My Text in Your Handwriting

Tom S. F. Haines, Oisin Mac Aodha and Gabriel J. Brostow
University College London

There are many scenarios where we wish to imitate a specific author's pen-on-paper handwriting style. Rendering new text in someone's handwriting is difficult because natural handwriting is highly variable, yet follows both intentional and involuntary structure that makes a person's style self-consistent. The variability means that naive example-based texture synthesis can be conspicuously repetitive.

We propose an algorithm that renders a desired input string in an author's handwriting. An annotated sample of the author's handwriting is required; the system is flexible enough that historical documents can usually be used with only a little extra effort. Experiments show that our glyph-centric approach, with learned parameters for spacing, line thickness, and pressure, produces novel images of handwriting that look hand-made to casual observers, even when printed on paper.

Categories and Subject Descriptors: I.4.7 [**Image Processing and Computer Vision**]: Feature Measurement—*Texture*; I.5.4 [**Image Processing and Computer Vision**]: Applications—*Text processing*

Additional Key Words and Phrases: Texture Synthesis, Handwriting, Generative Models

ACM Reference Format:

Haines, T. S. F., Mac Aodha, O. and Brostow, G. J. 2015. My Text in Your Handwriting. ACM Trans. Graph. VV, N, Article XXXX (XXXX 2015), 28 pages.

1. INTRODUCTION

The worldwide adoption of digital messaging has given handwritten communication a special status as the approach for important, personally expressive messages. But what can an author do if, for example, suffering a stroke changes their handwriting, impacting their style and hurting the legibility of their messages? Handwriting-like fonts [Knuth 1979] and writing on a tablet [Zitnick 2013] represent two families of solution that may help, but look obviously synthetic. Our proposed algorithm is a new alternative that allows the author to render and print new text in their original pen-on-paper handwriting style.

As a one-time input to our system, an end-user annotates an author's historical handwriting sample and follows a simple printer calibration procedure. The user can then ask our system to turn any new text they like into handwriting that looks like it was written by the original author. Figure 1 shows a message synthesized after our algorithm learned the handwriting style of author Sir Arthur Conan Doyle.

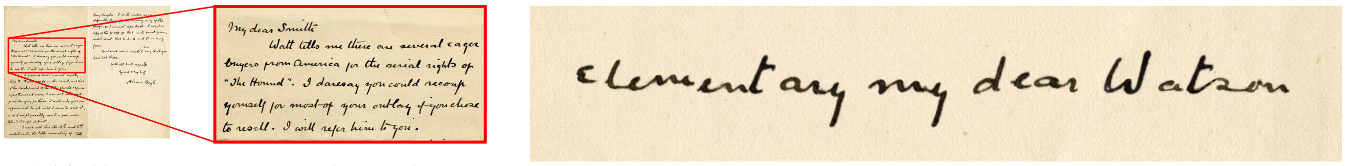
Realistic handwriting synthesis has many other practical uses. Gifts ordered from online retailers and flower shops can include personal messages, but they are rendered in an impersonal font. Handwriting synthesis completes the personalization as the gift giver could use their actual handwriting. This could also include the option of using a celebrity's handwriting. Text in films and computer games can be generated in an actor's handwriting, as they are already being hired for their likeness and voice. In computer games, the text is often dynamic and can appear in large quantities, making synthesis the only reasonable approach. A similar argument applies to the possibility of a "handwritten" mail merge, *e.g.* for Christmas cards. Comics can include handwritten lettering; synthesis allows style to be preserved during localization to a foreign language. Further, the ability to render the handwriting of living and historical authors allows for numerous creative uses, *e.g.* personalized books. Sensitive materials, such as credit cards, can be intercepted when sent through the post. Some banks are using synthesized handwriting as camouflage, which can be improved. Garner [2005] demonstrated that including a handwritten note with a survey can increase the response rate from 33% to 70% – more than double.

Our overarching contribution is the design of a system that renders new text in a specific person's pen-on-paper handwriting style. We perform a paper-based user study and demonstrate an approach that is convincing to a casual observer. No previous work has done this. Our approach is also flexible. Handwriting is simply scanned from paper samples. A tablet may be employed, but only to annotate the scanned samples. Samples may be joined-up (cursive), print (not cursive), or in-between. Natural handwriting is used, rather than filling out a grid with isolated letters/short sequences of letters [Guyon 1996]. This provides the novel ability to model the handwriting of historic figures, as demonstrated by Figure 1. The realism of the synthesized writing is measured through user-studies, in terms of how human-like it looks, its similarity to a specific person's style, and the apparent authenticity of the written document when printed on paper using a common printer. We also provide extensive qualitative results, most of which are in the video and appendices.

Our generative model is built around glyphs[1]. From training examples, the model learns to replicate an author's character choices, inter-character ligatures, pen-line texture/color, and vertical/horizontal spacing. It bears some resemblance to nonparametric texture synthesis, but handles the intricacies of handwriting. To enable synthesis, our system also includes a semi-automatic user-interface for tagging handwritten text in a training image.

Authors may be contacted at `T.Haines@cs.ucl.ac.uk`, `O.Macaodha@cs.ucl.ac.uk` and `G.Brostow@cs.ucl.ac.uk`.

[1]A glyph is a specific instance of a character, *e.g.* each font provides a different glyph to represent an "a." With handwriting, every glyph is arguably unique, though authors have a recognizable style which limits the perceived variety.


My dear Smith
Watt tells me there are several eager buyers from America for the aerial rights of "The Hound". I daresay you could recoup yourself for most of your outlay if you chose to resell. I will refer him to you.

elementary my dear Watson
Original letter
Tagged paragraph
Our synthesized result

Fig. 1. A handwriting example rendered by our system using a model created from Sir Arthur Conan Doyle's real handwriting [Doyle 1901]. One-time tagging is semi-automatic. This style is semi-decorative, making it particularly difficult to emulate. Despite its current popularity, Doyle never wrote the Sherlock catchphrase, "elementary my dear Watson." A fictitious version of our system is portrayed in the Spike Jonze film "Her" (2013), where the futuristic protagonist works for a company called "Handwritten Greeting Cards."

2. RELATED WORK

This work lies at the intersection of applied texture synthesis, font synthesis, and handwriting analysis. Its relationship with texture synthesis is discussed first, followed by handwriting research. Recently, Elarian et al. [2014] provided a wide ranging review of handwriting synthesis methods. See Figure 2 for an overview of the key handwriting terms used in this paper.

2.1 Texture Synthesis

Handwriting synthesis bears many similarities to standard texture synthesis [Wei et al. 2009]. In both cases our goal is to render plausible new images from existing examples. Early texture synthesis approaches were parametric [Lewis 1984; Portilla and Simoncelli 2000]. Once a sample could be "explained" with a mathematical model, extrapolation was straightforward [Ebert et al. 2002]. Realistic synthesis of a much larger range of textures became possible with the advent of non-parametric methods that sampled image patches from the input example [Efros and Leung 1999; Efros and Freeman 2001]. This was a major shift, but came at the expense of sacrificing artistic control via parameter adjustment.

Several texture synthesis approaches can be explicitly supervised. [Kwatra et al. 2003] simulated perspective by restricting flowers at the top of the canvas to come from scaled-down versions of the input. [Sun et al. 2005] takes user-indicated line-continuity into account when inpainting. [Bonneel et al. 2010] use an artists semantic sketch to synthesize a coherent landscape image. When synthesizing other specific content, different amounts of manual and semi-automatic labeling of the input data enable control over the rendering of faces [Mohammed et al. 2009], video [Schödl et al. 2000], motion capture [Brand and Hertzmann 2000; Cooper et al. 2007] and speech [Schroder 2001]. Our model is also a form of structured texture synthesis, where the text being rendered is parameterized by what the user types in, but the style is non-parametrically sampled from a tagged image of the original pen-on-paper handwriting sample. From a user's perspective this is ideal. The parametric information is user understandable and in their control, while the data-driven style is easy to obtain but hard to otherwise create.

2.2 General Handwriting

Handwriting remains a ubiquitous skill, taught as a cornerstone of education. It is therefore unsurprising that significant effort has gone into integrating computers to fit with our natural mode of written communication. The whole field of handwriting recognition [Plamondon and Srihari 2000] works toward this goal. Input can either be online, arriving in real time via a digital tablet, or offline, scanned later and without temporal information. We collect

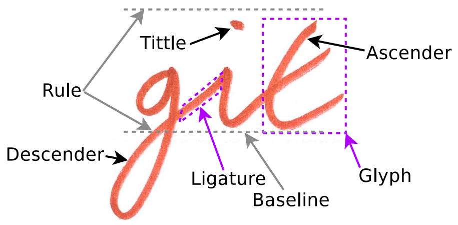


Fig. 2. Glossary of the key parts of handwritten text.

data offline. Handwriting features are analyzed at various scales, from pixel slices to timing information, if available. With features extracted, a classifier then recognizes either individual characters or entire words. While recognition is not our goal, we still need it to prepare data, where we use a pixel-slice type approach to recognize individual characters. Writer adaptation [Connell and Jain 2002] involves adjusting the pre-trained model to match a specific writer and to improve recognition accuracy. [Shilman et al. 2006] adjusted their model as the user corrected errors in the output. As with Optical Character Recognition [Antonacopoulos et al. 2007], the bigger obstacle can often be discovering and isolating the text in the first place [Huang et al. 2013]. We leave this task to a human operator.

Synthetic text generation has proven very successful for improving the results of printed word recognition in the wild [Parizi et al. 2014]. In the context of recognition, handwriting generation has also been explored for augmenting and enlarging training sets. Mao & Mohiuddin [1997] explored affine and non-linear transformations, and salt and pepper noise, to train a classifier invariant to all three. Other approaches consider interpolation between glyphs [Mori et al. 2000; Zheng and Doermann 2005] and deformations driven by cosine curves [Varga and Bunke 2003; 2004]. While classification performance can be improved, many of these deformations create unrealistic writing. Learning an active shape model [Dinges et al. 2013] might help, but requires that writing strokes be matched, which is not always possible. Our goal is not to improve handwriting recognition systems. Instead, we aim to create believable handwriting, that is indistinguishable from the original.

Identifying the author of writing for forensic purposes remains a predominantly manual activity. However, computers were used to automatically assess the validity of writer-identification [Srihari et al. 2002; Pervouchine and Leedham 2007]. Signatures remain a popular biometric, and signature verification is an interesting area of research [Gupta and Mccabe 1997]. Signature synthesis [Wan and Lin 2009] exists as its counterpoint, in an effort to break verification systems so they can be improved. CAPTCHAs [Thomas et al. 2009] have a similar arms-race for barely-legible handwriting. Signature are a special case, and not considered by the presented system.

To improve the appearance of messy tablet-written text, Zitnick [2013] beautified online handwriting by clustering short path segments and deforming them toward their means. We want output that is exactly as messy as the original handwriting. Closer to our use-case of pen-on-paper input, Mueller et al. [2013] trained a robot to do calligraphy by example.

2.3 Handwriting Generation

Since [Knuth 1979], significant effort has gone into making fonts that simulate handwriting. While [Andre and Borghi 1990; Devroye and McDougall 1995] explored adding some of the expected variability in real writing, finding a pre-existing font that matches your own style is practically impossible [Srihari et al. 2002]. Websites exist, either manual [Williams 2014] or automatic [Reinhardt 2014], which enable the creation of custom handwriting-like fonts based on the vectorization of individual glyphs. These services produce results that are obviously artificial, as joined-up writing is not supported, glyphs lack both variability and texture, and the long range interactions of real handwriting are not reproduced. Handwriting "flow," the result of adjusting letter spacing and the vertical offset to create a visually pleasing line of text that doesn't always precisely follow the rule of the page, is another property of real handwriting that is not reproduced by font engines. Methods for automatically interpolating fonts with identical topology exist [Campbell and Kautz 2014] but suffer from similar limitations. Accurately replicating an individual's handwriting with current font engines is prohibitively difficult, so we instead generate textures directly.

To overcome some of the limitations of the font-like approach, instead of capturing glyphs individually, Guyon [1996] acquired common two and three letter sequences with a tablet. However, their results appear unnatural for joined-up handwriting, as the two and three letter sequences are joined-up while the rest of the glyphs are not. However, they are, to the best of our knowledge, the only other paper to consider rendering pen ink, and use pressure and angle of travel to drive a calligraphy-like model. In our work, instead of rendering ink-like curves we sample the texture of the original pen from the paper, with the option to substitute other pens if desired. Lin & Wan [2007] asked joined-up writers to draw the ligatures without attaching them to anything. These floating ligatures are then distorted and clipped for use during synthesis. As they only made use of one glyph, they added random scale and rotation variations in an effort to hide repetition. Elarian et al. [2011; 2015] attempted to match line width when choosing which glyphs to place next to one another. We include the same idea in our optimization. As they were replicating Arabic writing, they made use of its hard ligature rules: the existence or not of a ligature is fixed by the specific letters, and ligatures are always rendered onto the baseline. Similar to our system, Chowdhury et al. [2009] accept entire sentences written on paper as input. They thresholded and converted each letter into a letter-specific spline, whereas we sample image patches containing glyphs. Their assumptions are limiting, as they assume the characters are evenly spaced and sit on the baseline exactly. No use was made of the ink texture or the flow of the text, which are both available from the handwriting samples. The *random selection* algorithm we compare against in Figure 14 and Appendix E approximates [Chowdhury et al. 2009], as it uses their placement for laying out the scanned-in glyphs.

Alternative approaches have considered modeling handwriting in a biologically plausible way, *i.e.* as a sequence of muscle movements. Models include simple pen-on-a-spring approaches [Hinton and Nair 2005], discrete firing of learned motion primitives [Williams et al. 2007], and attempts to replicate the high-level structure of the human brain [Gangadhar et al. 2007]. Such methods do not yet generate realistic output, but strive to validate biological models. They fail to take the various feedback mechanisms into account, and often lose the subtle details important to an individuals style.

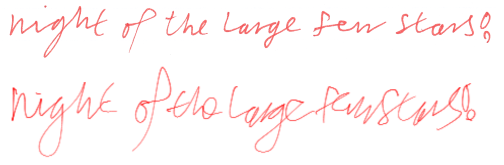


Fig. 3. Two lines of text written by the same author – the top using a real pen, the bottom a graphics tablet. Note that the author in question is an experienced tablet user.

Graves [2013] outputs the path of a pen without biological considerations. It is one of the few approaches beside ours to tackle individual style and to support joined-up input/output. Using a neural network with memory, it was trained on many different writing styles, as captured by a tablet. It can then output the pen path for a user provided phrase. It can be forced to output a specific writing style, and the results are good, though obviously limited to tablet-writing.

Wang et al. [2002] fitted splines and matched glyphs together to learn Gaussian distributions over position. They extracted tail and head regions from the glyphs, which were entered as complete sentences, and then joined glyphs together using splines. Individual specific models were learned, but their noise model is unrealistic, such that outputs appear messier than the author's true handwriting. Choi et al. [2004] presented a similar hierarchical Gaussian graphical model with the same issues. In a follow up to their earlier work, Wang et al. [2004] used active shape models instead of Gaussian distributions to help resolve the messiness issue. Additionally, a delta log-normal model [Guerfali and Plamondon 1995], commonly found in biologically inspired synthesis, was used to synthesize ligatures, and a pseudo-Gibbs sampling approach was used to select a set of visually cohesive glyphs from the active shape model. All text has equal spacing though, and is always placed directly on the rule. Our approach varies the spacing, learning models from the data, and lets glyphs leave the rule if that is consistent with the author's style.

Character based languages, such as Chinese, require different considerations for 2D character layout. Wang et al. [2008] used their capture-then-arrange method to arrange individual strokes, driven by a font-like specification of the strokes required by each word to make single Chinese characters. Consequently, they can generate all Chinese characters despite having very few samples. Given the 2D structure of characters, their spatial relationship model is particularly advanced, allowing them to capture some complex relationships. Chang & Shin [2012] simulated multiple languages. While only considering individual glyphs, they included a 2D layout model for strokes.

Except for the works that augment handwriting recognition datasets and Chowdhury et al. [2009], all of the above approaches use graphics tablets for input. Due to low friction and a lack of visual feedback, writing on a tablet is not the same as writing on paper, as handwriting is distorted, as shown in Figure 3. The effect is particularly strong in terms of pen pressure [van Galen 1991],

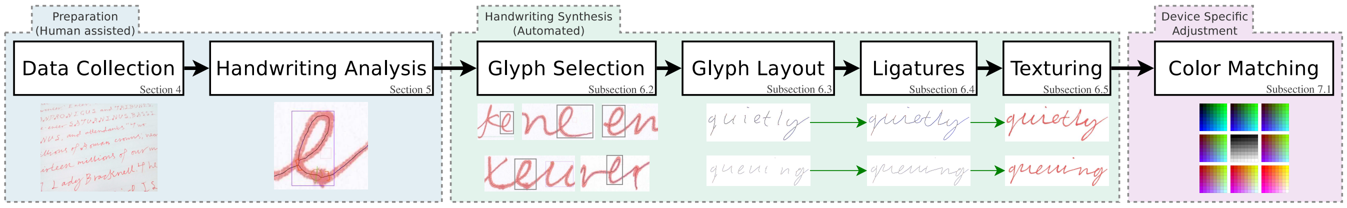


Fig. 4. System diagram showing our processing pipeline, with representative images for each stage. After samples are collected and analyzed, the rendering system selects a glyph to represent each character, *e.g.* "e" as shown here. If there are many choices, it must choose one that fits the surrounding text. The glyphs are then positioned on the page, and ligatures added if the author uses joined up writing. Two example words are given for these three stages, "quietly" and "queuing." Finally, the texture is transfered from the original input to the vector output and, if being printed, color correction is applied.

so current virtual-pen renderers [Lee et al. 2006; Lu et al. 2012] would require considerable extensions to achieve our pen-on-paper look. We validate the realism of our rendered handwriting with user studies, which, to date, have been neglected in the literature. Chang & Shin [2012] is the single exception, as they used a user study to calibrate how much they could distort writing before it is no longer the same style. See Appendix G for examples of results generated by competing approaches.

3. SYSTEM OVERVIEW

Given a sample of an individual's handwriting, the goal is to render realistic looking handwriting in the same style, with user specified words. At a high level, the system reuses glyphs provided by an author and analyzed through our interface, manipulating them to obtain believable output; see Figure 4. There are two phases: First the author's sample, A, is captured and analyzed. Second, the model renders each user-provided target text t as a "handwritten" output, R. Finally, additional post-processing may occur, depending on the use case.

The input to the system is a sample of an author's digitized writing, *e.g.* a scanned document written using a fountain pen; see Section 4. For cooperative authors, the process can be optimized to obtain sufficient samples with the least manual effort, as detailed in Subsection 4.1. Before synthesis can occur, the sample must be analyzed. Tagging is semi-automated, and consists of extracting the path of the pen, and then segmenting it into labeled ligatures and individual glyphs. The rule the words are written on is also extracted, and the original texture alpha matted, so it may be composited with an arbitrary background. See Figure 5 for a summary. Section 5 explains our new handwriting analysis procedure and the assistive algorithms that make it easy to add a new author's style to the database. Once an author's writing sample is analyzed, it can be used to render new text as "handwriting." This is the core of our method and is described in Section 6. Optional post-processing is detailed in Section 7.

4. DATA COLLECTION

For each author, a sample of their handwriting is required to drive our model. The captured handwriting must be representative of the author's style. There are two scenarios:

—In historical cases, the author's writing samples are fixed to those which have survived history. If there is no "Q" in the source text, the later stages of the system will be unable to synthesize a "Q." Depending on the target text, this may or may not be a concern.

—The interesting case is when the author is available. We can generate the source text for the author to write out, optimizing it to maximize the algorithm's ability to generate hand written text from a target corpus. This is the subject of the following subsection.

In our standard experiments, the total sample is approximately four A4-sized pages, using every other line so that descenders do not overlap with the ascenders on the line below. Authors took 10 – 30 minutes to write out the source. A sample is then scanned at 600 dpi and automatically analyzed with our human-in-the-loop interface (Section 5). See supplementary video for more details, and Appendix I for an example.

4.1 Source Text Optimization

We optimize the source text to be written out by the author so we can obtain sufficient samples with the least effort. In this collaborative scenario, our aim is to construct a source text that includes the most commonly used letter combinations, but is short enough to not burden the author unduly. Obviously, for a short text, the user wouldn't use a computer at all. The source text consists of complete sentences to be written by the author on the back of single-sided ruled paper, in the pen of their choosing. They can see the ruled lines through the paper, without the lines appearing when it is digitized. The sentences that comprise the source text are selected automatically from a larger source corpus of text. We used the top 100 English language books on project Gutenberg [Project Gutenberg 2014] as the source corpus. Certain volumes were removed for not being work-safe, containing foreign text, or representing accents textually. These last two are an issue of flow. It was found that if the author is reproducing something unfamiliar, they will pause frequently, which affects their handwriting.

The process proceeds by greedily selecting the best sentence S from the corpus, iteratively. A target corpus defines the best, in terms of generating a source text that matches the unary and pairwise statistics of the target. For example, a corpus of gift greetings would be used in the case of the card scenario. If no target text is specified (our default assumption), we use the same Gutenberg corpus as our target corpus. Our next-best sentence-selection cost function, $C_B(S)$, contains two terms that take account of, respectively, the unary and pairwise neighbor occurrence of letters in the target-corpus,

$$C_B(S) = \sum_{c \in S} \frac{1 + F(c)^2}{Pr(c)} + \omega \sum_{\{c_1, c_2\} \in S} \frac{1 + F(c_1, c_2)^2}{Pr(c_1, c_2)}. \quad (1)$$

S is the sentence under consideration to become a source-sentence; the lowest energy one will be appended to the source text at each iteration. $Pr(c)$ is the probability of seeing letter c in the target corpus, and $Pr(c_1, c_2)$ is the probability of seeing c_1 followed by c_2. $F(\ldots)$ provides the number of times the given letter or letter pair has been selected thus far. ω weights the two terms, and is set to $\frac{1}{32}$. Ignoring the pairwise term, the cost of each sentence on the first selection is approximately the letter cost of the sentence in Scrabble [Norvig 2014][2]. Successive selection iterations emphasize obtaining glyphs that have not yet been collected, until a four-page limit is reached. This procedure can be thought of as selecting a good Scrabble hand, with the additional consideration of pairwise relationships. For experimental reasons, we used a randomized subset of the source corpus each time, to obtain different sentences from each author.

5. HANDWRITING ANALYSIS

After obtaining data, there are two parts to our handwriting analysis. They are i) *automatic tagging* of those samples (Subsection 5.1), and ii) *human assistance* for correcting the automatic tags, as needed (Subsection 5.2). Though they can be the same person, we distinguish between the role of the *author*, who wrote the handwriting sample being emulated, versus the *user*, who is interacting with our system to render new images of handwriting. Similarly, the *source* text is the UTF-8 transcript of characters that appear in the author's sample, and the *target* text is that which the user wishes to render. We refer to all characters, symbols, and digits simply as letters.

5.1 Automatic tagging

For each sample page of handwriting, the goal is to segment the ink from the scanned image, and tag pixels with their associated source text letters. Tablet-based sample acquisition would be simpler, but captures visually unnatural training data without texture. From our scans of pen-on-paper, the real pen trajectory and timing will be ambiguous, but the glyph characteristics of ink-appearance, radius, and density suffice for our synthesis purposes. Errors in the automatic tagging will show up as errors in the synthesized output. An interactive user-tagging stage follows the automatic one, to fix any errors, as detailed in Subsection 5.2. We now proceed with automatic segmentation of the ink, followed by calculating all the needed ink and glyph characteristics.

5.1.1 *Ink Extraction.* We assume that most pixels in the image depict either the paper or the ink color. (We use *ink* throughout, though it may in fact be graphite (pencil) or another material.) Mean shift [Fukunaga and Hostetler 1975; Comaniciu and Meer 2002] is used with a Gaussian kernel to identify the two largest modes in RGB space. Instead of just a binary classification, we seek to model ink density on a continuous scale, to capture the variation in pressure as the author writes. A line passing through both modes is defined, and the color of each pixel projected onto the line. The half-way point between the paper and ink color modes, which would otherwise mark a classifier's decision boundary, is treated as zero ink, and the ink color mode treated as ink with strength one. Linear interpolation maps every pixel to an ink level, but ink is deposited by different pens according to different density profiles. We want different pens to be comparable, for ink replacement (Subsection 7.2), so we distort the density profiles to all be identical.

[2]Scrabble's letter costs were derived from the letter frequencies of the front page of *The New York Times* $\sim$ 1938, and subjected to human tweaking [History of Scrabble 2014] and discretization. Our cost is better tuned by using a much larger corpus directly.

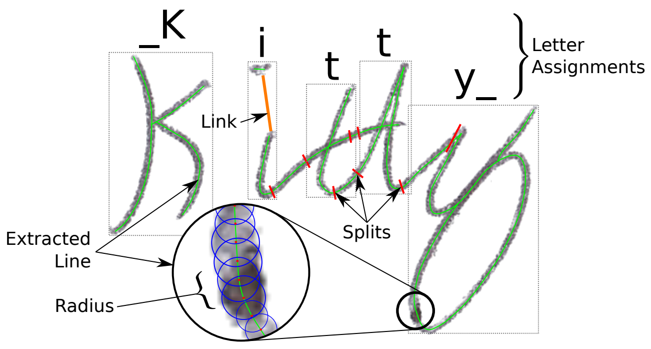


Fig. 5. Visualization of the output of the tagging process. The *line* is vectorized, and has *radius* calculated at every pixel along it. Not shown is the ink *density*, which is also calculated for every pixel as the average within the circle defined by the radius. The line forms a graph, to which *splits* are added to delineate each letter and ligature. If a letter has multiple parts then a *link* combines them. Finally, the parts are *tagged* with the relevant letter/digit/punctuation and *underscores* used to indicate the starts and ends of words. *Ligatures* are left implicit, as any path that connects two tagged glyphs.

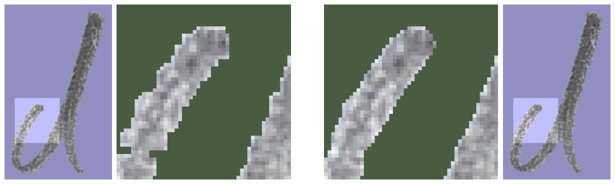


Fig. 6. Segmentation on the left shows the graph cuts result without line aware smoothing. The right side shows the improvement of smoothing. Writing implement is a pencil.

Using the same kernel as for mean shift, a kernel density estimate is applied to the ink levels. The cumulative distribution function of the estimated density is then used to generate a non-linear mapping to a uniform distribution. This fixes the density to 0 for the half way point and 1 for the color of the ink mode in the original distribution, with an even distribution between the two extremes, regardless of pen type. Pixels projecting higher than 1 could potentially extend to arbitrarily high density values, so density values above 2 are clamped, as are values below 0. (This occurs as a result of regularization and the user overriding the ink segmentation.) To efficiently apply this model to the entire image, it is converted into a thin plate spline [Bookstein 1989], that directly converts RGB values to the final ink density values.

The ink density map is used to segment the line from the background, by expressing the problem as a binary Markov random field (Ising model) and solving with graph cuts; see Appendix A.

5.1.2 *Density, Radius & Matting.* The segmentation thus far has no concept of pen-stroke lines. Mapping ink pixels to lines, essentially vectorizing of bitmaps, is comparatively simple for high quality pens that deposit ink smoothly and uniformly. Unfortunately, the two most popular writing implements are cheap ballpoint pens and pencils. These have rough lines, so as demonstrated in Figure 6, a line-aware regularization method is used. Other approaches, such as bilateral smoothing, will break lines that are only a few pixels wide. See Appendix B for details.

The smoothed mask is converted into a line with skeletonization [Blum 1967]. Islands smaller than a period are deleted (threshold on pixel count). The thinning of Zhang and Suen [1984] is used, and the output represented as a graph – junctions exist wherever the line crosses itself. The graph is a piecewise linear representation of the path the pen took, sampled at every pixel. No attempt is made to resolve the ambiguity of which way the pen went at junctions.

Ink-line radius along the line is set to the largest circle that can be drawn at that location without violating the mask. This tends to underestimate radius, but this is factored into later processing steps. Line density is simply set to the average density value in the inscribed circle. The ink-line paths with radius and density information are used later as features to match glyphs and connecting ligatures, in aid of handwriting synthesis.

Though ink density has been computed, an alpha matte is still needed so that the output can be composited onto a background of the user's choosing. An image with transparency is obtained by replacing the ink through inpainting, then using the standard alpha-over compositing equation [Porter and Duff 1984] to setup a linear equation for each pixel. See Appendix C for details.

5.1.3 *Tagging.* Finally, three types of tags are assigned automatically: 1) a label, indicating which letter the glyph represents. The labels also indicate if the character is at the start or end of a word, 2) a split, indicating the point where a line transitions from glyph to ligature, and 3) a link, indicating that two separate lines are part of the same glyph (*e.g.* the tittle and stem of an "i").

Compared to normal handwriting recognition, this is an unusual scenario because the source text is known, and just the transition points are unknown. An approach broadly similar to [Marti and Bunke 2002] is used here, but with different features and the output constrained to match the available transcript. The features are documented in Subsection 7.3; they are histogram-type features defined for every pixel on the extracted line of the glyph.

Using manually tagged data (see Section 5.2), a random forest [Ho 1995; Breiman 2001] classifier is learned[3]. 64 trees are used[4], obtaining an out of bag success rate of 85% on 96 classes. The classes are English upper and lower case letters, with digits, common punctuation, and ligatures. This accuracy is an overestimate, as the training data is not independent, but the constraint of knowing the true text means even a weak classifier would be sufficient.

The extracted ink line is a graph containing loops, so conforming a labeling to obey the known sequential transcript requires it be linearized. Linearization is done identically to [Marti and Bunke 2002], where the rule on which the text is written is split into 1 pixel wide vertical slices, with all features in each slice combined to create a Markov chain. To enforce the known source text, the labels of the Markov chain are positions in the transcript, the transition costs are then zero to stay on the same label or move to the next, but infinity to move elsewhere. Unary terms are obtained by indexing the probability of each character, converted to negative log likelihood, by the position in the transcript associated with that label. The first entry in the chain is required to be the first label and the last entry the last label. Note that in addition to the characters of the transcript, the labels are augmented with a spacer label between each character pair to represent a ligature/space. The maximum a posteriori (MAP) assignment is found using dynamic programming.

[3]As the training set grows and more handwriting is analyzed, the random forest is retrained, so it gets better with time.

[4]No depth or leaf size limits were imposed, and the number of splits tested at each node was the square root of the feature count.

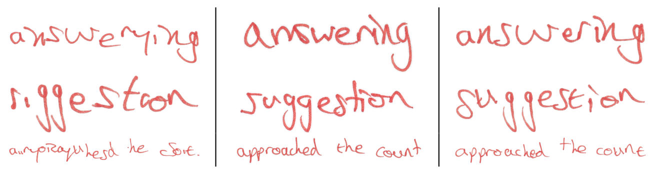


Fig. 7. Comparison of automatic only (left column) with human assisted tagging (right column). Ground truth, a sample of the authors actual writing, is in the central column.

The three types of tags must now be assigned to the extracted ink line. Labels are assigned to a point on the line that is closest to the baseline within the slice at the center of a labeled region. Splits are then added at all lines that cross the transition between regions, but only when the line connects the labels of the adjacent regions. This avoids splitting the tails of glyphs that travel underneath another character, or chopping up writing with a strong slant. Consequently, it is quite conservative at introducing splits – a missing split is the most common problem requiring human assistance. This is a fast fix however, so it is preferred to bias towards missing splits rather than splits that are wrong. Finally, any subgraph that is still not labeled, is linked to its nearest labeled subgraph. This is primarily to link the tittle of an "i" to its stem; it can fail if an author is imprecise with their tittle placement, but that is an issue resolved with human assistance.

5.2 Human Assistance

The automatic segmentation and labeling described saves the user substantial effort. However, mistakes in the output are immediately obvious, such as a wrong letter, a badly rendered line, and unusual ligatures. Figure 7 demonstrates this. For this reason, convincing handwriting requires a human user to perform the one time task of interactively correcting mistakes, as the presented algorithmic approach is not flawless. Additionally, there is a first step in our user interface that is completely manual. Namely, the user indicates the rotation and spacing of virtual ruled lines to match the rule of the handwritten sample. For the user, this means click-dragging on screen. The ruled lines account for scale differences between different samples of the same author, and specify the coordinate frame in which rendering will ultimately occur.

Problems with ink segmentation can occur, particularly for authors with long flowing tails, as the pen pressure decreases and trails off along the length of the tail. Also, low quality pens often have unreliable ink deposition, and gaps can appear that have to be filled in. Low quality historic documents can have dirt and damage that needs to be removed. A straightforward painting interface is provided, allowing a human to paint areas that must belong to the foreground or must belong to the background. They can iteratively update this to correct errors in the segmentation, similarly to other segmentation approaches [Boykov and Jolly 2001; Rother et al. 2004].

All tags can be edited, in support of the automatic glyph recognition. See Appendix D for a description of this work flow, plus an analysis of how much time the automatic tagging saves. The supplementary material includes a video of the interface in use.

6. HANDWRITING SYNTHESIS

Given a database of tagged writing samples (Section 5) for a specific author, the system can render an arbitrary quantity of target text in their handwriting. To summarize, the database consists of multiple scanned samples of an author's handwriting, where the path of the pen is extracted as a set of lines. These lines include ink density and radius information as shown in Figure 5. Density and radius are a function of the variable pressure applied when writing. Additionally, the line is split into segments which are labeled as a specific letter, digit, punctuator, or as a ligature. The presented approach can be broken down, and individual parts switched off; Appendix F qualitatively demonstrates how performance improves as each feature is enabled.

6.1 Objective

Given A, an author's handwriting sample and t, user-provided target text, the system generates R, the rendered output. This can be thought of as an optimization that minimizes a cost function,

$$\begin{aligned} &C(R,t,A) = \\ &G_A(g,t) + S_A(g,x) + L_A(g,x,l) + T_A(g,x,l,R). \end{aligned} \tag{2}$$

The four terms take parameters s, x and l, and are

—Selection of glyphs, $G_A(g,t)$: Constrains the rendered output, R, to *select* glyphs $g \subset A$, such that the output matches the user-provided output text, t. This cost has two parts. Firstly, it penalizes choosing the wrong letter with a cost of infinity, as misspellings are unacceptable. Secondly, it penalizes using glyphs that are a poor match, *e.g.* starting a word with a glyph taken from the middle of a word, as the glyphs at the start of a word look different to those in the middle [Srihari et al. 2002].

—Spacing of glyphs $S_A(g,x)$: Optimizes the positions, x, of the glyphs on the page, to match the author's style in terms of both the vertical and horizontal *spacing* between adjacent glyphs. This is to replicate the author's *flow*, including keeping each row of text on the baseline while allowing drifting away from it before returning, as typical of handwriting (arcs form as the hand pivots, instead of sliding).

—Ligature use $L_A(g,x,l)$: Penalizes rendering ligatures between glyphs if the author's writing in A was not joined-up, and similarly penalizes generating glyphs that should be joined-up but aren't. The set of ligatures to generate is represented by l.

—Texture rendering $T_A(g,x,l,R)$: Minimizes the difference between the output image and the original glyph texture, while ensuring continuity of the pen path. It avoids unnatural discontinuities caused by changes in pressure, radius, or velocity.

Their purpose is illustrated in Figure 8. After computing the optimized texture R, color correction may be applied to match the look of ink on the final output device. Note that while presented as a minimization problem, variability is introduced to avoid repetition. This is necessary as humans are sensitive to the repetition that would otherwise occur, leaving it obvious that the text is fake.

Optimizing the above cost function directly is problematic, as the individual terms would have to be overly simplistic, or the complexity would be too high. Instead, each term is optimized greedily in the above order, with proxy terms used to approximate the remaining terms at each of the four stages. Such an approach allows complex costs to be used for each term, as long as proxy terms are selected so that efficient optimization methods may be used. As an example, when optimizing $G_A(g,t)$, the texture term ($T_A(g,x,l,R)$) is approximated as the difference between the average density/radius of adjacent glyphs. This allows for efficient optimization (as it may be represented by a pairwise term, for dynamic programming). When actually optimizing $T_A(g,x,l,R)$, it is calculated pixel-wise based on color differences, and optimized with

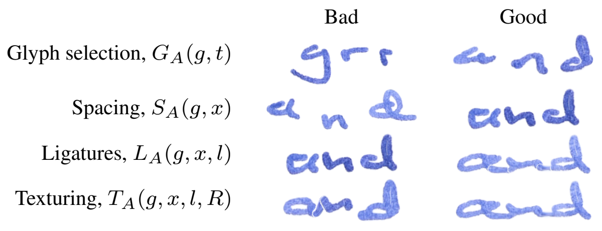


Fig. 8. Synthesis of the word "and", demonstrating what each component of the cost function does. All steps are dependent on the author's specific style, e.g. here we show that having ligatures is preferred, but if the author has print handwriting then the inclusion of ligatures would be wrong. Note that the last two *good* exemplars are different, as visible between the "a" and "n."

graph cut textures. Because the earlier step selected glyphs using a proxy for this later term, the pixels will match reasonably well, and the texture optimization will never have the impossible task of blending a sudden change in line density/radius. This pipeline is illustrated in Figure 4. The following subsections explain the procedure, and are organized to match with the four terms in (2). Figure 9 gives a step by step overview of the entire process.

Relation to Fonts At a high level, this cost function can be compared to the processing of a font engine [Knuth 1979; Warnock et al. 1984; Werner 1999; Correll 2000]. The glyph selection term, $G_A(g,t)$, for a font is typically a one-to-one function, while we have a one-to-many function, and must optimize the glyph selection. Some font engines support a random mapping [Andre and Borghi 1990], while others support human-specified context dependence [Werner 1999]. Spacing, $S_A(g,x)$, in a font engine takes kerning into account, and only adjusts horizontal spacing. For handwriting, spacing is considerably more variable, and includes a vertical component as authors drift from the baseline. Font systems, with a few exceptions [Correll 2000], do not support ligatures, as required for handwriting and expressed in our system with the ligature term, $L_A(g,x,l)$. Finally, a font engine rasterizes the font as a bitmap, either from splines representing the outline [Warnock et al. 1984] or mathematical equations representing the path of the pen [Knuth 1979]. For handwriting, we need to replicate the texture of the writing implement, $T_A(g,x,l,R)$, including subtle density and radius variations.

6.2 Glyph Selection

The first step is to select glyphs, $g \subset A$, from the author's sample to match the desired output text, t. This is the optimization of $G_A(g,t)$ from the cost function (2). It is trivial to find all matching glyphs in the database and randomly select one for each character, but this does not take into account the rest of the cost function. Instead, all glyphs for each letter are considered as potential candidates, with the rest of the cost function represented temporarily in proxy form by a pairwise cost term. Dynamic programming then allows us to select an optimized set of glyphs.

We are solving a dynamic programming [Bellman 1952] (DP) problem of the form

$$C_{dp}(g) = \sum_{i \in t} U(g_i) + \sum_{i,i+1 \in t} P(g_i, g_{i+1}). \tag{3}$$

It is an instance of DP because the solution for t from neighboring glyphs g_0 to g_{i+1} characters can be found in constant time, given the solution for t from g_0 to g_i characters [Bellman 1952]. $U(g_i)$ represents the *unary term* and $P(g_i, g_{i+1})$ the *pairwise term*.

```
(1) Select glyphs using dynamic
 programming:
 (Subsection 6.2)
 ---Unary cost encourages sensible glyph
    choice.
 ---Pairwise cost ensures adjacent glyphs
    can flow (spacing) and generate a
    coherent line (texture).
(2) Draw horizontal spacing from uniform
 distributions.
 (Subsection 6.3)
(3) Optimize vertical spacing with Kalman
 smoothing: (Dynamic programming with
 Gaussian distributions)
 (Subsection 6.3)
 ---Unary cost encourages glyphs to stay
    on the line.
 ---Pairwise cost encourages glyphs to
    have the vertical offsets seen in the
    training data.
(4) Add ligatures if found in author's
 sample.
 (Subsection 6.4)
(5) Optional line replacement. Break line
 into short segments and replace with
 short segments extracted from target line
 style.
 (Subsection 7.2)
(6) Merge all line segments into a single
 texture using graph cut textures.
 (Subsection 6.5)
(7) Optional color correction.
 (Subsection 7.1)
```

Fig. 9. Steps of the presented algorithm. Note that both of the pairwise costs may use random forests to estimate their costs, with the random forest being used for distance learning in the first use (step 1) and as a regression forest in the second (step 3).

6.2.1 *Unary.* The unary term, $U(g_i)$, is set to $G_A(g, t)$. It is 0 for exact matches, but inexact matches are also considered if there are fewer than 8 exact matches available. An exact match is when a glyph is the correct letter with the same case and from the same placement in a word. The first and last letters of words are written differently [Srihari et al. 2002], so glyphs are tagged with their word placement – start of word, middle of word, or end of word. In rare cases two types of inexact match are allowed. Using a glyph with the wrong placement is accepted with a cost of $\frac{\eta}{2}$. Lower case letters can replace upper case letters, at a cost of η. These costs are high and such replacements are intended to be rare. Mismatches mainly occur when working with small handwriting samples, as it is preferable for the output to remain readable even if there are no exact matches. There are also cases when an inexact match may be selected over an exact match due to the other cost terms, *e.g.* a large difference in writing pressure.

6.2.2 *Pairwise.* The pairwise term, $P(g_i, g_{i+1})$, represents the three remaining cost terms in (2), specifically spacing, ligatures and texture. For all three a *proxy* is used: an approximation of the real term, such that we can efficiently optimize glyph selection while simultaneously selecting a set of glyphs that will result in a low overall cost after all parts of (2) are optimized. The true costs are defined in Sections 6.3-6.5, when they are subsequently optimized.

Spacing and ligatures are, at this stage, considered together. Most authors either write entirely joined up, or entirely not joined up (print handwriting) [Srihari et al. 2002]. Some write partially joined up, but in such cases the tendency is for specific letters to act as breaks, *e.g.* never connecting an "a" to the preceding letter with a ligature. More complex behaviors do exist, often as letter-pair specific exceptions to the above generalization. Unfortunately they require excessive data capture to model accurately, due to the combinatorial number of samples required. Hence, it is reasonable to assume that if both source glyphs have ligatures, the output should. If either glyph is missing a ligature then the output should omit generating one. The pairwise cost function assigns a cost of zero to selecting glyphs that were adjacent in the input, so these are likely to be selected. Short words such as "the" can be copied over wholesale. Consequently, unusual ligatures between specific characters can be used, which helps to maintain an author's style for partially joined up and other unusual styles. We now define the pairwise cost for the two scenarios – when both glyphs have ligatures (joined up) and when one of them is missing a ligature (partially joined up or print handwriting).

Two Ligatures Where two adjacent glyphs both have ligatures towards the other, the spacing is set so their ligatures overlap and can create a smooth transition when they are interpolated. The cost is a measure of how smooth a transition can be obtained. Each ligature has a start position, s, and end position, e, as shown in Figure 11 C); subscript $i + 1$ refers to the second of the two glyphs. The pairwise cost is then the distance between points that will be coincident after interpolation, for the positioning, p, that minimizes this distance,

$$\begin{aligned} P(g_i, g_{i+1}) &= T_{dp}(g_i, g_{i+1}) + \\ &\eta \times \underset{p}{\operatorname{argmin}}(|s_i - (e_{i+1} + p)| + |e_i - (s_{i+1} + p)|), \end{aligned} \tag{4}$$

where it is trivial to show that $p = 0.5(s_i + e_i - s_{i+1} - e_{i+1})$ is a minimum. This is calculated in line space, such that the height of the rule on which the text is written is 1. η is a scale factor, appearing throughout, that can be set to any positive constant. $T_{dp}(g_i, g_{i+1})$ is the texturing term defined below.

Missing Ligatures In the case that either glyph lacks an appropriate ligature, an alternate cost is required. This cost must be equivalent to the joined up cost, as otherwise *partially joined up* authors would be biased towards either joined up or not joined up, depending on which cost calculation method was typically lower. The system hallucinates ligatures where none exist[5]. This comes from the idea that while the pen may not be touching the page, it must travel to the next letter. Humans are fundamentally conservative in their motion [Alexander 1984], so the path of the pen is likely to be similar regardless of the presence or not of a ligature[6]. Additionally, we need a between-word pairwise term, otherwise the transition from word to word may not appear visually coherent. The same argument that the pen must travel from one word to another applies, and the missing ligature pairwise term is used again.

[5]The hallucinated ligatures are only used for cost estimation, and not rendered.

[6]This principal is unlikely to apply to a child who is still learning to write. It is expected that in such cases the output will appear too neat for the author.

Specifically, the cost (4) from the last glyph of the first word to the first glyph of the second word is used. As the distance between words is larger, the author has more space to adjust the path of the pen, so the cost is down weighted to half; this was found to improve the quality of the output.

Given the equivalence requirement, we use *distance learning*. This allows us to learn the cost, using the *two ligature* examples in the training set, and *transfer* it to the *missing ligature* cases. The distance learning technique of Xiong et al. [2012] is used; it makes use of regression forests [Breiman 2001; Criminisi and Shotton 2013]. Feature vectors are defined for every pair of adjacent glyphs, regardless of if they have ligatures or not. A regression forest learns the costs (4), of the glyph pairs for which costs are available, *i.e.* those with ligatures. Ligatures are generated, *i.e. transfered* [Pan and Yang 2010], by evaluating the regression forest on the glyph pairs that are missing ligatures. To obtain sufficient training exemplars, particularly for non-joined up authors, samples from many authors are used for training. This is arguably transferring style between authors, but it is only choosing glyphs that appear natural when placed side by side, and acting as a proxy for the real spacing approach, given in Subsection 6.3.

The distance learning/regression forest requires a feature vector for each glyph. We use the same feature as earlier, for tagging; see Subsection 7.3 for details. Text requires order dependence, *e.g.* the ligature of "ab" is typically different from the ligature of "ba." Xiong et al. [2012] lacks order dependence, so we modify their feature construction approach to obtain it by splitting the letters into halves. The feature used is defined for every vertex on the pen line. It is summed for all pixels in each vertical half of the glyph and each half renormalized to sum to one, consistent with the features being histograms. The notation is given in Figure 11 A): h^n is the feature for the half of the glyph next to the (potentially hallucinated) ligature, and h^f the feature for the half of the glyph not adjacent to the (potentially hallucinated) ligature. Whether ligatures exist or not must not affect the feature vector, so ligatures are excluded from feature calculation. The feature vector given to the distance learning/regression forest is then $[h^n_i + h^n_{i+1}, h^f_i + h^f_{i+1}, \mathrm{abs}(h^n_i - h^n_{i+1}), \mathrm{abs}(h^f_i - h^f_{i+1})]^T$, where $\mathrm{abs}(\cdot)$ is defined element-wise.

Texture Proxy The texturing term, $T_{dp}(g_i, g_{i+1})$ from (4), is an approximation of $T_A(g, x, l, R)$ from (2). It represents the need to select glyphs that will not look anomalous when positioned nearby, and can have their textures blended to create a coherent line when there are ligatures. For each glyph, we calculate its average density and radius for all vertices in the extracted line. The cost is then the sum of the absolute differences, for both average density and average line radius, which is then multiplied by η. This is a bias towards selecting glyphs that have similar pen-on-paper properties. We have assumed that the large scale properties of the writing implement vary slowly across the page, *e.g.* authors don't suddenly change how much pressure they are applying while writing.

6.2.3 *Avoiding Repetition.* Repetitions in the text often result in repetitions in the selected glyphs, which is noticeable to an observer and gives away that the synthesized handwriting is fake[7]. Using dynamic programming, with the given unary and pairwise cost terms, the optimal set of glyphs can be selected, but this will generate such repetitions. Randomness in the generated output is introduced by interpreting the cost function (3) as a negative log probability, $\exp(-C_{dp}(g))$, and drawing from the distribution rather

[7]While the selected glyphs are dependent on the context of the repeated text, and hence will vary, certain words such as "the" appear often.

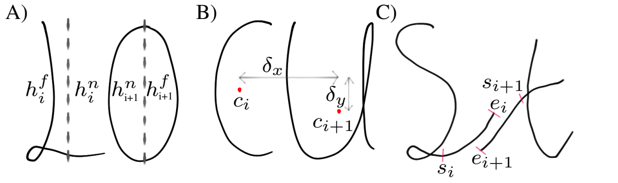


Fig. 11. A) The near-far labeling scheme for halves of a glyph, as used for the distance learning feature. B) Two glyphs, "c" and "u", with their centers (c_i and c_{i+1}) and the horizontal (δ_x) and vertical (δ_y) offsets between them marked. C) Two glyphs, "s" and "t", with attached ligatures delineated by the regions within the red marks. The ligature end points are labeled with the variables used for their coordinates.

than maximizing. There are multiple approaches to performing this draw; the approach of [Papandreou and Yuille 2011] proves convenient as it involves a trivial modification of the cost function without changing the actual DP implementation. Specifically, Gumbel noise with scale 1 and location 0 is added to the unary terms before finding the MAP solution,

$$U'(g_i) = U(g_i) + \lambda u, \quad u \sim \mathrm{Gumbel}(0,1), \tag{5}$$

$$u \sim \mathrm{Gumbel}(0,1) \Leftrightarrow u = \log(-\log(v)), \quad v \sim \mathrm{uniform}(0,1). \tag{6}$$

The pairwise term remains the same. λ is principally 1, but varying it is equivalent to scaling the entire cost function, $C_{dp}(g)$, and adjusting the tightness of the probability distribution function from which we are drawing. Consequently, the system generates low cost solutions with a small amount of variability, which is tunable by scaling λ. With $\lambda = 0$ the randomness is switched off; as it is increased, so does variability, until the samples no longer look like the author's handwriting. A value of $\lambda = 10\eta$ was found to be a good compromise; examples are shown in Figure 10 A).

6.3 Glyph Layout

With the glyphs selected, their relative position in the output is then optimized. This is the optimization of the spacing term, $S_A(g, x)$, which is separated into the *horizontal* and *vertical* offsets between adjacent glyphs. The texture term of the cost function is irrelevant during this optimization step, so no proxy is needed.

Each glyph needs a center, c_i, for the offsets to be relative to. See Figure 11 B). The average position of each glyph's line vertices is used, weighted by density and radius squared. For robustness, line vertices below the line (*e.g.* long descenders such as on a "y") and beyond the bounds of their neighbors (*e.g.* long crosses on a "t") are excluded. Position is then calculated in terms of offsets from this assigned center. The center needs to be assigned consistently, and while this approach is reliable for typical glyphs, it can cause problems for punctuation. As described below, vertical spacing uses a feature vector, but it is unable to distinguish different punctuation types, *e.g.* a comma from an apostrophe. Consequently, it confuses punctuation and positions them incorrectly, at the average height of all punctuation. This is resolved by special casing punctuation, by assigning the same height above the baseline to all punctuation.

Horizontal spacing, δ_x, is driven by the original position of adjacent glyphs in the author's handwriting sample. Each glyph in an adjacent source pair provides an estimate of the offset, by reference to its assigned center. Output spacing is set to a uniform draw from the range of offset estimates from adjacent glyphs. For a pair that were originally neighbors, the range would be zero, so the range is extended slightly. This introduces randomness, which helps avoid human detectable patterns. It also gives the flow step

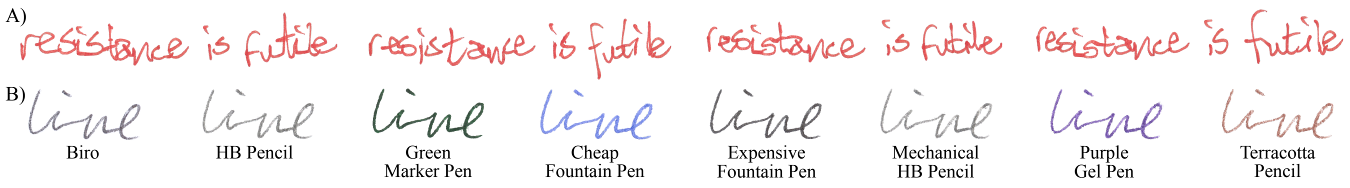


Fig. 10. A) Same sentence generated multiple times, to demonstrate output variability. The first instance is the maximum likelihood output, which would otherwise be returned every time. B) Line replacement, where one writing implement is replaced with another. First on the left is the source with the original texture. While the replacement is visually coherent, the geometric path and density variability of the previous pen is kept, which is sometimes unrealistic.

(see below) the freedom to distort the letter pair back towards the baseline. Source spacing is not always correct, due to the case when a letter is used out of position, *e.g.* the first letter of a word is used in the middle of a word. In such cases it will use only one estimate, or if no estimates are available, a draw from a Gaussian distribution parameterized by the mean and variance of all adjacent glyph estimates in the data set. There are two such estimates, one for the gap between letters and one for the gap between words. These scenarios again only occur when limited source text is available.

Vertical spacing, δ_y, depends on the ligature term, $L_A(g, x, l)$. When both glyphs have ligatures, the vertical spacing is set to minimize the distortion required of the ligatures. As ligatures cannot be matched precisely, a Gaussian distribution is placed over the vertical spacing, such that a draw has a 95% chance of being between the offsets implied by matching the start/end points of ligatures. With reference to Figure 11 C), using the vertical ordinate only,

$$\delta_y(i \rightarrow i+1) \sim \mathcal{N}\left(\frac{(e_{i+1}-s_i)+(s_{i+1}-e_i)}{2}, \frac{|(e_{i+1}-s_i)-(s_{i+1}-e_i)|}{4}\right) \quad (7)$$

where $\delta_y(i \rightarrow i+1)$ is the vertical offset from glyph i to glyph $i+1$. Note if adjacent glyphs are copied over then $\delta_y(i \rightarrow i+1)$ will be drawn from a distribution with a standard deviation of zero – the glyphs will maintain their original relationship. However, flow adjustment (described below) is required. For this, a lower limit is enforced on the standard deviation, so there is always some flexibility in the vertical glyph placement. This allows distortion to be hidden throughout the text.

When ligatures are unavailable for either glyph, a regression forest, this time trained on the vertical offset, is used. It is setup identically to the random forest in Subsection 6.2, with the same features (Subsection 7.3) but trained to infer vertical offset rather than match cost. The regression forest outputs a standard deviation with its estimate, which is used directly for drawing $\delta_y(i \rightarrow i+1)$. This approach does result in some stylistic transfer between authors, from authors with neat handwriting to authors with messy handwriting, and vice versa.

Flow If the above vertical spacing was applied directly, the text would not stay on the ruled line. In practice, the author adjusts the flow of their text to stay on the rule, rather than drifting off due to the accumulation of small offsets (∼Brownian motion)[8]. We simulate this corrective process with Kalman smoothing [Kalman 1960], over the length of the horizontal line – see Figure 12. Specifically, each glyph's vertical position is the hidden state, which is linked to the adjacent glyph's vertical position by the expected vertical offset,

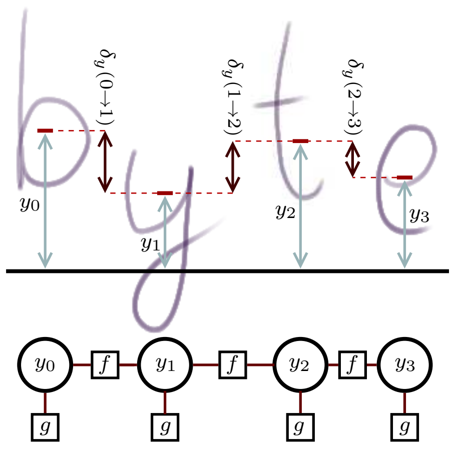

$\delta_y(0\rightarrow1)$
$\delta_y(1\rightarrow2)$
$\delta_y(2\rightarrow3)$
y_0
y_1
y_2
y_3
f
g

Fig. 12. Factor graph of Kalman smoothing, used to ensure the synthesized text flows. The y_i random variables (circles) indicate how far off the baseline to position each glyph. Factors (squares) indicate probabilities over the random variables to which they are connected. Kalman smoothing finds the most probable assignment of y_i values. There are two types of factor. g factors indicate that the glyphs are probably on the baseline, as a Gaussian distribution where the mean is the displacement of the glyph in the training data, h_i. This factor is $y_i \sim \mathcal{N}(h_i, 1)$. f factors indicate the displacement between adjacent glyphs, δ_y, as given in (7). When ligatures exist they provide this term, but when omitted (e.g. print handwriting) it is provided by a regression forest.

$\delta_y(i \rightarrow i+1)$. The unary term indicates that glyphs are probably on the ruled line. It takes the form $\mathcal{N}(h_i, 1)$, where h_i is the height from the rule in the author's sample and the standard deviation is a parameter we tuned. Note that this is in line space, so one standard deviation is the entire height of the rule; this is a very weak requirement to return, but it proves sufficient for a realistic look. See Appendix H for a large block of synthesized handwriting, in which this is apparent. This approach is mathematically equivalent to dynamic programming with Gaussian distributions.

6.4 Ligatures

Ligatures, $L_A(g, x, l)$, are added whenever adjacent source glyphs have ligatures in the author's sample. This is accurate for fully joined up and print handwriting, but for the relatively rare partially joined up handwriting it may put ligatures in the wrong places or omit them when it shouldn't. However, when given a limited amount of data, there is no way to resolve this, and it proves a rea-

[8]You may demonstrate this to yourself by attempting to write on a line with your eyes closed.

sonable approximation. Identically to glyph layout, the texture term is irrelevant for this step, and no proxy required.

Both original ligatures are generated – the texturing stage is then responsible for resolving the overlap. They are distorted to precisely overlap, including adjusting them to have identical radius at overlapped vertices. The path of the ligature is defined as $f(t)$, where $0 \leq t \leq 1$. Start points for the ligatures are fixed, such that $f(0) = s_i$ and $f(1) = s_{i+1}$, using the notation of Figure 11 C). Given that $p_i(l)$ is the position and radius of ligature i at location l, *e.g.* $p_i(l) = [x, y, r]^T$, then $f(t)$ can be defined as

$$f(t) = (1 - w(t))p_i(t) + w(t)p_{i+1}(1 - t), \quad (8)$$

where $w(t)$ is an interpolating function, *e.g.* linear interpolation would be $w(t) = t$. We found $w(t) = \sin(0.5\pi t)$ to be a good choice, as the sine curve preserves the perception of momentum at the transition from the glyph to the ligature.

For clarity, the system has been described as only considering the possibility of one ligature connecting adjacent glyphs. In practice it supports an arbitrary number of ligatures; ligatures are matched with the closest ligature vertically, conditioned on the match being reciprocal. We are only aware of one scenario in English where this is common, specifically a double “t” in a joined up style, where the cross is typically drawn as a single line across both “t”s. This scenario is supported by the system, and can be seen in Figure 16.

6.5 Texturing & Graph Cuts

With three of the four optimization objectives fixed, only $T_A(g, x, l, R)$ remains to be optimized. Input to this stage is a vectorized line; the output a rendered image. The author’s sample provides examples of the relevant texture. Texture coordinates are attached to the vectorized line, such that the source pixels can be queried and copied over. As the input contains multiple overlapping line segments, particularly in the case of ligatures, an optimization is required to generate the final texture.

A mapping from the output coordinate system to the source coordinate system is required for each line segment, as the line can be rotated and scaled when matching ligatures. This is achieved using UV coordinates attached to the line segments. For every pixel in the output, the texture is indexed and written out, forming multiple layers where ligatures overlap. Note that an extra margin of 8 pixels is added beyond the radius, to be sure of capturing all edge detail; transparency information is available (see Appendix C).

The mapping of the original texture to the output will generate multiple estimates for some pixels. Graph cut textures [Kwatra et al. 2003] is used to choose which pixel color to keep. It optimizes the pixel color selection such that the color difference between adjacent pixels is similar to the color difference in the source image. This is minimized by selecting pixels that were originally adjacent. Consequently, the seams between textures from different sources are hidden, by locating them where they are visually inconspicuous. The original color difference term of Kwatra et al. [2003] is extended to include an alpha channel, so transparency levels are preserved,

$$M(s, t, A, B) = ||A(s) - B(s)|| + ||A(t) - B(t)||. \quad (9)$$

M is the cost of transitioning from line segment A to line segment B between two adjacent pixels, s and t. In the original $A(x)$ and $B(x)$ return $\mathbb{R}^3$, representing the three color channels, red, green and blue. We extend it to $\mathbb{R}^4$ with the addition of alpha. The minimum cost cut is then found using the maximum flow algorithm [Ford and Fulkerson 1956; Boykov and Kolmogorov 2004]. This is repeated for all overlapping line segments to generate the final output image, R.

7. POST-PROCESSING

The system described can render realistic “handwriting” on screen. However, there is the further consideration of color matching the output printer, to fully sell the illusion. Swapping one pen type for another in the output phase is additionally explored.

7.1 Color Matching

The output will often need to be rendered to actual paper. Given the potential use cases, complex calibration methods are best avoided – the desire is to print on a typical printer with minimal effort. Our calibration consists of printing the color calibration target given in Appendix K, and then scanning it in using the same scanner as the actual text sample, closing the loop. This provides a mapping from the color sent to the printer to the color that is actually on paper, in the same color space as the collected author’s sample. Given this set of matches, $\{s_i, p_i\} \in M$, where s_i is a scanned color, and p_i is the color sent to the printer, we need a function that maps from s_i to p_i. As the set of samples is sparse, an interpolation method is required to generate a dense $3D$ look up table (LUT).

Thin plate splines [Bookstein 1989] (TPS) are used; there are many alternate splines that would be similarly effective, as the only requirement is to exactly map s_i to p_i at the locations where matches are available. TPS have the advantage of being parameterless however, which is ideal given that non-technical users would be expected to perform this step. One TPS is required to model each of the three color channels.

Printer ink is relatively limited in its dynamic range, so the values output by the TPS may be clamped, as the ink used in many pens exceeds the printer’s range. However, for ballpoint and pencil, it is a good match. Note that inkjet printers perform much better than laser printers for this purpose, as they have both a larger color space with a more accurate reproduction, plus the ink bleeds slightly, similarly to a pen.

7.2 Line Ink Replacement

An author will provide a sample of their handwriting using only one pen, but a user may want output with another. For a deceased author, samples might have been written with multiple pens, and attempting to combine them without line replacement would prove visually inconsistent. Our system allows complete replacement of the line, with a writing implement of the user’s choice. A sample of the target ink must be available. Line extraction must have occurred (Section 5) but glyph labeling is unnecessary. Real writing samples work best, as they include all the shapes and density variations found in handwriting.

Replacement interrupts the default renderer just before the texturing stage, so graph cut textures will generate a visually compelling result. First, both the text line and replacement ink line are divided into short overlapping segments, each 16 pixels long with 50% overlap. Each segment is assigned a *line* feature vector, specifically the position delta, radius and density in 4 histogram bins along the length of the linear segment. Segments from the text line are then replaced by segments from the ink line that are closest using the *line* feature (nearest neighbor), with the ink segments distorted to match the path of the text segments. To avoid visual repetition, the system avoids using the same ink segment near to where it was previously used. The *line* feature vector encourages matched segments to have comparable density, radius, and shape.

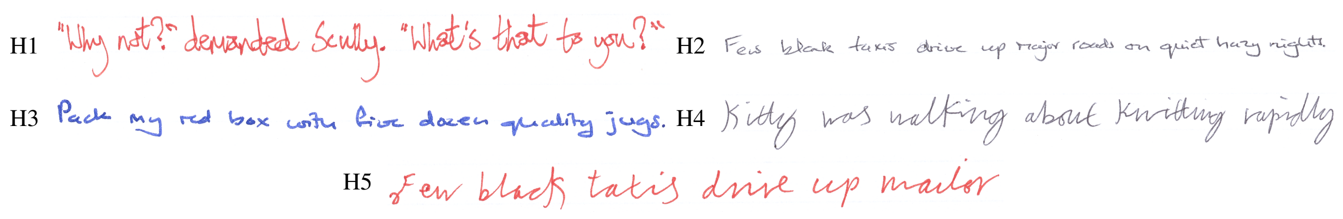

H1 "Why not?" demanded Scully. "What's that to you?"
H2 Few black taxis drive up major roads on quiet hazy nights.
H3 Pack my red box with five dozen quality jugs.
H4 Kitty was walking about knitting rapidly
H5 Few black taxis drive up major

Fig. 13. Example line of input to the system for the five styles used in the user studies.

The ink segments keep their texture information so when this alternate line information is passed to the graph cut textures phase, it is merged to generate a visually consistent output. While this approach obtains plausible results, it does not affect the output shape, so converting writing on a high friction surface to writing on a low friction surface, or vice versa, may not be entirely convincing. Density profiles of pens are matched (see Subsection 5.1.2), but not all pens have compatible density profiles, *e.g.* a fineliner has a very consistent line, with no density variation, so if converted to pencil there will be no density variations in the output, which is inconsistent with a real pencil. Figure 10 B) depicts representative results.

7.3 Glyph Feature

A feature vector is required for several parts of the algorithm, specifically 1) transfer learning the adjacency cost, 2) regressing on the relative position of glyphs, 3) letter recognition. In all cases this occurs after line extraction, including estimating the radius and density of the line, so this information can be utilized. The feature is defined for every pixel on the extracted line (which is one pixel wide after thinning).

Similarly to SIFT [Lowe 2004] and HOG [Dalal and Triggs 2005], we construct histograms over properties extracted from the region surrounding each pixel. Unlike these approaches which obtain their region from the pixel grid directly, we obtain a region over the extracted line of the text. This minimizes sensitivity to the pen type, particularly as we use the extracted features, which have been modified to have the same statistics regardless of writing implement (Subsection 5.1.1).

The feature summarizes the information using a histogram of relative occurrence for position and density[9]. Unlike SIFT/HOG, the feature vector is defined on the line of the text, in terms of random walks [Einstein 1956], so the implicit density estimate in the feature vector is conditioned on distance traveled. Without the random walk this would be shape contexts [Belongie and Malik 2000] augmented with extra features (extra histogram dimensions); the addition of a random walk encodes information about the path the pen traveled, which improves performance.

Specifically, for every point on the line, (equivalent to a pixel after thinning) a random walk over the graph of the text is considered. After traveling distance δ_t, the change in density δ_d is noted, along with the change in location, δ_x and δ_y, relative to the starting location. At junctions, the walk chooses an exit at random, discounting the entry point; it stops at line ends. The change in location is in the coordinate frame defined so that positive x is the direction of travel at the start of the walk (forward differencing, after traveling 12 pixels for robustness to noise/junctions). This is necessary for rotation invariance and to nullify the effect of the starting direction of the walk (particularly at junctions, as otherwise the feature would not vary smoothly).

Given a prior on travel distance, a density estimate over the end points of random walks is calculated, $P(\delta_t, \delta_d, \delta_x, \delta_y)$. This is represented with a histogram, which then constitutes the feature vector. Eight bins are used for travel distance, three bins for each of the rest, making a feature vector of length $216 = 8 \times 3 \times 3 \times 3$. Linear interpolation is used when a point falls between the centers of bins, *i.e.* it is a histogram with overlapping $4D$ triangular kernels. Maximum travel distance is set to 512 pixels[10]. The prior on travel distance is even odds of falling into each bin, where the bins are subject to a geometric relationship such that the next bin as you move away from the start is 1.25 the size of its predecessor. Diffusion is used to calculate the feature vectors exactly, as for an infinite number of random walks [Einstein 1956]. Note that the parameters given above are selected to maximize recognition performance.

8. RESULTS

The system encapsulating our algorithm was validated experimentally to measure the realism and style-similarity of its rendering, and to demonstrate the printer calibration stage. We used our own dataset for quantitative evaluation, to which we added samples from historical figures for qualitative results.

8.1 Handwriting Dataset

We collected a dataset containing handwriting samples from different subjects consisting of a variety of writing implements and handwriting styles. For each subject, the source text contained complete sentences to be written on the blank side of single-sided ruled paper. Subjects were able to see the ruled lines through the paper, without the lines appearing in the final scans. The sample length was typically four pages, obtained as in Section 4. The source text for each subject was obtained using the algorithm in Subsection 4.1. Note that each of our subjects received a different random source text, to avoid repetition in the later user studies.

When evaluating our system, we used a subset of our complete dataset, denoted as H1–H5, to demonstrate the range of supported styles. These samples were chosen as they offer variety in both handwriting style and pen type. H1 is a red fineliner, H2 and H3 are black ballpoint and blue fountain pen inputs from a second individual, and H4 and H5 are black ballpoint and red fineliner provided by a third individual. The samples each contain an average of 1387 glyphs. Examples of the input handwriting are shown in Figure 13.

[9]The inclusion of radius was experimented with, but found to confer no advantage.

[10]At 600dpi, a glyph is typically around 50 pixels wide, so a walk can travel around $5 - 10$ characters depending on ligatures, suggesting that classification utilizes entire words, a result consistent with the handwriting recognition literature [Plamondon and Srihari 2000].

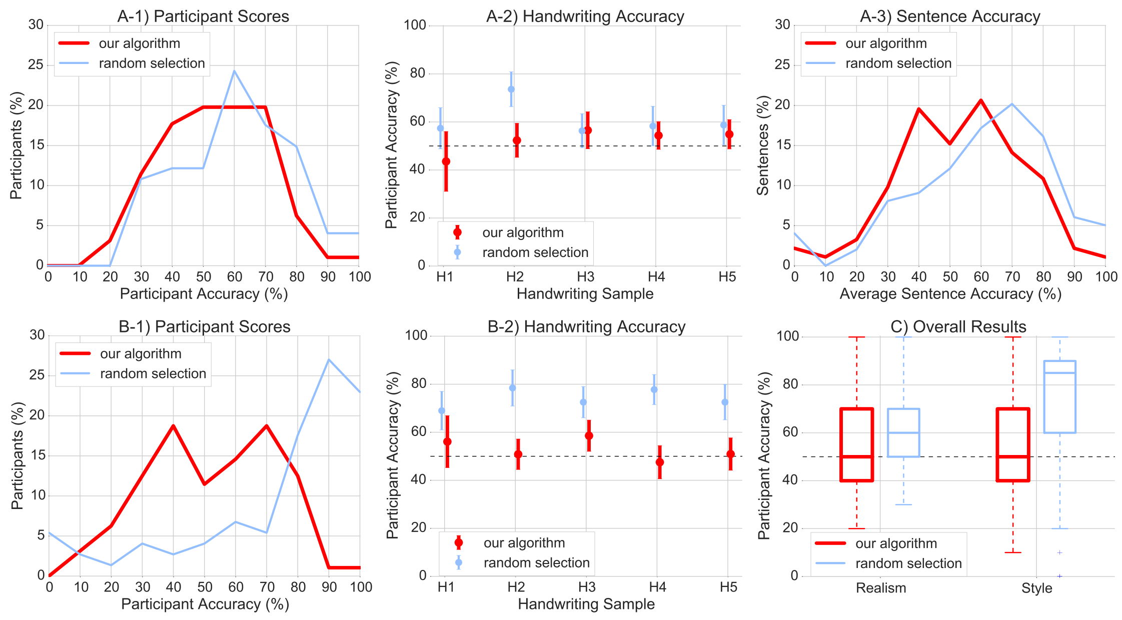


Fig. 14. Results from our 170-person study. A-1) Gives the percentage of users (y-axis) who got a given score (x-axis), for both synthetic algorithms. Participants who do a better job of identifying fakes appear further to the right, and those in the middle (50%) are effectively guessing. A-2) Gives the performance (y-axis) for each tested pen choice/writing style (x-axis), with 95% confidence intervals indicated. A-3) Gives the percentage of sentences (y-axis) in task A at each accuracy level (x-axis). B-1,2) Same as A-1,2) but for task B. Note how in B-1) most participants are correctly identifying "random selection" while for our algorithm, the graph remains in the center, due to the difficulty of identifying our fakes. C) gives the overall performance on both tasks for both algorithms, with the 95% confidence (bar), the median (line) and minimum/maximum values.

8.2 User Study

We conducted a perceptual study to determine how effective our system is at generating realistic handwriting. The evaluation is split into three study-questions: 1) Can we synthesize realistic handwriting? 2) Can we synthesize handwriting in the same style as a given author? and 3) How realistic do our synthesized results look compared with real pens when printed with a standard printer?

Questionnaire To evaluate study-questions 1) and 2), we gave a printed questionnaire containing both real and synthesized handwritten sentences to 170 different participants. We opted to use a printed questionnaire, as opposed to an on-screen one, to more naturally represent how handwriting is typically seen. Each questionnaire contained two tasks A) and B), respectively addressing study-questions 1) and 2), with 10 questions per task. Using the five handwriting styles dataset, we excluded 12 sentences per handwriting style from the training set to use for participant questionnaires. We also rendered our synthetic version of each sentence using two different algorithms – our algorithm (Section 6) and a variant we refer to as "random selection." In "random selection" most of our algorithm's components have been disabled, such that it is comparable to both Guyon [1996] and Chowdhury et al. [2009]. See Appendix F for a detailed breakdown of our algorithm's components. Each questionnaire contains a random sampling without replacement of these handwriting examples, ensuring that each sentence was seen by multiple participants. Participants were given questionnaires containing only results from our algorithm or only results from "random selection", in addition to real examples. They took approximately five minutes on average to complete the whole questionnaire. A sample questionnaire is in Appendix J.

Realism Evaluation In task A, each participant was shown 10 sentences. For each sentence they were asked to determine if it was real or "fake." We chose the term "fake", as opposed to alternatives such as "computer generated", as we did not want to bias participants into expecting particular types of artifacts. When choosing the 10 sentences for each questionnaire, we picked 50% real and 50% "fake" (synthesized). Figure 14 A-1) shows the distribution of the participants' scores for both algorithms. A perfect score of 100% indicates that the individual correctly marked all five real sentences as real and all five synthesized sentences as fake. The percentage of correct answers across all participants was 53.75% for our algorithm, and 60.68% for "random selection." This indicates that participants found it easier to identify writing generated by "random selection" as being synthesized. Both values are close enough to random chance (50%) to indicate that users are struggling to tell the difference between real and "fake." In Figure 14 A-2) we can see the average percentage of sentences in each of the five different styles that were labeled correctly. Here, a score of 100% indicates that all the sentences from that style were correctly labeled. We also display the 95% confidence interval in red. Given these confidence intervals, we conclude that the differences between the two synthetic approaches, when viewed in isolation, are not obvious to casual observers. Both algorithms have succeeded at task A, with neither better than the other.

In Figure 14 A-3) we can see a difficulty histogram for the sentences. The x axis identifies the difficulty of a sentence (percentage of test subjects who got it right), while the y axis identifies the percentage of test sentences that fell into each difficulty level. Participants tended to be better at identifying "random selection" generated sentences as being fake compared to our algorithm. Fig-

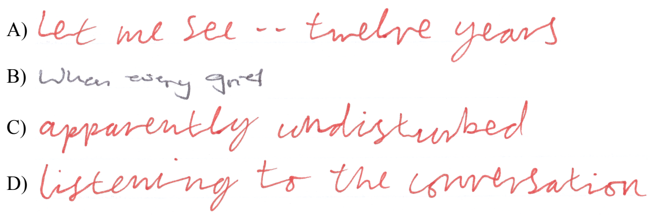

A) Let me see -- twelve years
B) When every grief
C) apparently undisturbed
D) listening to the conversation

Fig. 15. The four handwriting examples above were all generated by our system. Fewer than 20% of participants detected that A) and B) are fakes, while 80% correctly stated that C) and D) are fakes. Note that C) has a bad ligature between "l" and "y" in the first word while D) also has a long gap after the second "i" in the first word – we expect these mistakes gave them away.

ure 15 contains the example sentences from our algorithm that were both the easiest and hardest for our participants to detect as fakes.

Style Synthesis Evaluation Next, in task B, we evaluated if our system is capable of generating sentences in the same style as a specific author. Participants were given a real sentence from one of H1–H5 for reference, and below this were given two other samples, in the same handwriting style and pen type. Of the two sentences below, one was real and the other was generated by one of the algorithms. Each participant was shown 10 different sentence triplets and, for each, asked to determine which of the two bottom sentences was fake. We ensured that none of the sentences from task 1 appeared in task 2. The overall accuracy across all participants was 52.4% for our algorithm and 74.1% for "random selection." In contrast to the realism evaluation in task 1, participants performed much worse for the competing method when compared to ours. While "random selection" can generate plausible sentences in isolation (task 1), when viewed beside real handwriting from the same individual, it is not convincing. It is worth noting that participants were only slightly better than guessing at random (50%) for our algorithm. Figure 14 B-1) displays the participant accuracy, while B-2) gives the accuracy breakdown per style. We see that most participants found it difficult to determine if a given sentences' style was authentic or synthetic.

Calibration Evaluation In our final perceptual test, we wished to determine how good participants were at detecting the synthesized handwriting from our algorithm when printed on paper, compared with real pens. This is a challenging test, as real ink often exceeds the capabilities of a printer (Subsection 7.1). We created a set of 15 envelopes with addresses displayed on the front; see supplementary video for examples. Eight of the addresses were generated by our system and then printed, and the remaining seven were provided by real subjects with different handwriting styles and writing implements. Both the real and synthetic envelopes featured red fineliner, black ballpoint, and blue fountain pen. When printing our synthesized results, we first color calibrated the output using the method outlined in Section 7.1, and then printed them using a typical color inkjet printer (Epson WP-4535). Examples of our synthesized addresses are shown in Figure 16.

Participants were presented with each envelope in turn and asked to inspect each one to determine if it was written by a human using a pen, or if it was printed by a computer. They had five seconds to inspect each envelope. We conducted our test in an office environment with standard overhead lighting and natural lighting coming from large outdoor facing windows. 18 participants took part and overall only 60.73% of the 15 envelopes were correctly identified.

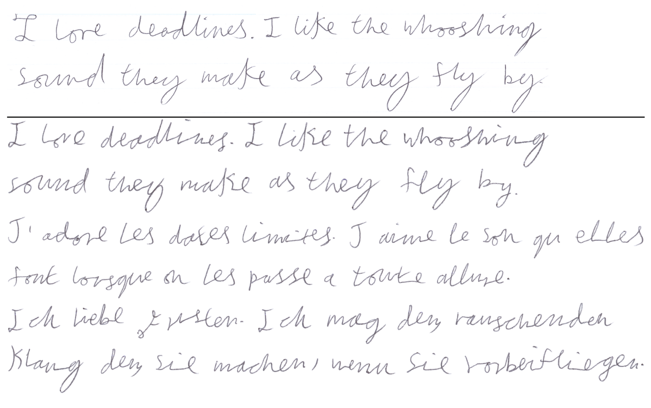

I love deadlines. I like the whooshing sound they make as they fly by.
I love deadlines. I like the whooshing sound they make as they fly by.
J'adore les dates limites. J aime le son qu elles font lorsque on les passe a toute allure.
Ich liebe Fristen. Ich mag den rauschenden Klang den sie machen, wenn sie vorbeifliegen.

Fig. 18. Example of translation of a Douglas Adams quote into French and German. The first version is real writing, captured many months after the sample used to generate the three below it.

8.3 Qualitative Results

Here and in the supplementary video, we explore a range of applications that our system enables. As described in Section 7.2, we can transfer the ink style from one pen to another. The input pen in Figure 10 B) is a black ballpoint. We can render many different pen types, from pencil to marker, while keeping the same writing style. Figure 17 shows new sentences synthesized in the style of famous historical figures. Both of these examples took a novice user under 45 minutes to tag. Figure 18 shows that it is possible to synthesize in different languages, provided they use the same characters. We have only explored the English alphabet so far. Appendix H shows a rendering of multiple paragraphs of text, for medium-sized handwriting tasks. Appendix E shows a qualitative comparison of our output and the "random selection" algorithm.

9. DISCUSSION

We have presented a texture synthesis system that can convincingly replicate the handwriting of a specific author. Handwriting synthesis is semi-supervised, as a user selects the target text, which is the interaction required in our use cases. The precise texturing required to create convincing handwriting is then generated to match the target text without further user input (unsupervised). Rendering a sentence typically takes around 8 seconds on a 3.8Ghz i7. User studies indicate that our approach generates output that looks real to a casual observer. We are the first to replicate the writing of a historic figure.

Limitations Joined up handwriting proves the simplest to generate, as the pen leaves the page relatively rarely. When the pen does leave the page, effort has to be made to infer its path, to generate realistic output. This makes print and partly joined-up writing more challenging. Decorative styles can fail however, particularly ones with long range interactions such as when the cross on a "t" is drawn across the entire word, with gaps for the ascenders of other letters. No effort was made to consider writing in a limited space, where humans gradually start to bunch up their glyphs. Our model of the writing implement does not include orientation; consequently calligraphy does not work.

Good quality data is required: Figure 19 shows a failure case where there is insufficient resolution (glyphs are 30 pixels high) and data (only 52 glyphs in the source). The semi-automated handwriting analysis is the greatest time-burden for a user. Our UI makes this task possible, but future improvements could make the one-

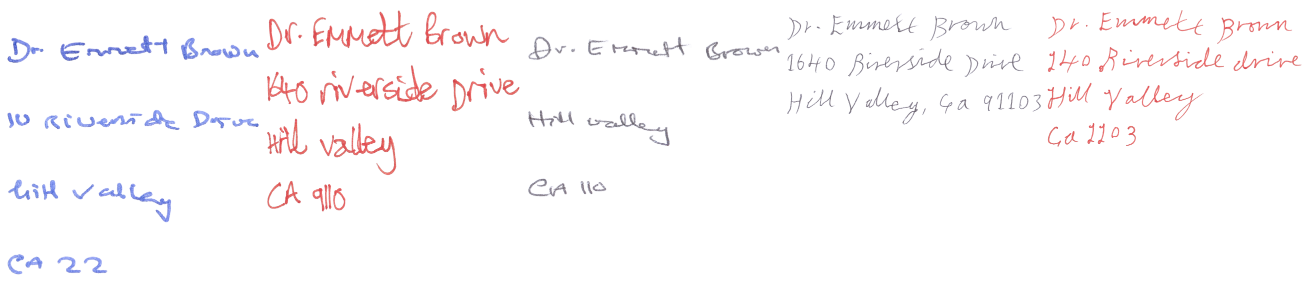


Fig. 16. Addresses synthesized by our algorithm.

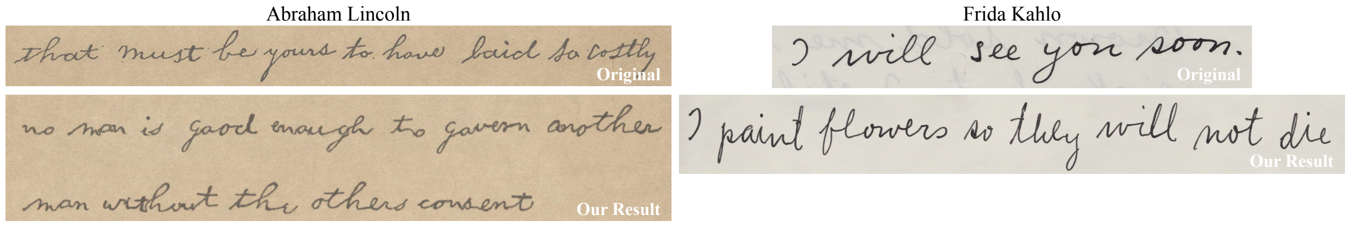

Abraham Lincoln
that must be yours to have laid so costly
Original
no man is good enough to govern another
man without the others consent
Our Result
Frida Kahlo
I will see you soon.
Original
I paint flowers so they will not die
Our Result

Fig. 17. Rendered quotes, in the handwriting style of famous individuals, though Abraham Lincoln's sample is from the contentious Bixby letter. Examples of the authors actual handwriting are on the first row; synthesized results are on the second.

time-effort both simpler and faster. There is also an issue of coverage, as the system does not synthesize entirely new glyphs. If no sample of a letter written by an author is available then any text containing it cannot be successfully synthesized. This unfortunately limits use cases built around translation to a foreign language. Languages with high character counts, *e.g.* Chinese ($>$ 3000), would be prohibitively difficult to synthesize, as it is unreasonable to capture sufficient data.

Future Work Given a large enough database of samples, the coverage issue could be resolved by finding authors with similar handwriting, and sharing glyphs between them. Currently, cooperative authors all provide around four pages of text. For authors with simplistic styles this proves excessive, for authors with complex styles it proves insufficient. An active learning approach, *e.g.* [Cooper et al. 2007] or [Haines and Xiang 2014], could adapt to the author and request the right quantity of handwriting samples to build a successful model.

Our optimization strategy assumes the output is going to a blank canvas. Instead, document-repair may be possible by adding another term to the cost function $C(\cdot)$, such that the rendered output must agree with existing ink. This would inpaint the damaged areas of a document, and provide a probability distribution over words that are no longer readable. Further, layout-style constraints like those in [Jacobs et al. 2004] could be incorporated to synthesize mixed-content documents with handwriting that wraps around images and other visual elements. Finally, though meant for aesthetic purposes, this system has potential forensic applications, *e.g.* identifying the author of a handwritten note.

To encourage future work we have releasing both the code and our complete dataset, which is available from the project website, `http://visual.cs.ucl.ac.uk/pubs/handwriting`.

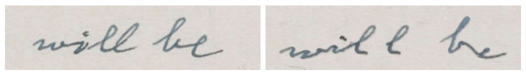


Fig. 19. A failure case with Albert Einstein's handwriting. On the left his real writing, on the right output generated with insufficient samples and resolution.

Acknowledgements

Project supported by EPSRC grants EP/J021458/1, EP/K023578/1 and CR-PLAY, EU project #611089.
We would like to thank Aindri Chakraborty, Bryan O'Regan, Melissa Terras and the reviewers for their kind assistance.

REFERENCES

ALEXANDER, R. M. 1984. The gaits of bipedal and quadrupedal animals. *Robotics Research*, 49–59.

ANDRE, J. AND BORGHI, B. 1990. Dynamic fonts. *Raster imaging and digital typography*, 198–204.

ANTONACOPOULOS, A., GATOS, B., AND BRIDSON, D. 2007. Page segmentation competition. *Document Analysis and Recognition*.

BELLMAN, R. 1952. On the theory of dynamic programming. *PNAS 38(8)*, 716–719.

BELONGIE, S. AND MALIK, J. 2000. Matching with shape contexts. *Content-based Access of Image and Video Libraries*, 20–26.

BLUM, H. 1967. A transformation for extracting new descriptors of shape. *Models for the Perception of Speech and Visual Form*.

BONNEEL, N., VAN DE PANNE, M., LEFEBVRE, S., AND DRETTAKIS, G. 2010. Proxy-guided texture synthesis for rendering natural scenes. *Proc. Vision Modeling and Visualization*.

BOOKSTEIN, F. L. 1989. Principal warps: Thin-plate splines and the decomposition of deformations. *PAMI 11(6)*, 567–585.

BOYKOV, Y. AND KOLMOGOROV, V. 2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI 26(9)*, 1124–1137.

BOYKOV, Y. Y. AND JOLLY, M.-P. 2001. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. *ICCV*, 105–112.

BRAND, M. AND HERTZMANN, A. 2000. Style machines. *SIGGRAPH*.

BREIMAN, L. 2001. Random forests. *Machine Learning*.

CAMPBELL, N. D. F. AND KAUTZ, J. 2014. Learning a manifold of fonts. *SIGGRAPH*.

CHANG, W.-D. AND SHIN, J. 2012. A statistical handwriting model for style-preserving and variable character synthesis. *Document Analysis and Recognition 15(1)*, 1–19.

CHOI, H., CHO, S. J., AND KIM, J. H. 2004. Writer dependent online handwriting generation with bayesian networks. *Frontiers in Handwriting Recognition*, 130–135.

CHOWDHURY, S., DAS, S., ROY, D., SARKAR, U., AND CHAUDHURI, B. B. 2009. A complete method of personal handwriting synthesis. *Advances in Graphonomics*, 250–253.

COMANICIU, D. AND MEER, P. 2002. Mean shift: A robust approach toward feature space analysis. *PAMI 24(5)*, 603–619.

CONNELL, S. D. AND JAIN, A. K. 2002. Writer adaptation for online handwriting recognition. *PAMI 24(3)*, 329–346.

COOPER, S., HERTZMANN, A., AND POPOVIC, Z. 2007. Active learning for real-time motion controllers. *SIGGRAPH*.

CORRELL, S. 2000. Graphite: An extensible rendering engine for complex writing systems. *Unicode Conference 17*.

CRIMINISI, A. AND SHOTTON, J. 2013. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer.

DALAL, N. AND TRIGGS, B. 2005. Histograms of orientated gradients for human detection. *CVPR*, 886–893.

DEVROYE, L. AND MCDOUGALL, M. 1995. Random fonts for the simulation of handwriting. *Electronic Publishing*.

DINGES, L., AL-HAMADI, A., AND ELZOBI, M. 2013. An approach for arabic handwriting synthesis based on active shape models. *Document Analysis and Recognition*, 1260–1264.

DOYLE, A. C. 1901. *Letter from Arthur Conan Doyle to Herbert Greenhough Smith.* `https://commons.wikimedia.org/wiki/File:Letter_from_Arthur_Conan_Doyle_to_Herbert_Greenhough_Smith.jpg`, Creative Commons Attribution-Share Alike 2.0, Toronto Public Library.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann.

EFROS, A. A. AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. *SIGGRAPH 28*, 341–346.

EFROS, A. A. AND LEUNG, T. K. 1999. Texture synthesis by non-parametric sampling. *ICCV 2*, 1033–1038.

EINSTEIN, A. 1956. Investigations on the theory of the brownian movement. *Courier Dover Publications*.

ELARIAN, Y., ABDEL-AAL, R., AHMAD, I., PARVEZ, M. T., AND ZIDOURI, A. 2014. Handwriting synthesis: classifications and techniques. *IJDAR 17*, 455–469.

ELARIAN, Y., AHMAD, I., AWAIDA, S., AL-KHATIB, W. G., AND ZIDOURI, A. 2015. An arabic handwriting synthesis system. *Pattern Recognition 48,* 3, 849–861.

ELARIAN, Y. S., AL-MUHTASEB, H. A., AND GHOUTI, L. M. 2011. Arabic handwriting synthesis.

FORD, L. R. AND FULKERSON, D. R. 1956. Maximal flow through a network. *Canadian Journal of Mathematics 8*, 399–404.

FUKUNAGA, K. AND HOSTETLER, L. D. 1975. The estimation of the gradient of a density function,with applications in pattern recognition. *Trans. Information Theory 21*, 32–40.

GANGADHAR, G., JOSEPH, D., AND CHAKRAVARTHY, V. S. 2007. An oscillatory neuromotor model of handwriting generation. *Document Analysis and Recognition 10(2)*, 69–84.

GARNER, R. 2005. Post-it note persuasion: A sticky influence. *J. Consumer Psycology 15*, 230–237.

GRAVES, A. 2013. Generating sequences with recurrent neural networks. *arXiv Preprint*.

GUERFALI, W. AND PLAMONDON, R. 1995. The delta lognormal theory for the generation and modeling of cursive characters. *Document Analysis and Recognition 3(1)*.

GUPTA, G. AND MCCABE, A. 1997. A review of dynamic handwritten signature verification. Tech. rep., James Cook University.

GUYON, I. 1996. Handwriting synthesis from handwritten glyphs. *Frontiers of Handwriting Recognition*, 309–312.

HAINES, T. S. F. AND XIANG, T. 2014. Active rare class discovery and classification using dirichlet processes. *IJCV 106(3)*, 315–331.

HINTON, G. AND NAIR, V. 2005. Inferring motor programs from images of handwritten digits. *NIPS*, 515–522.

HO, T. K. 1995. Random decision forests. *Proc. Document Analysis and Recognition 1*, 278–282.

HUANG, W., LIN, Z., YANG, J., AND WANG, J. 2013. Text localization in natural images using stroke feature transform and text covariance descriptors. *ICCV*.

JACOBS, C., LI, W., SCHRIER, E., BARGERON, D., AND SALESIN, D. 2004. Adaptive document layout. *Commun. ACM 47,* 8 (Aug.), 60–66.

KALMAN, R. E. 1960. A new approach to linear filtering and prediction problems. *ASME–Basic Engineering 82*, 35–45.

KNUTH, D. E. 1979. Metafont: a system for alphabet design. Tech. rep., Stanford University.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. *TOG*.

LEE, H., KWON, S., AND LEE, S. 2006. Real-time pencil rendering. In *NPAR*. 37–45.

LEWIS, J. P. 1984. Texture synthesis for digital painting. *Computer Graphics 18(3)*.

LIN, Z. AND WAN, L. 2007. Style-preserving english handwriting synthesis. *Pattern Recognition 40*, 2097–2109.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *IJCV*.

LU, J., YU, F., FINKELSTEIN, A., AND DIVERDI, S. 2012. Helpinghand: Example-based stroke stylization. *SIGGRAPH 31(4)*.

MAO, J. AND MOHIUDDIN, K. M. 1997. Improving ocr performance using character degradation models and boosting algorithm. *Pattern Recognition Letters 18*, 1415–1419.

MARCELO, B., SAPIRO, G., CASELLES, V., AND BALLESTER, C. 2000. Image inpainting. *SIGGRAPH*, 417–424.

MARTI, U.-V. AND BUNKE, H. 2002. Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition systems. *Hidden Markov models*.

MOHAMMED, U., PRINCE, S. J. D., AND KAUTZ, J. 2009. Visio-lization: Generating novel facial images. *TOG 28(3)*.

MORI, M., SUZUKI, A., SHIO, A., AND OHTSUKA, S. 2000. Generating new samples from handwritten numerals based on point correspondence. *IAPR-IWFHR*.

MUELLER, S., HUEBEL, N., WAIBEL, M., AND ANDREA, R. D. 2013. Robotic calligraphy - learning how to write single strokes of chinese and japanese characters. *IROS*.

NORVIG, P. 2014. Letter frequencies for scrabble. `http://norvig.com/scrabble-letter-scores.html`.

PAN, S. J. AND YANG, Q. 2010. A survey on transfer learning. *Knowledge and Data Engineering 22(10)*, 1345–1359.

PAPANDREOU, G. AND YUILLE, A. L. 2011. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. *ICCV*, 193–200.

PARIZI, S. N., VEDALDI, A., ZISSERMAN, A., AND FELZENSZWALB, P. 2014. Automatic discovery and optimization of parts for image classification. *arXiv preprint, arXiv:1412.6598*.

PERVOUCHINE, V. AND LEEDHAM, G. 2007. Extraction and analysis of forensic document examiner features used for writer identification. *Pattern Recognition 40*, 1004–1013.

PLAMONDON, R. AND SRIHARI, S. N. 2000. On-line and off-line handwriting recognition: A comprehensive survey. *PAMI*.

PORTER, T. AND DUFF, T. 1984. Compositing digital images. *Computer Graphics 18*, 253–259.

PORTILLA, J. AND SIMONCELLI, E. P. 2000. A parametric texture model based on joint statistics of complex wavelet coefficients. *IJCV 40*, 49–70.

HISTORY OF SCRABBLE. 2014. `http://www.scrabble-assoc.com/info/history.html`.

PROJECT GUTENBERG. 2014. `http://www.gutenberg.org/`.

REINHARDT, T. 2014. My script font. http://myscriptfont.com.

ROTHER, C., KOLMOGOROV, V., AND BLAKE, A. 2004. "grabcut" interactive foreground extraction using iterated graph cuts. *SIGGRAPH*.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video textures. *SIGGRAPH*, 489–498.

SCHRODER, M. 2001. Emotional speech synthesis: A review. *Interspeech*.

SHILMAN, M., TAN, D. S., AND SIMARD, P. 2006. Cuetip: A mixed-initiative interface for correcting handwriting errors. *User interface software and technology*, 323–332.

SRIHARI, S. N., CHA, S.-H., ARORA, H., AND LEE, S. 2002. Individuality of handwriting. *J. Forensic Science 47(4)*.

SUN, J., YUAN, L., JIA, J., AND SHUM, H.-Y. 2005. Image completion with structure propagation. *SIGGRAPH*.

THOMAS, A. O., RUSU, A., AND GOVINDARAJU, V. 2009. Synthetic handwritten captchas. *Pattern Recognition*.

VAN GALEN, G. P. 1991. Handwriting: Issues for a psychomotor theory. *Human Movement Science 10*, 165–191.

VARGA, T. AND BUNKE, H. 2003. Generation of synthetic training data for an hmm-based handwriting recognition system. *Document Analysis and Recognition*, 618–622.

VARGA, T. AND BUNKE, H. 2004. Comparing natural and synthetic training data for off-line cursive handwriting recognition. *Frontiers in Handwriting Recognition 9*, 221–225.

WAN, L. AND LIN, Z. 2009. Signature sample synthesis. *Encyclopedia of Biometrics*, 1205–1210.

WANG, J., WU, C., XU, Y.-Q., AND SHUM, H.-Y. 2004. Combining shape and physical models for online cursive handwriting synthesis. *Document Analysis and Recognition*.

WANG, J., WU, C., XU, Y.-Q., SHUM, H.-Y., AND JI, L. 2002. Learning-based cursive handwriting synthesis. *Frontiers in Handwriting Recognition 8*, 157–162.

WANG, Y., WANG, H., PAN, C., AND FANG, L. 2008. Style preserving chinese character synthesis based on hierarchical representation of character. *Acoustics, Speech and Signal Processing*.

WARNOCK, J., GESCHKE, C., BROTZ, D., TAFT, E., AND PAXTON, B. 1984. PostScript. `https://www.adobe.com/products/postscript/pdfs/postscript_is_20.pdf`.

WEI, L.-Y., LEFEBVRE, S., KWATRA, V., AND TURK, G. 2009. State of the art in example-based texture synthesis. *Eurographics*.

WERNER, L. 1999. Getting java ready for the world: A brief history of IBM and Sun's internationalization efforts. `http://www.icu-project.org/docs/papers/history_of_java_internationalization.html`.

WILLIAMS, B. H., TOUSSAINT, M., AND STORKEY, A. J. 2007. Modelling motion primitive and their timing in biologically executed movements. *NIPS*.

WILLIAMS, G. 2014. Font forge. `http://fontforge.org/`.

XIONG, C., JOHNSON, D., XU, R., AND CORSO, J. J. 2012. Random forests for metric learning with implicit pairwise position dependence. *Knowledge Discovery and Data Mining*.

ZHANG, T. Y. AND SUEN, C. Y. 1984. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*.

ZHENG, Y. AND DOERMANN, D. 2005. Handwriting matching and its application to handwriting synthesis. *Document Analysis and Recognition*.

ZITNICK, C. L. 2013. Handwriting beautification using token means. *SIGGRAPH*.

APPENDIX

A. INKED PIXEL EXTRACTION

The task is to classify pixels as belonging to the background or foreground, using the the estimated ink density, d (Subsection 5.1.1). Our approach is related to [Rother et al. 2004]. Each pixel in the image is modeled as a foreground/background binary random variable, connected in a grid to form a conditional random field (CRF). The goal is to minimize

$$\sum_{p \in V} U(p) + \sum_{(p,q) \in Ed} P(p,q), \tag{10}$$

where V is the set of random variables (pixels) and Ed is the set of edges connecting pixels in a 4-way neighborhood. The unary term, $U(p)$, is derived from the density map, d,

$$U(p) = m_p \begin{cases} -8 & d_p = 0 \\ 12 \min(d_p, 1) & d_p > 0, \end{cases} \tag{11}$$

where m is the mask being optimized: 1 for foreground, 0 for background. It encourages high density pixels to belong to the foreground, while pixels with no density are strongly encouraged to belong to the background, which is important to separate lines that are spatially near each other. The pairwise term, $P(p,q)$, encourages pixels with similar colors to share the same label,

$$P(p,q) = \begin{cases} 0 & m_p = m_q \\ \frac{\gamma\lambda}{\lambda + \sqrt{(\bar{c}_p - \bar{c}_q)^T(\bar{c}_p - \bar{c}_q)}} & m_p \neq m_q, \end{cases} \tag{12}$$

where $\bar{c}$ is the color of a pixel ($\{c_r, c_g, c_b\} \in [0,1]$), and λ sets the color difference at which the cost of assigning different labels to adjacent pixels has dropped to half its maximum. γ is the maximum value, which is set to 64; λ is set to 0.5. This CRF is solved with graph cuts [Boykov and Kolmogorov 2004].

B. LINE-AWARE SMOOTHING

The initial segmentation, from Appendix A, is converted into a signed distance function, where each pixel contains the distance to the nearest ink edge. The distance is negative if it is inside the ink, positive otherwise. Our aim is to smooth this field. For pixel (i) and two of its direct neighbors (j, k), the vector $\hat{n}$ toward the nearest line can be computed as

$$\hat{n} \propto (s_j - s_i)\hat{n}_{ij} + (s_k - s_i)\hat{n}_{ik}, \tag{13}$$

where s is the signed distance field and $\hat{n}_{ij}$ is the unit vector going from pixel i to pixel j. This estimate assumes that the pixels have the same nearest line, which can be approximated as being straight over short distances. For a pixel's 8-way neighborhood, there are 8 such estimates (using adjacent neighbors only). They are all calculated, and the most consistent used to smooth the field at the current pixel. Smoothing consists of estimating the current pixel's signed distance value using each of its 8 neighbors

$$s_i' = s_j + \hat{n} \cdot \bar{n}_{ij}, \tag{14}$$

where $\bar{n}_{ij}$ is the offset from pixel i to pixel j (unnormalized $\hat{n}_{ij}$). The pixel's signed distance is updated to a robust average of the estimates, specifically the median. The most consistent estimate is defined as the line direction that results in the least variability in the estimates. Iteratively updating the signed distance function with this robust estimate smooths the edges of lines without creating holes in lines that are only a few pixels wide. Finally, the field is converted back to a mask by checking the sign of each pixel.

C. MATTING

The segmentation of the line is dilated to make sure all parts of the line are excluded, then the ink area is inpainted [Marcelo et al. 2000] to estimate the background on which the author wrote. At each pixel we have the *matting equation* [Porter and Duff 1984]

$$\bar{c} = \alpha\bar{i} + (1-\alpha)\bar{p}, \tag{15}$$

where $\bar{c}$ is the known image color, $\bar{p}$ the background that has been estimated with inpainting, $\bar{i}$ the unknown ink color[11], and α the unknown alpha value. This is a linear equation that is satisfied by many values of α and $\bar{i}$. All values are constrained to lie in $[0,1]$, and we select the lowest α value within this range. This is equivalent to setting at least one value of $\bar{i}$ to be 0 or 1, depending on if the background is brighter or darker than the ink. Given that (15) rearranges as

$$\alpha = \frac{c_j - p_j}{i_j - p_j}, \tag{16}$$

where $j \in \{r, g, b\}$ we can solve for all 6 combinations of color (r, g and b) and limiting ink values ($i_j = 0$ and $i_j = 1$). This equation, calculated for each combination, indicates that the real α must be in $[\alpha, 1]$, otherwise the constraint is not satisfied. Therefore, the maximum of the six α values will be the lowest α that does not violate the $[0,1]$ constraint for $\bar{i}$, so alpha is set to

$$\alpha = \max_{j \in \{r,g,b\}} \left[\max\left(\frac{c_j - p_j}{0 - p_j}, \frac{c_j - p_j}{1 - p_j} \right) \right], \tag{17}$$

and ink color to

$$\bar{i} = \frac{\bar{c} - (1-\alpha)\bar{p}}{\alpha}, \tag{18}$$

noting that care has to be taken to handle division by zero. This is justified under the assumption that the colors of the ink and paper are linearly independent. If they are not (*e.g.* black ink on white paper), it is equivalent to assuming the ink is the strongest color it can be. For black ink, this is correct, while for gray ink on white paper, its transparency will be over-estimated, *i.e.* it will be modeled as transparent black ink. At least when placed on the original background, this approach guarantees the ink will appear identical to the original.

D. TAGGING INTERFACE

For correcting tags the interface offers the possible interactions of:

—Drawing a line across the pen path to *split* the path.

—Drawing a line from one part of the pen path to another to merge them. If there are *splits* dividing them, they are removed, otherwise a *link* is created.

—Drawing a line *across* a link to delete it.

—Typing while the cursor is over a segmented glyph to *label* it. Underscores are used to indicate the start and end of a word: Before the character to indicate the start of a word, after the character to indicate the end of a word. Single letter words are tagged as "_a_."

[11] Assuming the ink is the same color as the previously extracted mode does not work, as ink color can vary considerably.

Dataset	Line Extraction	Tagging	Algorithms	Total
Neat Fineliner, Manual	12:13	3:02	N/A	16:15
Neat Fineliner	0:17	2:24	0:12	2:54
Messy Fineliner	0:28	2:20	0:12	3:00
Messy Pencil	3:06	2:21	0:10	5:37

Fig. 20. Timing for several data sets, given as hours:minutes. *Line extraction* is the time spent fixing segmentation errors for the line; *Tagging* is the time spent fixing errors with character tagging, including separation. Algorithms is the time spent running the automatic algorithms, which does not involve the user. Times are extrapolated from tagging a subset of each data set. For fairness, the character recognition model did not include any training exemplars from these authors. In actual usage, you retrain the model regularly, such that recognition gets better as you train more examples from a given author.

The above works with either a mouse or tablet. Panning and zooming are supported using the tablet pen buttons and swipes, or the right mouse button and wheel. Please refer to the supplementary video for a demonstration of the full tagging process.

D.1 Timing

The process of preparing a handwriting sample for synthesis requires a substantial investment of time by the user, as even if the automatic process performed perfectly, its output still needs to be checked for errors. Figure 20 contains the times for tagging several typical files. The biggest factor is the writing implement. A fineliner gives a consistent clean line that is almost always extracted without error – the time given is for looking over the line and checking it is correct. At the other end of the spectrum is pencil, where the line is inconsistent, can contain gaps and can become very faint when the pencil needs sharpening. This requires a lot of work to correct. The tagging time without retraining remains extremely consistent, at around 2 hours and 20 minutes, as our recognition model lacks the quantities of training data required for it to generalize well to previously unseen authors. However, in practice we find that the per file tagging time is reduced due to retraining, typically to half the starting time (*e.g.* 563 seconds to 256 seconds per line, for one complete sequence) as the model learns to recognize the specific author, and hence it remains a valuable feature. As a point of comparison, an estimate of how long manual tagging would take is included for neat fineliner. It is extrapolated from a single line as the tagger got hand cramps, despite using a tablet. "Manual" refers to avoiding all automation – line extraction by painting over the text in an image editor to create a binary mask, followed by using the tagger without any automation. In conclusion, while the automated system is not perfectly automatic it is at least three times faster than manual, and can be as much as eight times faster. Convincing handwriting synthesis would be impossibly difficult without it.