Generative Type Design: Creating Glyphs from Typographical Skeletons

Fábio André Pereira
CISUC, Department of Informatics Engineering, University of Coimbra
Coimbra, Portugal
fasp@student.dei.uc.pt

Tiago Martins
CISUC, Department of Informatics Engineering, University of Coimbra
Coimbra, Portugal
tiagofm@dei.uc.pt

Sérgio Rebelo
CISUC, Department of Informatics Engineering, University of Coimbra
Coimbra, Portugal
srebelo@dei.uc.pt

João Bicker
CISUC, Department of Informatics Engineering, University of Coimbra
Coimbra, Portugal
bicker@dei.uc.pt

Figure 1: Glyphs generated with the presented system.

ABSTRACT

It is through typography that order and form, visible and durable, is given to written communication. Typography has been accompanied by new technologies, which designers and typographers have adapted. The use of these technologies in the design process boosted the exploration of new approaches for type design. Designers began to explore generative processes in their design process in different types of projects, such as dynamic visual identities or generative typography. In this work, we present an algorithmic system capable of generating glyphs from typographical skeletons. This system, which is online at cdv.dei.uc.pt/2019_/letterspecies, fills a typographical skeleton using a drawing technique, both selected by the user, and outputs an OpenType font. With the presented system, we explore the relation between legibility and coherence in an innovative type design approach where the generated typefaces can be applied in the most diverse media.

ARTECH 2019, October 23–25, 2019, Braga, Portugal

ACM ISBN 978-1-4503-7250-3/19/10…$15.00
https://doi.org/10.1145/3359852.3359866

CCS CONCEPTS

• Information systems → Multimedia information systems; Information systems applications; • Applied computing → Arts and humanities.

KEYWORDS

Generative design, Type design, Typographical skeleton, Typography

ACM Reference format:

F. Pereira, T. Martins, S. Rebelo, and J. Bicker. 2019. Generative Type Design: Creating Glyphs from Typographical Skeletons. In *Artech 2019, 9th International Conference on Digital and Interactive Arts, October 23–25, 2019, Braga, Portugal.* 8 pages. https://doi.org/10.1145/3359852.3359866

1 INTRODUCTION

Typography is one of the main components of graphic design. It is through typography that designers provide order and form,

visible and durable, to written communication. Typography makes frontier, on the one hand, with writing and, on the other hand, with graphic design. Writing is the domain of authors, in the sense that they are the ones who formulate ideas and transpose them into words. Making these words accessible and attractive to the reader is the domain of graphic designers, typographers and type designers. Type design is a process where the designer creates typefaces based on a set of characteristics that reflect a certain purpose [2,11].

The constant technological innovation over the last centuries has provided means and tools with enormous potential in the most diverse fields. Graphic design and type design are not exceptions. These tools accelerated the design process and facilitated the production and reproduction of typefaces as well as opened new horizons in creative exploration [11]. Consequently, algorithmic and generative processes, previously unsustainable and/or unthinkable in the context of graphic design, such as dynamic visual identities or generative typography, began to be explored. Through these processes, the designer is able to automate tasks and introduce new variables into the artefacts produced, such as randomness.

In this work, we present an algorithmic system capable of generating glyphs, or letterforms, by filling typographical skeletons using different drawing techniques. These typographical skeletons are automatically extracted from existing typefaces and consist of linear structures that represent the essential shape of each glyph. We employ different drawing techniques to fill and give shape to the typographical skeletons. Each drawing technique implements an algorithmic drawing process that allows us to achieve coherent glyphs with different visual styles. With the presented system, we are able to dynamically create glyphs with legibility, provided by the typographical skeletons, and expressiveness, provided by the drawing techniques (see, for example, Figure 1).

The main contributions created with this work include: *(i)* the development of an algorithmic system capable of generating glyphs that form a coherent typeface; *(ii)* the exploration of a new approach and, consequently, new creative possibilities in the field of generative type design; *(iii)* the proposal of a practical tool that enhances the laborious process of creating all the glyphs for a typeface; *(iv)* an investigation on how algorithmically generated typefaces impact legibility and readability; and *(v)* the generation of a series of different typefaces that can be used in diverse design applications, such as in the creation of logotypes, text compositions, or animated letterings.

The present paper is structured as follows. In the first section, introduction, we present the idea of this work and the resulting main contributions. In the second section, related work, we analyse a series of related works on generative processes for the creation of typefaces. In the third section, system, we describe the proposed system. In the fourth section, experiments, we describe some experiments conducted to test the application of the proposed system. In the fifth section, conclusion, we present our main conclusions on the developed work and future work.

2 RELATED WORK

Generative design processes have been used in many applications to explore new approaches to solving different problems. Thus, we analyse some generative type design approaches that informed our work in some way.

Elien is a monospace generative typeface created with a system developed by Tatevik Aghabayann, in Processing. It uses two-dimensional metaballs, *i. e.,* organic-looking shapes, containing up to five levels of circles arranged together and creating transitions when the distance between them is reduced. Thus, the metaballs are distributed by the skeleton of the letters, forming connections between the circles of the same level, and also between the circles of neighbouring letters. The size of the circles is set randomly. This system provides a set of parameters that influence the visual shape of the metaballs, allowing unlimited variations: (*i)* spacing, which influences the distance between letters and consequently the numbers of links between neighbouring letters when distance is too small; *(ii)* density, which influences the number of metaballs in the skeleton; (*iii)* contrast, which influences the size difference between the metaballs; and *(iv)* levels, which influences the number of circles in each metaball. The generated typeface contains 39 glyphs, including some punctuation marks [1].

Irratio is a generative typeface developed by Ingo Reinheimer, in Processing. Based on a simple non-serif typeface, this typeface is constructed with Bézier curves connecting successive anchor points, *i. e.,* the connection between the first and third points, the second and fourth, and so on, until all the points of the letter are connected [10].

Ntype is a web system developed by Kevin Zweerink to explore the visualization of the 4th dimension on extrusions of letterforms. It provides the user with parameters to control the extrusion visualization such as axes of rotation, speed of rotation, length of the track and the option to draw the shape, track, or both. It uses WebGL and Three.js to render the letterforms and opentype.js to export the extrusions as an OpenType Font (OTF) format [15, 16].

Jéssica Parente developed a system that explores the generation of typefaces from the combination of layers of several typefaces. This computational system consists of separating the different anatomical elements of the letters into layers. Then, these layers are modified and combined to form a new structure to which a drawing method that uses shapes such as circles, squares, and triangles is applied to generate new typefaces [7, 8].

Phase is a system conceptualised by Elias Hanzer and developed by Florian Zia that explores the concept of generative typefaces using variable font technology. This system consists of selecting a module and the subsequent real-time parametrisation of the associated characteristics. The system has the possibility of using sound as input, which provides numerous shapes for the glyphs, and also allows changing the parameters to random values. The resulting letterforms can be exported to a TrueType Font (TTF) [4].

Metaflop is a web application for custom typeface creation developed by Marco Müller and Alexis Reigel. It consists of

choosing one of the predefined typefaces and parametrising its anatomical characteristics using the Metafont language. The system gives access to a parameter scheme, as well as the glyph preview, a grid with all glyphs, and an area where it is possible to write text. Also, the system has the possibility to change the values of the parameters randomly, undo actions, and return to the initial state. The created typefaces can be exported to OTF or Web Open Font Format (WOFF) [9].

Prototypo is a web application for creating typefaces using a parametric font design process. It consists of choosing a typeface model and modifying its characteristics such as x-height, letter width, contrast, axis, and serifs. These characteristics are modified through a set of sliders and the resulting modifications on the glyphs are visualised in real-time. Subsequently, you can also edit the glyphs individually to add more detail or fix imperfections. Also, provides the ability to create various weights and the possibility to export the typeface created [6].

The analysis of the works described above allowed us to identify some aspects that we considered relevant and, consequently, informed the presented work. The parametrisation of the anatomical characteristics of letters gives the user the freedom to experiment with the creation of typefaces and explore their visual without knowing programming languages. Applying changes to typefaces and visualising the result in real-time makes the system more dynamic, appealing, and comprehensible to users, as they are able to associate each parameter to a specific visual aspect of the typeface.

Giving to the user the possibility to set the parameters to random values provides a quick way to test different combinations of parameters and this way achieve unexpected results. Allowing the user to export the typefaces he/she created makes the process more flexible and complete, as the user is able to use the typeface on any design or edit it using other software tools.

3 SYSTEM

We developed a computational system that allows the algorithmic creation of glyphs from typographical skeletons that are given shape using different generative drawing techniques. We use typographical skeletons extracted from existing typefaces to provide legibility to the generated glyphs. We then explore different drawing techniques to fill these skeletons and this way work the visual style of the glyphs.

The system is implemented in JavaScript and its architecture is designed to allow scalability. This way, more typographical skeletons and drawing techniques can be easily added later to the system. One can experiment with the presented system online at cdv.dei.uc.pt/2019/letterspecies. In the following subsections, we explain: *(i)* what typographical skeletons are and how they are extracted from existing typefaces; *(ii)* how the drawing techniques give form to the typographical skeleton in order to create legible glyphs; and *(iii)* how the output glyphs are visualised and exported.

3.1 Typographical Skeleton

The typographical skeleton of a glyph can be seen as its main structure. It retains essential typographical characteristics such the relation between the anatomical elements (*e. g.,* ascenders, descenders, diagonals, vertical and horizontal lines) and metrics (*e. g.,* x-height, baseline, total height, and width). In technical terms, a typographical skeleton consists of sequences of two-dimensional points defining the lines that are located in the centre of the glyph shape, hence the term skeleton.

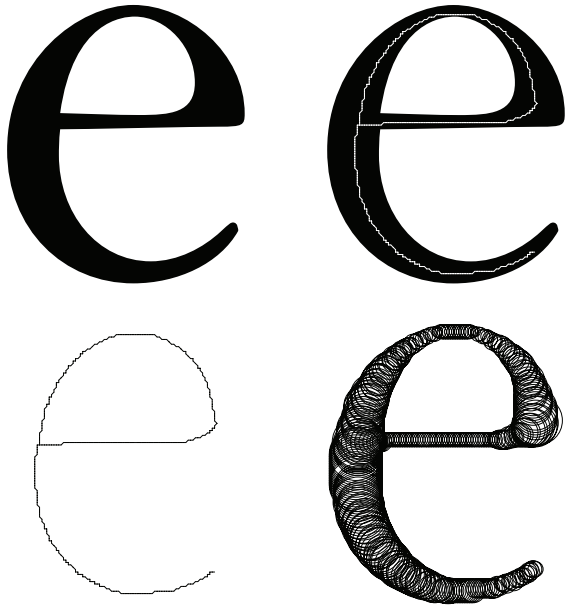

Figure 2: Extraction of the typographical skeleton from a glyph. From top left to bottom right: original glyph; selection of the skeleton points; skeleton points; and circles drawn at the skeleton points with radii given by their distance to the closest point outside the glyph shape.

The system uses a series of pre-calculated typographical skeletons as basis for drawing glyphs. These skeletons are extracted from existing typefaces using an algorithmic process based on the work by Parente et al. [7, 8]. Therefore, the Zhang-Suen thinning algorithm (1984) [14] is employed to extract the structural lines from a binary image by removing every point (pixel) unnecessary to the recognition of the shape in the image. Thus, for each glyph, a series of iterations are performed, wherein in each iteration, the farthest pixels of the shape is removed. When no more unnecessary pixels can be removed, the extraction process stops, resulting in a typographical skeleton. Through this process, we are able to achieve the x-coordinates and y-coordinates of the skeleton points, as well as the value corresponding to the radius of a circle associated with each point, which allows the replication of the overall original shape of the glyph (see Figure 2).

The extracted typographical skeletons are formed of a large number of points and the resulting curve contain lots of close points that are insignificant for recognising the structure of the glyph shape. Some of these points are redundant and thus unnecessary, for example, collinear points. Other points add noisy artefacts to the lines of the skeletons. In order to simplify and denoise the resulting typographical skeletons, we employ the Ramer-Douglas-Peucker algorithm (2003)[13]. This algorithm decomposes a given curve formed of line segments into a similar one with fewer points. Therefore, the simplified curve consists of the most significant points that defined the original curve. The degree of detail is controlled by a parameter, usually named epsilon, that defines the maximum distance between the original points and the simplified curve. By varying the epsilon value, we are able to achieve skeletons with different levels of detail (see Figure 3).

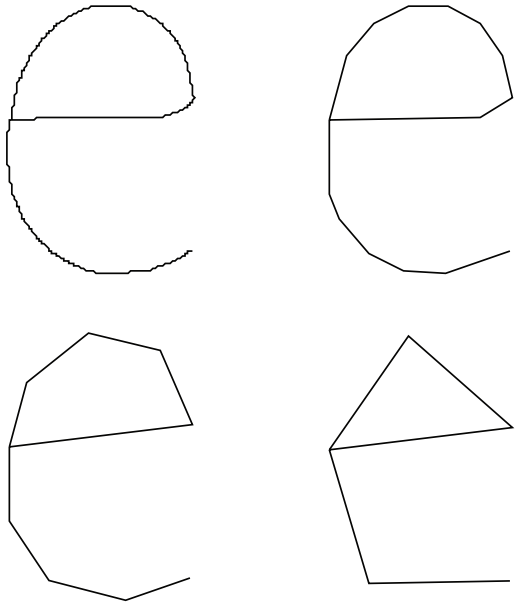

Figure 3: Typographical skeletons with different levels of detail. The epsilon value increases from top left to bottom right.

3.2 Drawing Techniques

The presented system creates glyphs by rendering the typographical skeletons described above using different drawing techniques. A drawing technique can be considered as a stroke style that gives shape to the lines that build the skeletons. Each drawing technique provides the skeletons, and thus the glyphs, with a unique visual style. Also, each drawing technique has its own set of parameters that allow the user to control different aspects of the drawing algorithm that implements to fill the skeletons. This way, the user can change these parameters to achieve different results with the same visual style.

In technical terms, each drawing technique consists of a programming method that draws a glyph given the following input arguments: *(i)* the data of the selected typographical skeleton (x-coordinates, y-coordinates and radii of the skeleton points); and *(ii)* the values of the parameters that control the inherent drawing algorithm.

We implemented three drawing techniques, which we describe below. Each one is based on a different visual concept and employs a combination of design processes and/or geometric shapes. As already mentioned, more drawing techniques can be easily added to the system.

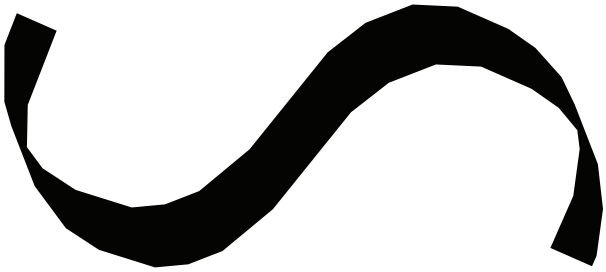

Figure 4: Demonstration of drawing technique D1.

The drawing technique D1 intends to simulate calligraphy (see Figure 4). To do so, we chose to analyse the broad nib pen to understand the main characteristics required to draw a calligraphic stroke. This pen is not flexible and so it has a fixed length. Also, the pen is usually held at an angle of 30 degrees. Therefore, we use these two main characteristics as the input arguments to create our programming method, along with the points of the selected typographical skeleton. In short, this method calculates for each point of the skeleton two extra points based on the angle and length values. Then, these extra points are used to form polygons. Finally, these polygons are joined together as a unique shape. In this drawing technique, the user can control two parameters: stroke angle and stroke length.

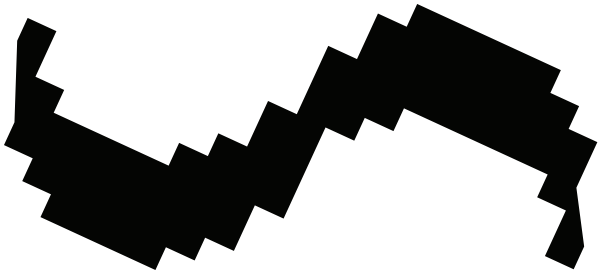

Figure 5: Demonstration of drawing technique D2.

The drawing technique D2 relates to digital media by creating pixelated shapes (see Figure 5). We explore the visual concept of

pixel because it represents well digital environments. The implementation of this drawing technique is based on D1, adding a feature that reduces the detail of the resulting stroke to achieve the pixelated effect. Therefore, in this drawing technique, the user can control three parameters: stroke angle, stroke length and pixel density.

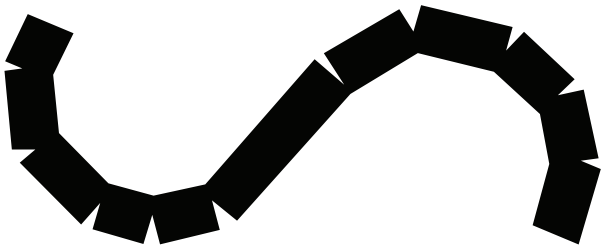

Figure 6: Demonstration of drawing technique D3.

The drawing technique D3 consists of employs basic shapes such as the rectangle and the triangle to achieve a geometric style (see Figure 6). Although the selection of the shape is given to the user as a parameter, in this variation of the style, we use the rectangle to create glyphs. This drawing technique draws a rectangle for every two consecutive points of the typographical skeleton. Thus, for this style, the user can control two parameters: shape to be used and shape width.

3.3 Output Glyphs

The system generates and renders the output glyphs in real-time. This feature provides a fluid interaction by enabling the user to manipulate the different parameters and instantly visualise the output glyphs. The user can visualise the output of the system through an editable text composed with the resulting typeface. This fluid interaction also enables instant visual feedback on the impact of each parameter on the design of the glyphs. Consequently, the user quickly gains control of the system and becomes capable of pursuing a given concept for a typeface.

The glyphs created with the system are represented and visualised using Scalable Vector Graphics (SVG). A typeface formed by all these glyphs can be exported to OTF file. This is accomplished through the opentype.js library. Working with these two output formats enables the user, for example, to make further refinements to the resulting glyphs/typeface. An SVG file can be easily editable using a vector editing program and an OTF file can be used on any computer and manipulated using a typeface design program.

4 EXPERIMENTS

We tested the presented system with the following major goals in mind: *(i)* assess the ability of each drawing technique, when applied to different typographical skeletons, to create legible and expressive glyphs; *(ii)* analyse the impact of each drawing technique parametrisation on the glyphs; and *(iii)* examine the visual diversity among the glyphs created with different combinations of typographical skeletons and drawing techniques.

In this work, we create glyphs for the Latin alphabet. As for the typographical skeletons, we used two open-source typefaces from the Google Fonts library: IBM Plex Sans and IBM Plex Serif (see Figure 7). The first one is a sans serif typeface, while the second is a serif typeface. To give shape to the two skeletons extracted from these typefaces, we use the three drawing techniques D1, D2 and D3, which were described in the previous section. Each drawing technique is tested with two parametrisations. In what concerns the visualisation of the results, although the system automatically generates OTF typefaces with most Latin glyphs, in this paper we only present a subset of glyphs to compose the word “glyph”.

Figure 7: Original glyphs (top) and their typographical skeletons with maximum detail (bottom) of the typefaces IBM Plex Sans (left) and IBM Plex Serif (right).

For the two different sets of parameters tested in D1, we explored two parameterisation values for the stroke angle. In one parameterisation, we used an approximately vertical angle on a typographical skeleton with a low level of detail (top glyphs in Figure 8). In the other, we used an angle of approximately 45 degrees on a typographical skeleton with a high level of detail (bottom glyphs in Figure 8).

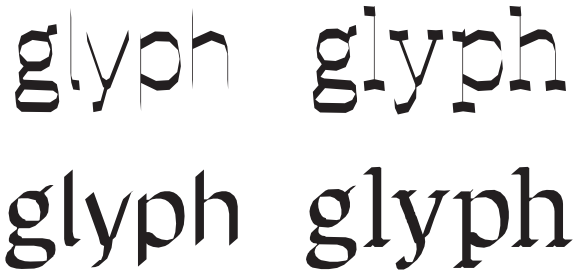

Figure 8: Glyphs generated with the presented system using the typographical skeletons of the typefaces IBM Plex Sans (left) and IBM Plex Serif (right), rendered with the drawing technique D1 configured with two parameterisations (top and bottom).

Analysing the glyphs created with D1, we can observe that the results are quite satisfactory in regard to our attempt to replicate calligraphy. This is because both typefaces show identical characteristics of the stroke to examples of calligraphy, for example, when the direction of the stroke changes its length also varies according to the angle. One can observe that the resulting typefaces display a high level of legibility because we can easily identify and recognise every letter/glyph. Perhaps this is because the D1 has a strong visual identity, *i. e.,* there exists a strong coherence between the glyphs as all of them display the same anatomical parts rendered in a similar fashion (*e. g.,* a consistent length at vertical directions and in curve directions display the same variation on stroke thickness). Nevertheless, in some glyphs (such as "l", "p" and "h"), the stroke length becomes too thin at vertical directions which may cause some imperfections on the glyph.

Comparing D1 with both sets of parameters, we observed that applying an angle closer to the 30 degrees we achieve a better representation of calligraphy, *i. e.,* glyphs with a more linear stroke (bottom glyphs in Figure 8), than using a more vertical angle *i. e.,* glyphs with a high contrast stroke (top glyphs in Figure 8). Also, the level of detail (epsilon value) of the typographical skeleton has a big influence in the approximation of the results to the calligraphy, *i. e.,* the higher the level of detail, better is the approximation to calligraphy. Furthermore, it is notable at Figure 8 the visual diversity created through the variation of the parameters of D1.

Figure 9: Glyphs generated with the presented system using the typographical skeletons of the typefaces IBM Plex Sans (left) and IBM Plex Serif (right), rendered with the drawing technique D2 configured with two parameterisations (top and bottom).

For the two different sets of parameters tested in D2, we explored the parameterisation of two values: the stroke angle and the stroke length. In one parameterisation, we used an angle of approximately 135 degrees and a small stroke length (top glyphs in Figure 9). In the other one, we used an angle of approximately 30 degrees with a bigger stroke length (bottom glyphs in Figure 9).

Analysing the glyphs created with D2, we can observe that the resulted glyphs display pixelated shapes. This indicates that our attempt to explore the visual concept of the pixel, as the key element to develop a digital style, was successful. It is notable the similarities between this style and the D1. This is because we employed the principles used in D1 to build the method for D2, removing only some detail to achieve the pixelated effect. Consequently, we achieve a synergic relationship between the two styles where the final typefaces are visually interesting and appealing.

Analysing the generated glyphs, it is possible to notice their high level of legibility, since we can easily identify and recognise every letter/glyph. This occurs because both typefaces display identical pixel densities allowing a satisfying coherence between them. However, we observe that some glyphs (such as "l" and "h") do not seem to be generated using the same drawing technique. Perhaps this is because two consecutive points are vertically or horizontally aligned, and a single shape is drawn between them. Therefore, is not possible to reduce more detail creating the pixelated effect pretended.

Both sets of parameters offer a vast diversity of resulting glyphs by simply modifying some values of the parameters, such as the angle and length of the stroke. We also notice that when the stroke is smaller is more likely to appears imperfections and/or incoherent shapes between the glyphs (top glyphs in Figure 9).

Figure 10: Glyphs generated with the presented system using the typographical skeletons of the typefaces IBM Plex Sans (left) and IBM Plex Serif (right), rendered with the drawing technique D3 configured with two parameterisations (top and bottom).

For the two different sets of parameters tested in D3, we explored the shape of a rectangle. We explored this using a shape with lower stroke width on a typographical skeleton with a high level of detail (top glyphs in Figure 10) and a large stroke width on a typographical skeleton with a low level of detail (bottom glyphs in Figure 10).

Analysing the glyphs created with D3, we can observe that the typefaces generated by this drawing technique are quite satisfactory because demonstrate a high level of detail even using a basic geometric shape. This is easier to observe on parameters applied to generate the top glyphs in Figure 10, where we use a higher level of detail (lower epsilon value) than in the parameters used to generate the bottom glyphs in Figure 10.

Also, we can observe the resulted typefaces present a satisfactory level of legibility due to the simple shape that the

glyphs acquire by repeating the same shape and width (in this case, a rectangle) through all the typographical skeleton. This allows a strong coherence between all the glyphs since the same shape is rendered on the same anatomical parts of typographical skeletons that share the same structure. Although some glyphs appeared to have imperfections (such as the "l" on the bottom left in Figure 10), this is because the system connects the highest point of the typographical skeleton with the lowest point that belongs to the anatomical part of the serif. Therefore, they are not vertically aligned creating a diagonal line rather than a fully vertical line.

Comparing D3 with both sets of parameters presented previous, we perceived that the variation of the parameters creates a shorter visual diversity between glyphs. This may be caused by the lower influence that the shape width has on the glyphs overall shape.

After analysing each drawing technique individually, we can present some observations about the overall results. In what concerns the typographical quality of the glyphs obtained, it is notable the high transmission of anatomical parts from the original font structure to the glyphs generated. For instance, the glyphs that come from the serif typeface kept the serifs and the glyphs that come from the sans serif typeface continues without serifs. Also, the main relations between glyphs are maintained, *e. g.*, the ratio between the x-height, total height and ascenders and descenders.

In what concerns the visual diversity of the glyphs created, the results indicate that the system is able to create glyphs with different visual styles. Figure 1 shows glyphs generated using different drawing techniques defined by different sets of parameters and typographical skeletons with different levels of detail. It is noticeable the differences at the visual level as the image displays a vast range of glyphs with distinct shapes which consequently create typefaces with the same wide range of diversity.

By allowing the user to adjust the parameters of the drawing techniques and implement further ones, the system enables and encourage graphic experimentation in type design. The user is able to generate unusual glyphs that push the boundaries between expressiveness and legibility.

We consider the system has the potential for application in different kind of graphic design projects. The system may aid the designer in the creation of bespoke typefaces that provide designers with a method to express a concept or represent data, therefore communicating something else than language. For instance, by mapping the parameters of the drawing techniques to external data, *e. g.,* sound or temperature, we are able to generate typefaces that dynamically adapt to different contexts in which they are used (see, for example, the typeface LAIKA [3] created by Michael Flückiger and Nicolas Kunz). This is also useful in the creation of dynamic visual identities, an increasing practice in graphic design [5]. The system has also the potential for open-ended design projects, as it enables, for example, the on-demand generation of unique typefaces characterised, for example, by randomness (see, *e. g.,* the typeface Beowolf [12] created by Erik van Blokland and Just van Rossum). Figures 11 and 12 show possible applications of generated typefaces in the design of posters.

Figure 11: Mock-up of a poster for a surf championship using a typeface generated with the presented system.

Figure 12: Mock-up of a poster using different typefaces generated with the presented system.

More applications of the system can be found at cdv.dei.uc.pt/letterspecies.

The adaptive capacity of the results obtained proves that the main advantage of our system is to designs reactive typefaces. This enables design typefaces that fulfill the necessities and opportunities imposed by new media.

5 CONCLUSIONS AND FUTURE WORK

We have presented a computational system that generates glyphs by rendering typographical skeletons using different drawing techniques. We tested the presented system in order to explore and analyse the possibilities it creates. We highlight the following contributions: *(i)* a system capable of generating typefaces with legible and coherent glyphs; *(ii)* the ability to apply the same drawing technique to different typographical skeletons and get coherent typefaces that can be applied in multiple fields of graphic design; *(iii)* a system that expedites the process of drawing all the glyphs needed to create a typeface, as it creates a base typeface that can be further refined; *(iv)* a system where the user has in his/her hands the possibility of generating a custom typeface by choosing the typographical skeleton and the drawing technique and in the subsequent modification of the parameters inherent to the chosen drawing technique to achieve a result that he likes; and *(v)* how legibility and coherence between glyphs can be achieved when utilizing innovative approaches to create typefaces.

Future work will focus on: *(i)* develop more drawing techniques; *(ii)* implement an automatic kerning process to adjust the space between the generated glyphs and this way achieve a more visually pleasing text composition; *(iii)* test the presented system in the generation of glyphs for non-Latin characters; *(iv)* further develop the system as a web-based tool in order to make it usable by anyone; *(v)* test the system with users to improve it by taking into account the feedback received; and *(vi)* integrate variable font technology in the system to make it possible to control the parameters of the drawing technique in design software tools that support this technology.

ACKNOWLEDGEMENTS

The third author, Sérgio Rebelo, is funded by Fundação para a Ciência e Tecnologia (FCT), Portugal, under the grant SFRH/BD/132728/2017.

REFERENCES

[1] Tatevik Aghabayan. [n. d.]. Gestalten mit Code, FH Mainz. http://generativetypografie.de/generativetypografie/elien/ accessed on January 2019.
[2] Robert Bringhurst. 2008. *Elementos do Estilo Tipográfico*. COSAC NAIFY, São Paulo, Brazil.
[3] Michael Flückiger and Nicolas Kunz. [n. d.]. LAIKA. http://www.laikafont.ch/ accessed on January 2019.
[4] Elias Hanzer. [n. d.]. Phase. https://www.eliashanzer.com/phase/ accessed on January 2019.
[5] Tiago Martins, João Miguel Cunha, João Bicker, and Penousal Machado. 2019. Dynamic Visual Identities: from a survey of the state-of-the-art to a model of features and mechanisms. *Visible Language* 53, 2 (2019), 4–35.
[6] Yannick Mathey. 2009. Prototypo. https://www.prototypo.io/ accessed on June 2019.
[7] Jéssica Parente. 2018. *Desenho Generativo de Tipos de Letra*. Master's thesis. University of Coimbra.
[8] Jéssica Parente, Tiago Martins, and João Bicker. 2018. Generative Type Design. In *Proceedings of the Ninth Typography Meeting (9ET)*. Instituto Politécnico de Tomar, Tomar, Portugal.
[9] Alexis Reigel and Marco Müller. 2012. Metaflop. https://www.metaflop.com/about accessed on January 2019.
[10] Ingo Reinheimer. [n. d.]. Gestalten mit Code, FH Mainz. http://generativetypografie.de/generativetypografie/irratio/ accessed on January 2019.
[11] Gerard Unger. 2018. *Theory of Type Design*. nai010 publishers, Rotterdam, Netherlands.
[12] Just van Rossum and Erik van Blokland. [n. d.]. Beowolf. https://www.fontshop.com/families/ff-beowolf accessed on January 2019.
[13] S. . Wu and M. R. G. Marquez. 2003. A non-self-intersection Douglas-Peucker algorithm. In *16th Brazilian Symposium on Computer Graphics and Image Processing* (SIBGRAPI 2003). 60–66. https://doi.org/10.1109/SIBGRA.2003.1240992
[14] T. Y. Zhang and C. Y. Suen. 1984. A Fast Parallel Algorithm for Thinning Digital Patterns. *Commun. ACM* 27, 3 (March 1984), 236–239. https://doi.org/10.1145/ 357994.358023
[15] Kevin Zweerink. 2015. NType. https://experiments.withgoogle.com/ntype accessed on December 2018.
[16] Kevin Zweerink. 2015. NType. https://github.com/kevinzweerink/ntype accessed on December 2018.