7
8	1.9	1.9	3.6	3.6
9	1.3	2.1	3.9	3.8
10	2.6	3.5	3.1	4.2
11	2.2	1.6	3.6	3.8
12	1.6	2.0	3.4	4.2
13	2.0	2.5	3.3	4.0
14	2.0	1.7	4.0	4.1
15	2.7	3.2	3.1	3.8
16	2.5	2.0	3.8	4.0
MOS	**2.3**	**2.4**	**3.2**	**3.9**

The results clearly show an increased comfort with the parameterized font with respect to the other popularly used fonts. One feature clearly distinguishing the different fonts is kerning. This is very evident in words 1, 3, 5, 6, 9, 11, 13, and 14 which had kerning applied. The addition of smooth *kashīdas* for extension also results in a large difference in scores as with words 7, 10, and especially 12 which contains a long *kashīda*. The results of word 15 show that the use of complex ligatures is also an important feature for the comfort of readers. We believe that more work and collaboration with calligraphers can yield even better results.

6 Conclusion

The work covered in this paper is just a small step towards the realization of a system to produce output comparable to that created by Arabic calligraphers, and much more work is still needed. We need to complete our design and finish all the required shapes.

There is a strong need for non-engineering research on the readability and legibility of the different kinds of Arabic fonts comparable to the many studies conducted on Latin letters. It is important to categorize the different calligraphers' writing styles as well as the regular computer typefaces: are they easy and fast to read in long texts, or not? Which are better to use in titles or other 'isolated' materials, and which are better for the running text?

By presenting our effort, we hope to open the door for future exciting work from many researchers.

References

[1] *The Holy Qur'an.* King Fahd Complex for Printing the Holy Qur'an, Madinah, KSA, 1986.

[2] Fawzy Salem Afify. *ta'leem al-khatt al-'arabi [Teaching Arabic calligraphy].* Dar Ussama, Tanta, Egypt, 1998.

[3] Mohamed Jamal Eddine Benatia, Mohamed Elyaakoubi, and Azzeddine Lazrek. Arabic text justification. *TUGboat*, 27(2), January 2007.

[4] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting.* Addison-Wesley, Reading, MA, USA, 1986.

[5] Mahdy Elsayyed Mahmoud. *al-khatt al-'arabi, dirasah tafsiliya muwassa'a [Arabic calligraphy, a broad detailed study].* Maktabat al-Qur'an, Cairo, Egypt, 1995.

[6] Thomas Milo. Arabic script and typography: A brief historical overview. In John D. Berry, editor, *Language Culture Type: International Type Design in the Age of Unicode*, pages 112–127. Graphis, November 2002.

[7] Ameer M. Sherif and Hossam A. H. Fahmy. Parameterized Arabic font development for AlQalam. *TUGboat*, 29(1), January 2008.

[8] Ahmad Sabry Zayed. *ahdath al-turuq leta'leem al-khotot al-'arabiya [New methods for learning Arabic calligraphy].* Maktabat ibn-Sina, Cairo, Egypt, 1990.