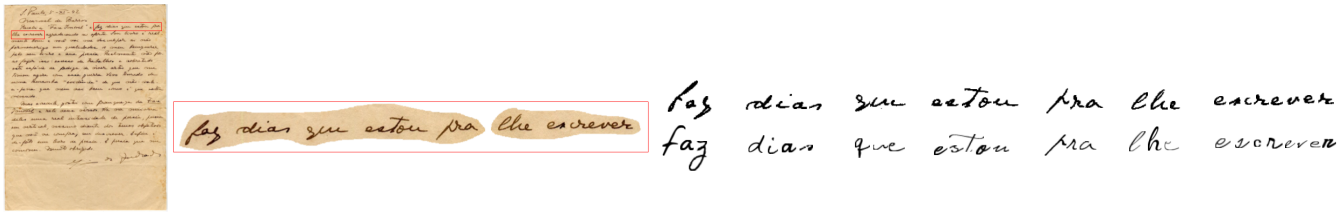
Handwriting Synthesis from Public Fonts

Dennis G. Balreira, M. Walter
Institute of Informatics - Universidade Federal do Rio Grande do Sul - Brazil



(a) (b) (c)

Fig. 1: Example of Handwritten Synthesis using our approach. (a) Digitized letter from Brazilian writer Mário de Andrade. (b) Excerpt of the letter used as input to our algorithm. (c) On the top the same excerpt processed to remove texture and color and below our synthesis result.

***Abstract*—Handwriting synthesis generates renderings of text which look like they were written by a human but are in fact synthesized by a model. From an input sample of the desired handwriting, we introduce an algorithm that finds the best match between characters using as source for the output text the large collection of publicly available fonts designed to look like handwriting. For each character in the desired output text, we find the best match among the public fonts using a metric that matches both the shape and appearance of the input real character. Once we have the set of best characters we build the output sentence or paragraph by concatenation of individual characters. Our results show that even though human calligraphy is highly individual and specialized, visually similar renderings are possible for many applications that do not demand full similarity. On a user study with 12 subjects, our synthesis results were considered, on average, 71% similar to the input samples.**

I. Introduction

With the advance of digital technologies, many forms of communication became old-fashioned and were exchanged by faster and more efficient ways. Handwritten text is one example. Today, we rely almost 100% of the time on typed digital text send using e-mail and many other forms. At the same time, handwritten messages now have special status and are used in particular cases, suggesting care and friendship between sender and receiver. There is even a market for letters written by hand, where companies handwrite the letter for customers [1]. Digital messages cost close to nothing to produce whereas handwritten text still requires attention and usually more time than digital communication. In this context, an interesting possibility is to turn any input string into images that look like they have been handwritten, down to the texture of the pen or pencil.

The goal of handwritten text synthesis techniques is to provide images that look like were written by a human hand, either an anonymous hand or, harder, in the handwriting of a particular person, alive or not. The field is not new, with research going as back as the early 90s [2] but has attracted new attention recently [3]. There are many applications which could benefit from faster automatic handwritten synthesis. Online cards for special occasions could be sent, for instance, with text messages written in the sender's handwriting. Applications in the creative industries include lettering in comics and special editions of books. Artificial handwriting could also be used to generate CAPTCHAS, instead of machine generated ones. Artificial CAPTCHAS continue to be readable by humans but less prone to automated attacks [4] [5]. Properly printed, artificial handwritten notes could be used for more personal communication between companies and customers.

Our main contribution is an algorithm that searches for the best match given an input glyph and a set of candidates. A *glyph* is a technical term often used in typography, and it conveys the idea of a graphic symbol that represents a writable meaningful character. For instance, the character **a** is a glyph, whereas the dot on top of the character **i** is not, since it does not convey meaning by itself. For a particular glyph, we can have many different graphical representations. The input glyphs are real handwritten text, and the candidates are from families of handwritten-like fonts freely available to the public. We define a two-step metric to compare glyphs, comparing first their shapes followed by how similar they are considering line thickness. From the selected glyphs we build the sentences and paragraphs using concatenation. In Fig. 1 we illustrate one result from our technique. We used as input the real handwriting of Brazilian writer Mário de Andrade to synthesize a similar output.

Although human writing is individual and can be traced to a particular person [6], we claim that the large number of fonts used as input for synthesis provides a degree of variability which allows renderings that look like were handwritten by a particular person, without the expensive cost of previous approaches. Our central insight is the use of a large number of publicly available fonts to find the best match to a particular person's handwriting.

II. Related Work

There are many websites [7]–[9] that generate a font family from a user's handwriting. The user writes the alphabet of characters, one by one, on a given template grid which is later digitized and used as input for computing the family font. However attractive, these approaches suffer from the fact that the writing of individual characters makes the process artificial and the results often look unnatural. Closely related to our work is the website **WhatTheFont** [27]. From an image of a particular character, they provide 10 suggestions of similarly looking characters. However interesting, many suggestions do not provide good matches. Besides, it is a black box without any information on how exactly the matches are computed, and apparently only suggests families from their own web service. Later, in the results, we show how our solution compares with one computed using the output from this website.

On more scientific approaches, the recent survey by Elarian and colleagues [10] provides a useful starting point. At the highest level, the methods are classified as either top-down or bottom-up. In the top-down approach, the solution tries to simulate the actual neuromuscular movements of the human arm and hand to produce the results. Bottom-up approaches also known as shape simulation, on the other hand, model the shape of individual units to obtain the result. The shape simulation category is further subdivided into generation and concatenation. Since our solution uses a concatenation approach, we, therefore, review here only these approaches.

Historically, many handwriting synthesis research was developed to help research on handwritten character recognition. The work by Rao [2] is one example of this approach. He synthesized characters as a collection of straight line segments approximated from a sample of a real written character. This representation could thus be used to recognize handwriting text. Guyon [11] introduced in 1996 a simple technique to render handwritten texts assembling *glyphs* from samples collected from the end user. The synthesis results look nice, but the technique needs as input a long list of 100 3-letter entries, called *lexicon* to be handwritten by the user. Also, the lexicon was targeted for English, and it is not clear how it would work with other languages. At least new sets of 3-letter entries would have to be built for the technique to work.

Wang *et al.* [12] presented a two-step learning model on the points describing a B-spline curve for each sampled character. They explored the idea of tri-units considering that each character, in general, is surrounded left and right by neighbors that affect how the cursive writing flows. Helmers and Bunke defined three different methods for synthesis of handwritten text [13] to be used in the context of handwriting recognition. They show that the recognizer usually performs equal to or slightly better than using only real handwriting. A learning approach combining shape models and physical models was presented by Wang and colleagues [14]. They used input text written on a tablet. As with many of the other approaches here revised, the validation is only subjective, by visual comparison between input and output. Lin and Wan collected features about a particular handwriting and used these to synthesize new text in a hierarchical fashion [15]. Variation is introduced at the character level, and the sampling of input text uses a specially designed interface. In the context of CAPTCHA generation, Thomas, Rusu, and Govindarauj [4] introduced artificial CAPTCHAS that look like they were handwritten but without being writer-specific. Their generated CAPTCHAS performed better against automated bots.

The following methods are also categorized as concatenation using a shape simulation approach. However, we do not review them here since they target either Arabic [16]–[18], Indian [19], or Japanese [20] glyphs.

Although there is more research on top-down approaches [10], the current state-of-the-art is a shape simulation algorithm recently proposed by Haines and colleagues [3]. They present a complex pipeline of manual and automatic steps to render high-quality images of handwritten text. From an annotated sample of the desired handwriting, their solution performs a semi-automatic analysis step followed by the synthesis. The synthesis step is formulated as an optimization problem where selection and spacing among glyphs are used in the cost function. The last step is color matching and texturing of the output.

Even though we present a simpler solution, ours is the first to explore the high availability and visual improvement of families of fonts designed to look like handwriting. We believe our handwriting synthesis useful for applications that do not demand full similarity and could be used in artistic or commercial applications. In Sec. IV we show how our renderings compare with the ones from [3], applied to the same input sample.

III. Handwritten Synthesis

In this section, we present our technique for handwriting synthesis using public fonts. Given a handwritten text and families of fonts as input, our aim is to match, for each real input glyph, the best one from the families of fonts. Once we have the collection of best matches, we can render any string similarly to the real input handwriting. Our inputs are the sample of handwritten text that we want to replicate, plus a collection of families of fonts that are publicly available. There are many possible options to capture a person's written text to use as input. Input samples captured with pen or pencil on paper are sometimes preferable to use as input since this mode better captures hand flow, but need to be digitized later. Samples collected from direct writing on tablets are already digitized but are usually less natural. Therefore, we used as input some of the samples written on paper available from [3], among others.

As our second input and source for the public fonts, we collected 120 families of fonts that are carefully designed to be similar to handwriting. Of these, 87 families are available as Google fonts (https://fonts.google.com/) and the remaining 33 families are available from the web portal Free Calligraphy Wedding [21] (18 families) and from the web portal 1001 fonts [22]. From this last portal, we used 15 families arbitrarily

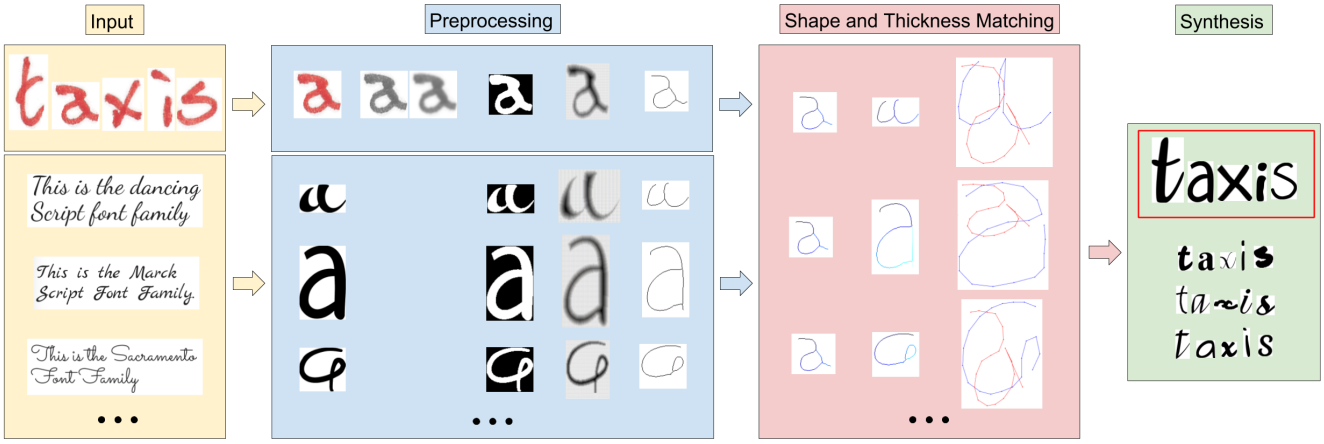


Fig. 2: Overview of our technique. Input: sample of handwritten text and families of handwritten-like fonts. Preprocessing (Sec. III-A): process the images representing the glyphs to extract information needed for the shape matching step. Shape and thickness matching (Sec. III-B): compares each glyph from the user input with all the font database glyphs in order to find the best match. Synthesis (Sec. III-C): concatenate the best glyphs to simulate the text input positions.

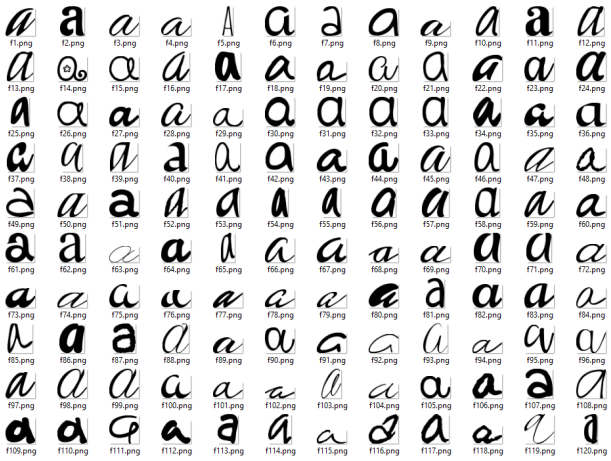


Fig. 3: Glyph 'a' for all 120 families of fonts used as input.

selected from the most popular ones. In Fig. 2 (Input) we illustrate three of the Google families and in Fig. 3 we show the glyph 'a' for all 120 families.

Given the sample of handwritten text, we first manually segment each glyph from the input sentence, generating an image for each. Then, we execute the next steps in order to select the best glyph among the same type from the font families. For instance, an 'a' glyph will only be compared with other a's. Our main technique is divided into three steps: preprocessing, shape matching and synthesis, detailed below. The synthesis part is user-assisted whereas the shape matching and preprocessing are automated, although some parameters can still be calibrated to control the results. In Fig. 2 we present the overall process.

A. *Preprocessing*

Our preprocessing perform operations on the inputs to extract the necessary characteristics for the comparison described in Sec. III-B (*Shape and Thickness Matching*). We apply simple image processing operations on the inputs, followed by a specific thinning strategy to capture specific features from the images. Fig. 4 illustrates all the preprocessing operations using a simple drawing of an 'i' glyph that has some specific characteristics and triggers all the six steps described below.

1. Basic operations on the sample input: since we do not use the pen texture, we start by converting the color image into a gray level one. Digitized natural handwritten text presents small variations intrinsic to the writing process which appear as noise at the border of the characters. We apply a Gaussian filter to remove this noise, which helps compute the glyph skeleton in the next step. We also binarize the image according to a threshold to separate the background (black) from the foreground (white). Next, we crop each image to adapt to its bounding box plus 1 pixel in each dimension to avoid future problems with the thinning algorithm. Fig. 4-a to 4-e illustrate these steps.

2. Size normalization: in this step we generate, for each glyph from each font family, an image that matches approximately in size the resolution of the corresponding sample input. Since the designed glyphs have very different aspect ratios, we need a criterion for normalization. For instance, if three occurrences of glyph 'a' are found in an input text, and the largest one has width of 90 pixels, the glyph 'a' for all families of fonts will have this same width but different heights, according to its original design and maintaining its aspect ratio. In this step, we also repeat all the basic operations described in step (1) above for the resized font images, except the Gaussian filter. Considering that the fonts usually do not have noise, we thought that it could erroneously mask some intended artistic effects created by the font's authors.

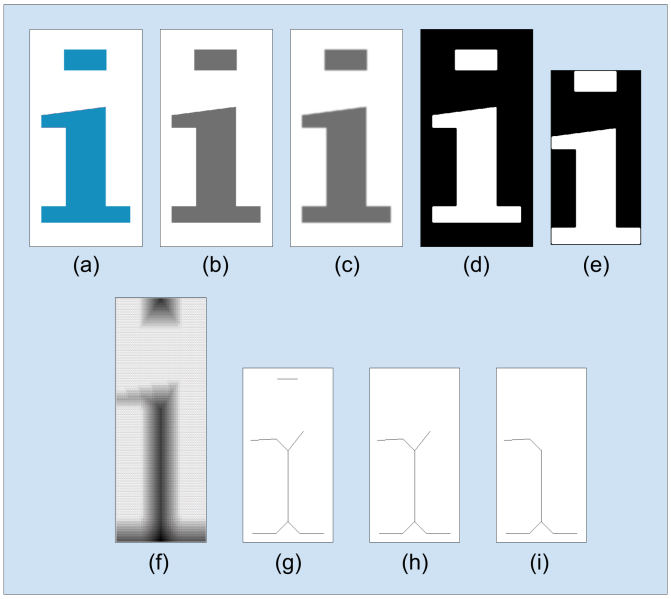


Fig. 4: Preprocessing pipeline for the glyph 'i' . (a) original image; (b) image a) converted to gray level; (c) image b) with gaussian filter; (d) image c) binarized; (e) image d) with bounding box + 1 pixel; (f) distance transform applied to image e); (g) image e) after thinning; (h) image g) after disconnected paths removal; (i) image h) after the small ramifications removal.

3. Distance transform: in this step, we compute the distance transform [23] for both the input text and the font images. We use it later in our thickness metric. Fig. 4-f shows a representation of the distance transform: the bigger the numbers, the stronger (black) the pixels.

4. Skeletonization: we apply the thinning approach by Zhang and Suen [24] to compute the skeleton of the glyphs. After computing the skeletons, we still need to perform two operations on the pixels describing the skeleton preparing them to apply our distance metric. First, we convert the skeleton into a one-pixel wide one through the algorithm described in [25]. This is important because our distance metric needs a continuous set of points, which is detailed in the next subsection. Second, we compute the skeleton's *end* and *branch* pixels, which will have an important role in the next steps. End pixels are all the pixels that have only one neighbor, whereas branch pixels are all the pixels that have three or more neighbors, both considering an 8-connected neighborhood. Fig. 4-g presents the skeletonization obtained from Fig. 4-e.

5. Disconnected paths removal: this operation consists in removing all the disconnected pixel paths except for the largest one, which will be considered the main skeleton. We start from an endpoint and count the number of pixels found, erasing them until they have all been processed. If an image has no end points, we start with the bottom-most left pixel of the skeleton by default. When a branch pixel is found, we store it for a later check when a path is over. When a path is over, we store its number of pixels and start all over again while the number of visited pixels is different from the total of the skeleton pixels. The biggest path is preserved as an image, and the other ones are erased. Fig. 4-h illustrates this step. The algorithm finds two disconnected segments and removes the upper line which had fewer pixels than the body of the 'i' glyph. As a side note, in English the majority of handwritten characters do not have disconnected parts, since English does not have accents.

6. Small ramifications removal: here we remove small pixel ramifications that could be considered as noise produced by the thinning algorithm. A ramification occurs when a pixel path starts on an end pixel and goes through a branch pixel or vice-versa. We defined as noise ramification all the ramifications that have at least 10% of the total number of skeleton pixels, not counting the branch points. We chose this percentage through simple experiments. Then, all noise ramifications are removed from the path. These paths are easily detected using a variation of the algorithm described in the last step. Fig. 4-i shows the upper right ramification removal, which in this case helps to improve the skeleton result considering the shape of glyph 'i'.

We now have the final skeleton of all the images to compare each glyph from the input text with the glyphs from the fonts. The next subsection describes the algorithm to compare the shape and thickness among glyphs.

B. Shape and Thickness Matching

For each glyph in the input text we have to find, among all available public fonts, the best match. For this task we define a metric that first compares the glyph shapes followed by a comparison of their line thickness. Our shape metric requires as input one-to-one point correspondences between the two shapes. This means finding an ordered list with the same amount of points for both glyph skeletons. We compute this list from the pixels that define the skeletons. We divided this process into five steps, detailed below.

1. Defining the starting pixels: since we will create an ordered list of pixels for each skeleton, our first task is to choose the most similar starting pixel for both skeletons. We accomplish this by using a simple Euclidian distance through all the end pixels respecting the size normalization of both images. As described in Sec. III-A step 5, if an image has no end pixels, we start with the bottom-most left pixel by default. The pair chosen is the one that has the least distance found among all the comparisons. Fig. 5-b illustrates the starting point found for both example images in 5-a.

2. Ordered pixel list creation: starting from the initial pixels found in the previous step, we need to go through both skeletons capturing all the pixels along the way. For each pixel from the starting one, we decide the next 8-neighbor according to the following priority: branch point, then clockwise order starting from the bottom pixel, as illustrated in Fig. 6 for a sequence of pixels. We mark each visited pixel and every time a pixel has no new neighbors we backtrack through all the pixels already visited, beginning with the branch points in their

visited order until there is no one left. Since all the pixels are guaranteed to be connected because of our preprocessing phase (step 5), the algorithm will surely end at some pixel. The use of this simple rule makes sure we are moving through the same path when creating the list of pixels for both skeletons. Not only that, but we also guarantee a consistent path even for considerably different skeletons. Fig. 5-c shows the list of pixels and their direction on the list. At the end of this step, we have an ordered list of pixels for each skeleton.

3. Point list creation: since the shape matching requires two sets of the same quantity of ordered points, we need to sample both sets of ordered pixels such that they have exactly the same amount of points. We used a linear approach which given the number of points to be sampled returns points equally spaced along the pixel list. If one ordered list of pixels has fewer points than asked, we sample everything possible and just repeat the last value till it reaches the given number. Even though this parameter can be changed, we opted to fix it to 20 points through all the experiments, since it presented good results overall. Fig. 5-d illustrates the final points chosen.

4. Shape matching: the shape comparison uses the Procrustes distance applied to the list of points obtained in the previous step. Given two shapes with one-to-one point correspondences, we compute the distance between them as the sum of the squared distances between equivalent points, known as Procrustes method [26] and usually used in shape analysis. We used Procrustes since it is simple and deals with eventual translations, rotations and scales. We use the return value $d \in [0, 1]$ to measure how similar the shapes are. The closer to zero, the more similar the shapes are. On a small subset of the best matches found using the Procrustes distance, we apply a second metric that compares the thickness of the characters, described in the next step.

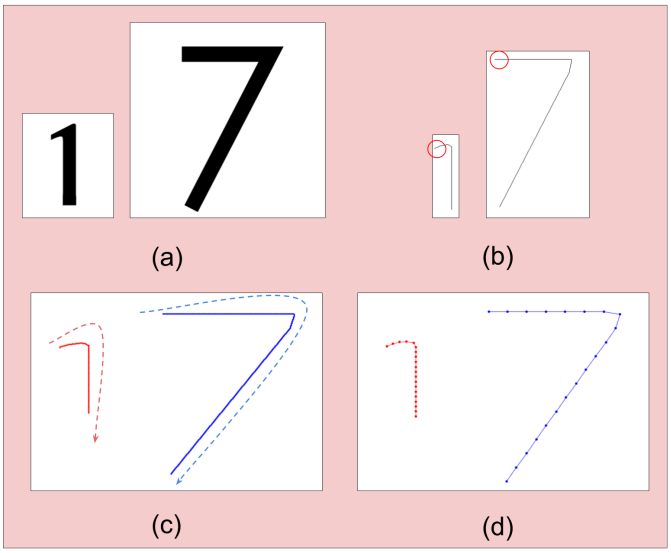


Fig. 5: Pipeline for computing an ordered pixel list for the skeletons of two characters representing numbers. (a) input images of a '1' and a '7'. (b) result of the preprocessing phase for both images in a) and their respective starting pixels selected. (c) ordered list of pixels and their direction on the list. (d) points from c) after linear sampling for 20 points.

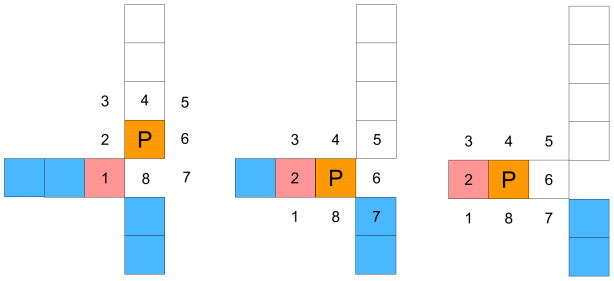


Fig. 6: Ordered pixel list creation. P represents the current pixel and the numbers 1 to 8 express the priority to visit the next pixel. Example of three iterations to show the pixel to be chosen. The white pixels are already visited, the blue ones still need to be visited and the red is the chosen one to be visited next.

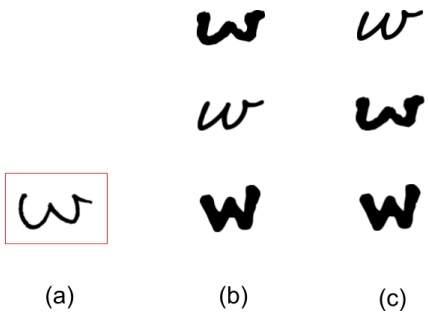


Fig. 7: Shape and thickness results. (a) glyph 'w' handwritten. (b) best three 'w' computed using only the shape comparison. (c) best three 'w' of our shape metric ordered by the thickness algorithm.

5. Thickness matching: our line thickness comparison process independently of the shape matching and consists in comparing the points using the distance transform previously calculated. For each sampled point on the list, we get its value in the distance transform matrix for both images. We calculate the absolute difference for each pair of values and keep on summing them all until no pair is left. At the end we obtain a value $t \geq 0$ representing the thickness. Just as with the distance metric, the closer to zero, the better the result. Since the points shall have a one-to-one correspondence, our approach captures the difference of thickness between the characters. In Fig. 7 we illustrate a glyph and the results of first applying the shape metric followed by the thickness metric. We can see the reordering of the best matches according to the thickness criterion.

C. Synthesis

This is the last and simplest step. Once we have the glyphs that minimize the distance between the input and the available

public fonts, we are ready to write the output as a collection of glyphs that will be placed side by side such that they look like they were written as a unique text. When available, we tried to manually mimic the same vertical and horizontal spacing of the original whenever possible.

IV. Results

Here we present the main results of our method. We implemented our prototype using Processing and Matlab. We computed our results in an i5 3.50GHz. First, we present example of possible uses for our results, followed by the results we collected from a user study.

In Fig. 1 we show a synthesis result from an excerpt of a letter from Mário de Andrade, an important Brazilian writer. We can see that even without the ligatures among glyphs typical of cursive writing, the overall result is still similar to the input. This result took 306 milliseconds in total to compute: $146ms$ for generating all the font images for all the glyphs and $160ms$ for computing the matching among glyphs. The largest bounding box of all the input was for the glyph 'f' with 42 x 43 pixels.

We also present other examples of applications of our approach in different contexts. For input, we used the sample handwriting from subjects 2 (S02), 3 (S03), 4 (S04), 6 (S06) and 9 (S09) from [3]. These samples provide variability among individuals as well as variability with respect to the sentences and writing tool used. S02, S03 and S06 used a fine liner pen; S04 used a fountain pen, and finally, S09 used a gel pen. Fig. 8 (a) presents a message created using the best matched fonts for S02 in a flower card, (b) S03 writing in an e-mail, (c) S06 in a medical prescription, and (d) an artificial CAPTCHA. For the artificial CAPTCHA we used the glyphs combination that was considered the worst in the study we performed to assess our results, described next.

We compared our results directly with [3] through a paragraph wrote by Sir Arthur Conan Doyle's real handwriting. We synthesized the sentence "Elementary my dear Watson", presented in Fig. 9. We extracted the samples from the same paragraph used in [3], although in our case we do not have individual variation for glyphs of the same letter. Even though our solution is simpler than [3], our synthesis still provides satisfactory results for many applications that do not demand full similarity. Further, we compared our results with the website WhatTheFont [27] in Fig. 10. We used as input a sentence from user S09. The website generated the top 10 best matches. We selected the first of each and placed them side by side. Even though a few characters are visually similar, in general, for real handwriting samples, WhatTheFont has difficulties in providing adequate suggestions.

A. Validation

We designed an experiment to assess how similar to the human input our results are. For input, we used the same as before, that is, the handwriting from S02, S03, S04, S06 and S09 from [3]. Fig. 11 shows four out of five results we obtained by running our algorithm. The original sentences are highlighted in a red box and our results highlighted in blue.

Since handwriting is usually seen on paper, we provided printed copies of the real handwriting input, printed at the top of the page, together with 5 of our results randomly positioned below the original handwriting. We ordered the results initially by the shape metric and then, only for the first three best matches, we ordered again by using the thickness metric. We provided 3 of our best matches along with the "best" results in positions 40th and 100th. We asked the subjects to grade each result on a scale of 0 (zero) to 10 (ten) assessing how similar they considered each result when compared to the original handwriting. Twelve subjects took part in the experiment, with an average age of 25 years, all Computer Science students. We had two different test versions in which we shuffled the answers and the subjects order.

Table I presents the average of grades for all cases. For the first three subjects, the results of our technique received the highest grades, as expected. For all subjects, the best grades were consistently assigned for the best three synthesized sentences. Further, we obtained an overall average grade of 7.1 considering only the highest scores for each result.

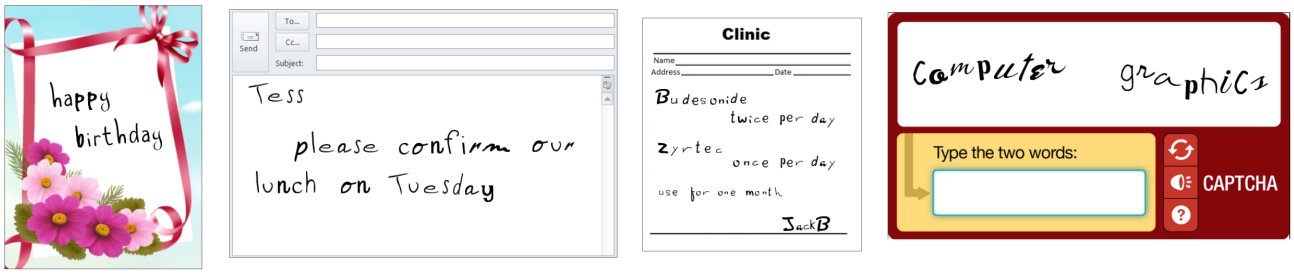
	S02	S03	S04	S06	S09
1st sentence (best)	**6.42**	**8.17**	**7.33**	5.58	6.33
2nd sentence	**6.42**	6.33	6,25	**6.75**	6.75
3rd sentence	5.17	5,17	4.42	5.00	**6.92**
40th sentence	3.08	3,42	2.75	3.58	4.25
100th sentence	1.33	2.08	3.17	2.58	1.83

TABLE I: Results containing the average scores of our experiment. In bold, the highest grades for each subject.

V. Conclusions

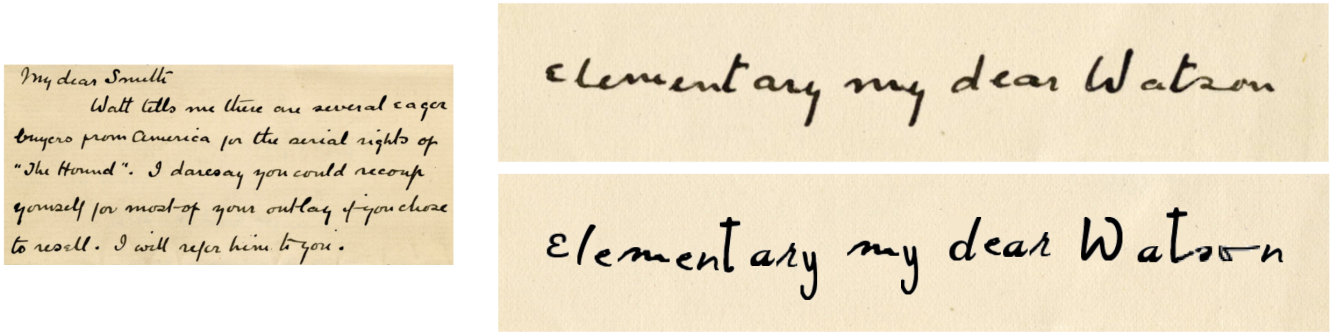
We have presented a technique for the synthesis of text that looks like it has been handwritten according to a particular style from a person. Given a user text sample and some families of fonts, our approach finds the fonts that best match the user calligraphy. Each person may have it's own font family created by the concatenation of the best samples from several families. We accomplish this by a preprocessing step followed by a matching and thickness step, applied to individual characters. We tested our approach with low resolution glyphs extracted from a real letter and achieved visually similar results. Besides, we conducted a user validation study that presented positive results overall. Our technique has several uses, which we demonstrated through some simple examples from artistic applications to CAPTCHA generation.

Our technique has few limitations. Both the text segmentation and the glyph concatenation uses manual work. This requires some effort to position the glyphs correctly so that they resemble the original. The thinning strategy can ignore some important features of the glyphs and the branch removal sometimes may delete an important part of a letter. For future work we aim to work on the above limitations and expand our inputs to deal with accents and other languages besides English. Furthermore, we intend to explore new matching



(a) (b) (c) (d)

Fig. 8: Examples of synthesized applications using the best match from three different subjects from [3] and the 100th match from the first subject. (a) birthday card (S02). (b) E-mail (S03). (c) medical prescription (S06). (d) artificial CAPTCHA (S02).



(a) (b)

Fig. 9: Comparison between results from [3] and ours for a letter wrote by Sir Arthur Connan Doyle. (a) Sampled paragraph of the original letter. (b) Result synthesized by [3] on the top and our synthesis result below.

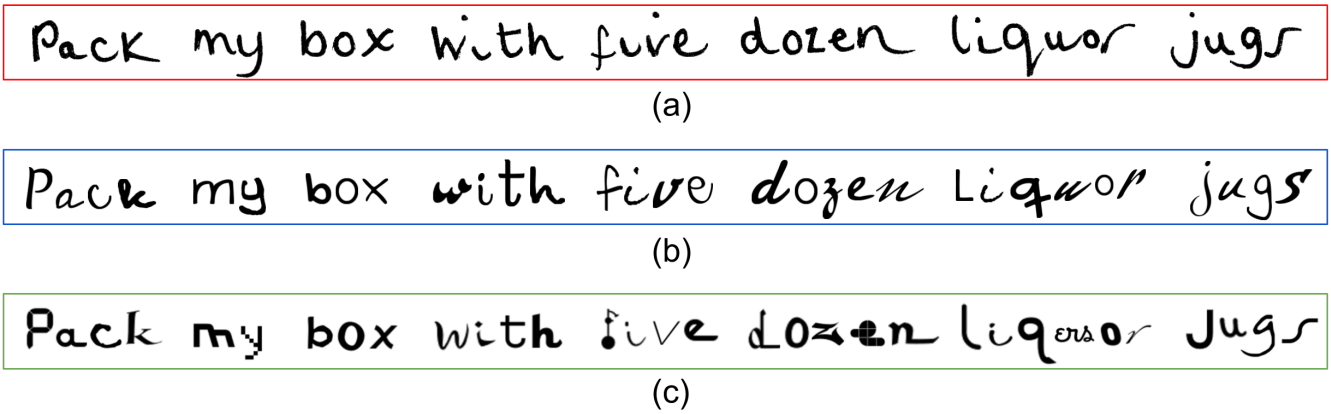


Fig. 10: Comparison between results from the website WhatTheFont [27] and ours using a sample of S09 [3]. (a) Original sample. (b) Our best result. (c) WhatTheFont first result.

techniques, such as Iterative Closest Point and Hausdorf Distance. It would be very nice to also replicate the texture of the original handwriting, when available, as done in [3]. We also intend to increment the number of families of fonts, support online input handwriting (from a tablet, for instance) and synthesize the ligatures among glyphs of different families of fonts. This possibility would increase the visual similarity.

Acknowledgments

We gratefully acknowledge the scholarship funding from CAPES for the first author.

References

[1] "The indian handwritten letter co." http://www.tihlc.com, accessed: 2017-05-30.

[2] P. Rao, "Shape vectors: an efficient parametric representation for the synthesis and recognition of hand script characters," *Sadhana*, vol. 18, no. 1, pp. 1–15, 1993.

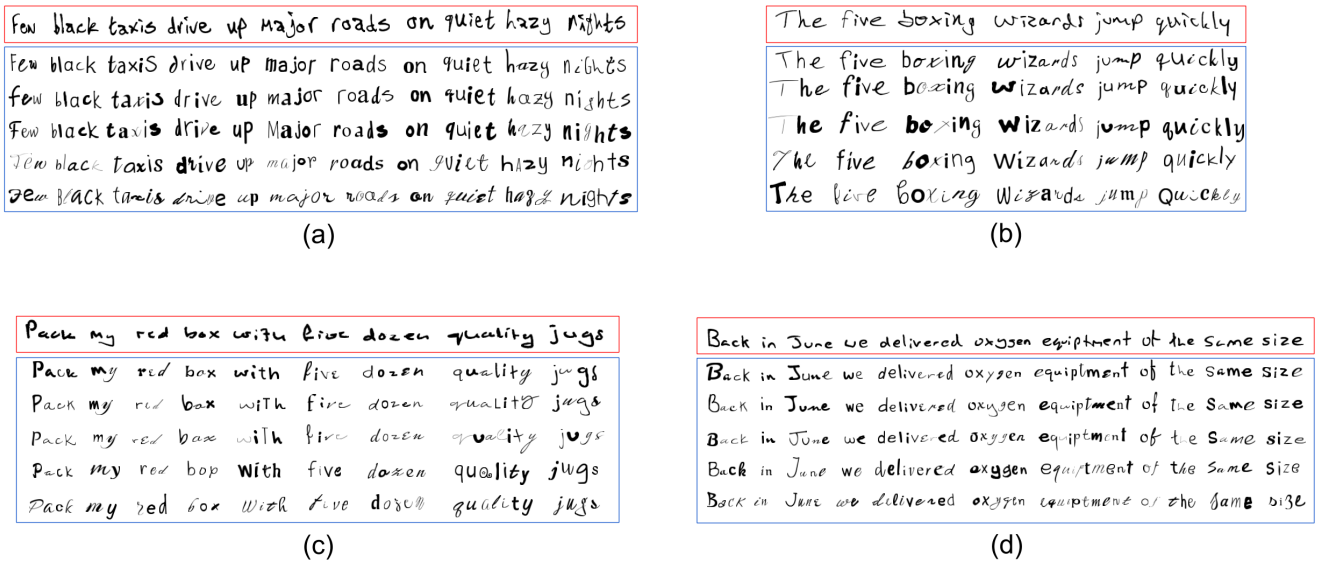


Fig. 11: Four out of five synthesis results used in our experiment. All the sentences are ordered from top to bottom: original (highlighted in red), 1st, 2nd, 3rd, 40th and 100th (highlighted in blue). (a) S02. (b) S03. (c) S04. (d) S06.

[3] T. S. Haines, O. Mac Aodha, and G. J. Brostow, "My text in your handwriting," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 3, p. 26, 2016.

[4] A. O. Thomas, A. Rusu, and V. Govindaraju, "Synthetic handwritten captchas," *Pattern Recognition*, vol. 42, no. 12, pp. 3365–3373, 2009.

[5] A. Rusu, A. Thomas, and V. Govindaraju, "Generation and use of handwritten captchas," *International Journal on Document Analysis and Recognition*, vol. 13, no. 1, pp. 49–64, 2010.

[6] S. N. Srihari, S.-H. Cha, H. Arora, and S. Lee, "Individuality of handwriting," *Journal of forensic science*, vol. 47, no. 4, pp. 1–17, 2002.

[7] "Calligraphr," https://www.calligraphr.com/, accessed: 2017-05-30.

[8] "Myscriptfont," http://www.myscriptfont.com/, accessed: 2017-05-30.

[9] "Yourfonts," http://www.yourfonts.com/, accessed: 2017-05-30.

[10] Y. Elarian, R. Abdel-Aal, I. Ahmad, M. T. Parvez, and A. Zidouri, "Handwriting synthesis: classifications and techniques," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 17, no. 4, pp. 455–469, 2014.

[11] I. Guyon, "Handwriting synthesis from handwritten glyphs," in *Proceedings of the Fifth International Workshop on Frontiers of Handwriting Recognition*. Citeseer, 1996, pp. 140–153.

[12] J. Wang, C. Wu, Y.-Q. Xu, H.-Y. Shum, and L. Ji, "Learning-based cursive handwriting synthesis," in *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*. IEEE, 2002, pp. 157–162.

[13] M. Helmers and H. Bunke, "Generation and use of synthetic training data in cursive handwriting recognition," *Pattern Recognition and Image Analysis*, pp. 336–345, 2003.

[14] J. Wang, C. Wu, Y.-Q. Xu, and H.-Y. Shum, "Combining shape and physical models for online cursive handwriting synthesis," *International Journal of Document Analysis and Recognition (IJDAR)*, vol. 7, no. 4, pp. 219–227, 2005.

[15] Z. Lin and L. Wan, "Style-preserving english handwriting synthesis," *Pattern Recognition*, vol. 40, no. 7, pp. 2097–2109, 2007.

[16] R. Saabni and J. El-Sana, "Efficient generation of comprehensive database for online arabic script recognition," in *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*. IEEE, 2009, pp. 1231–1235.

[17] R. M. Saabni and J. A. El-Sana, "Comprehensive synthetic arabic database for on/off-line script recognition research," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 16, no. 3, pp. 285–294, 2013.

[18] Y. Elarian, I. Ahmad, S. Awaida, W. G. Al-Khatib, and A. Zidouri, "An arabic handwriting synthesis system," *Pattern Recognition*, vol. 48, no. 3, pp. 849–861, 2015.

[19] C. Jawahar and A. Balasubramanian, "Synthesis of online handwriting in indian languages," in *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[20] H. Fujioka, H. Kano, H. Nakata, and H. Shinoda, "Constructing and reconstructing characters, words, and sentences by synthesizing writing motions," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 36, no. 4, pp. 661–670, 2006.

[21] "34 free calligraphy script fonts for wedding invitations," http://www.prettyweddingpaper.com/34-free-calligraphy-script-fonts-for-wedding-invitations, accessed: 2017-05-30.

[22] "1001 fonts," http://www.1001fonts.com/handwritten+handwriting-fonts.html, accessed: 2017-05-30.

[23] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno, "2d euclidean distance transform algorithms: A comparative survey," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, p. 2, 2008.

[24] T. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, 1984.

[25] L. Lam, S.-W. Lee, and C. Y. Suen, "Thinning methodologies-a comprehensive survey," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 9, pp. 869–885, 1992.

[26] M. B. Stegmann and D. D. Gomez, "A brief introduction to statistical shape analysis," *Informatics and mathematical modelling, Technical University of Denmark, DTU*, vol. 15, no. 11, 2002.

[27] "Whatthefont," https://www.myfonts.com/WhatTheFont/, accessed: 2017-08-16.