Handwriting Beautification Using Token Means

C. Lawrence Zitnick*
Microsoft Research

Figure 1: *(left) Several examples of handwritten text and shapes before and after beautification. The beautified results are found by shifting the strokes closer to means of token clusters. Notice how the beautified results are more consistent and easier to read, yet they still have the variation and style of the original writings. (right) Example tokens generated from writing "SIGGRAPH."*

Abstract

In this paper, we propose a general purpose approach to handwriting beautification using online input from a stylus. Given a sample of writings, drawings, or sketches from the same user, our method improves a user's strokes in real-time as they are drawn. Our approach relies on one main insight. The appearance of the average of multiple instances of the same written word or shape is better than most of the individual instances. We utilize this observation using a two-stage approach. First, we propose an efficient real-time method for finding matching sets of stroke samples called tokens in a potentially large database of writings from a user. Second, we refine the user's most recently written strokes by averaging them with the matching tokens. Our approach works without handwriting recognition, and does not require a database of predefined letters, words, or shapes. Our results show improved results for a wide range of writing styles and drawings.

CR Categories: I.3.8 [Computing Methodologies]: Computer Graphics—Applications;

Keywords: sketch, handwriting, beautification, vector graphics

Links: DL PDF

*e-mail:larryz@microsoft.com

ACM Reference Format
Zitnick, C. 2013. Handwriting Beautification Using Token Means. ACM Trans. Graph. 32, 4, Article 53 (July 2013), 8 pages. DOI = 10.1145/2461912.2461985 http://doi.acm.org/10.1145/2461912.2461985.

DOI: http://doi.acm.org/10.1145/2461912.2461985

1 Introduction

For thousands of years handwritten documentation has been a primary method for communication. The common use of paper and pencil provides an intuitive and simple user interface for creating a wide variety of artifacts from everyday notes to technical documents and even artistic illustrations. While pencil and paper have proven to be very versatile, it takes numerous years of study to learn how to write legibly. Even after significant schooling many people's notes are still difficult to read without being carefully and slowly written.

Recently, there has been an increase in interest in the tablet form factor for computers, which was lead by an associated interest in alternative methods for user interaction beyond a keyboard and mouse. These include the use of multi-touch and stylus input. The use of a stylus with a tablet computer closely mirrors that of pencil and paper, while providing the ability to re-imagine and improve the experience.

In this paper, we propose a novel approach to beautifying handwritten notes using a tablet and stylus. Our approach relies on one main insight. The *average* of multiple instances of a handwritten word or shape is in general better than the individual instances. For example, Figure 1 shows several examples of words or shapes being written multiple times. Notice the significant variation in each instance. If we pull the strokes towards the mean, a more consistent and pleasing result is achieved. Furthermore, when averaging is preformed throughout a document variation in the writing is reduced, which increases its overall readability and visual quality. Thus we gain back some of the benefits of typed text, while still maintaining the versatility and ease of use of stylus-based input.

Our approach possesses several critical properties. First, the end result are notes in the user's handwriting. That is we do not perform handwriting recognition followed by replacing the user's handwritten text with typed text. In fact, we do not perform handwriting recognition at all. Second, we improve the appearance of many

written forms, such as text, Chinese characters or commonly drawn shapes. Previous approaches have either focused exclusively on text [Simard et al. 2005; Zanibbi et al. 2001], shapes [Pavlidis and Van Wyk 1985; Igarashi et al. 1997; Arvo and Novins 2000] or drawings [Dixon et al. 2010; Orbay and Kara 2011; Thiel et al. 2011; Lee et al. 2011; Limpaecher et al. 2013]. The exception is the work of [Lu et al. 2012] which uses a data driven approach to style transfer that may be used to beautify strokes from both drawings and writings. Since we do not rely on handwriting recognition, our approach performs well on notes that contain many types of drawn items, and is agnostic to the user's language. Finally, we propose a real-time method that transitions a user's original strokes into refined strokes shortly after they are written.

We represent the user's strokes using overlapping fixed length sequences of stroke samples called tokens. The tokens represent parts of words or shapes and contain information about both strokes (pen down) and the spacing between strokes (pen up). The samples within a token may only include a part of a single stroke or may span multiple strokes as shown in Figure 1. We perform beautification by averaging a user's most recently written token with similar previously written tokens. It is critical to perform this averaging operation using the correct stroke sample representation. An average within the space must produce a result that agrees with what we would intuitively understand as an average. For instance, the average of multiple handwritten instances of the word "stylus" should still look like "stylus", Figure 1. Our solution is a stroke sample representation using curvature-based sampling [Whitney 1937; Mokhtarian and Mackworth 1992; Dudek and Tsotsos 1997]. This representation along with clustering allows for the efficient real-time search of similar tokens in a large database of user stroke samples. Results are shown for numerous users writing various forms of text and shapes. Our user studies show a clear improvement in the consistency and quality of a user's handwriting.

2 Related work

Numerous papers have explored methods for improving a user's handwriting or drawings. The early work of [Pavlidis and Van Wyk 1985] explores automatic techniques to beautifying geometric drawings by enforcing various relations, such as the collinearity of lines or the similarly of their length. A similar approach was proposed in [Igarashi et al. 1997] except they offer the user several choices when beautifying. Fluid Sketches [Arvo and Novins 2000] interactively morphs a user's drawings into ideal geometric shapes using a dictionary of pre-defined basic shapes. Recently, [Orbay and Kara 2011] propose a sketch beautification approach that expands upon these approaches using a method that automatically learns how to parse a drawing, and [Baran et al. 2010; Thiel et al. 2011] both propose methods to smooth drawn curves while maintaining detail. In [Zanibbi et al. 2001] they explore the improvement of handwritten mathematical expressions by first interpreting the written equations and then warping them to improve readability, while [Simard et al. 2005] normalize text for better alignment. Similar to our approach both of these methods attempt to maintain the user's style. An alternative approach was proposed by [Lu et al. 2012] that uses style transfer to beautify strokes. They propose first collecting a set of high-quality strokes from a trained artist, which may then be transferred to the strokes of a novice's drawings or writings. While this approach could potentially be used to beautify a user's own handwriting, [Lu et al. 2012] focus primarily on style transfer. In addition, [Lu et al. 2012]'s technique only refines the strokes themselves; in contrast, our method also models the spacing between strokes, which we found important in beautifying a variety of writing styles and other marks.

Taking advantage of domain specific knowledge, iCanDraw [Dixon et al. 2010] aids user's in drawing faces. Recently, ShadowDraw [Lee et al. 2011] helps a user sketch by interactively matching their drawing to images. The images provide suggestive contours to the user to guide their strokes in a manner similar to tracing. The concurrent work of [Limpaecher et al. 2013] uses crowdsourcing to improve sets of inherently aligned drawings produced by tracing the same image. They use a similar observation as in our work that the average or consensus of numerous drawings is in general better than the individual drawings. Unlike the approaches described above, our approach does not use predefined shapes, images, parsers or recognizers for beautification. As a result, our approach may be used in a wide variety of scenarios.

Methods for aiding 3D drawing and sculpting are also popular. In [Rivers et al. 2012] they propose an interface for enabling an unskilled user to create a 3D physical replica of a 3D model. Teddy [Igarashi et al. 1999], which was improved upon by [Karpenko and Hughes 2006], allows a user to create 3D objects by drawing a 2D sketch. A survey of similar approaches may be found in [Olsen et al. 2009]. A method for drawing in 3D was recently proposed by [Schmid et al. 2011].

Related to the improvement of hand-drawn sketches is the study of what lines people draw when sketching. In [Cole et al. 2012] they explore the lines drawn to convey the shape of 3D objects, while [Eitz et al. 2012] study how humans sketch various objects.

While our work does not involve handwriting recognition, we borrow several concepts and ideas from this area. Handwriting recognition may be split into two groups, online and off-line approaches [Plamondon and Srihari 2000]. Online approaches [Bahlmann et al. 2002; Graves et al. 2009; Bahlmann and Burkhardt 2004] use as input the coordinates of the stylus as it moves in time. Off-line approaches [Senior and Robinson 1998; LeCun et al. 1998; Plötz and Fink 2009] use either rendered or scanned pixel data from written characters with no temporal information. Handwriting databases [Guyon et al. 1994] typical store online stylus input using strokes sampled at even time intervals, while many handwriting approaches resample the strokes an even spatial intervals [Jain and Namboodiri 2003]. In contrast we propose a curvature-based approach to resampling that has specific benefits for our application.

Similar to our method, many online approaches use a combination of both online and off-line features [Graves et al. 2009; Vinciarelli and Perrone 2003]. These include online methods for stroke alignment using dynamic programming [Bahlmann and Burkhardt 2004; Jain and Namboodiri 2003], and features computed from off-line stroke renderings [Graves et al. 2009]. Our approach uses online features for fast and efficient retrieval of potential matches, and off-line features for later verification. Another area related to our work is the online retrieval of handwritten text [Jain and Namboodiri 2003; Jawahar et al. 2009], which also uses similar representations to those used by handwriting recognition.

Our approach has some interesting similarities to the non-local means approach to image de-noising [Buades et al. 2005]. Non-local means de-noises an image patch by finding similar image patches and averaging them together, much in the same way we search for similar stroke tokens.

3 Approach

In this section, we describe our pipeline for beautifying a user's handwritten notes using the temporal information provided by a stylus. We begin by discussing our stroke representation using a curvature-based resampling of the raw stylus input. Next, we describe how we refine a user's strokes using a set of matching tokens. A token is defined as a sequence of stroke samples. For instance, a token may represent a simple shape or several written letters, Figure 1. Finally, we describe how we efficiently find sets of matching

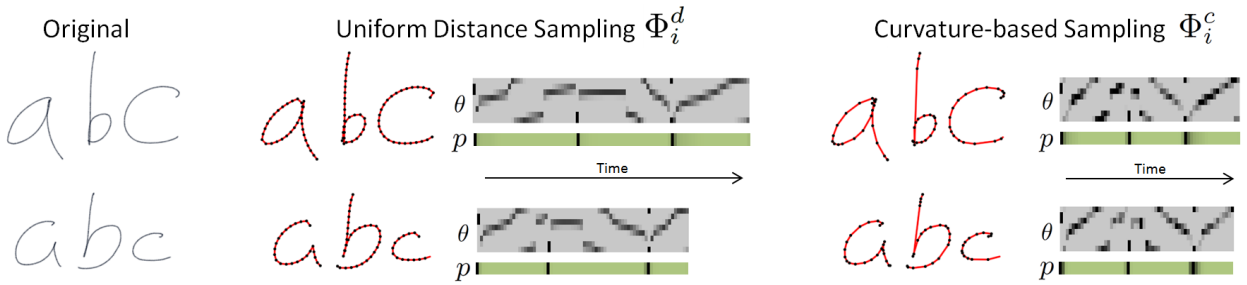

Figure 2: *Illustration of stroke resampling using uniform distance resampling and curvature-based resampling. The resampled strokes are shown in red. The grey plots illustrate the change in orientation and magnitude of the stroke samples through time. The amount of darkness indicates the stroke's magnitude. The stroke pressure is shown in green. Notice the samples from the curvature-based approach are more concise and better aligned.*

tokens in a large token database using a clustering based approach.

We assume we are given a sequence of raw stylus input indicating the stylus's position and pressure over time. We represent the stylus's samples by storing the difference vectors between the stylus positions i.e. $\Phi = \{\phi_1, \ldots, \phi_a\}$ with $\phi_i = \{x_i, y_i, p_i\}$ where (x_i, y_i) is the difference in the stylus's pixel position between samples $i-1$ and i. p_i is the stylus's pressure. We denote the magnitude and orientation of these difference vectors as $r_i = \|x_i, y_i\|$ and $\theta_i = \arctan(y_i, x_i)$ respectively. We make no assumptions about the stylus being uniformly sampled in space or time. We assume that a pressure of $p_i = 0$ indicates the stylus is not in contact with the screen.

3.1 Stroke resampling

In this section we discuss several methods for stroke resampling. That is, computing alternate sets of stroke samples similar to Φ that may have additional properties, while still accurately representing the original motion of the stylus. Many handwriting recognition approaches have either resampled strokes at uniform distance or temporal intervals. We represent samples taken at regular distance intervals using $\Phi^d = \{\phi_1^d, \ldots, \phi_n^d\}$ where the sample magnitude r_i is constant for all samples. An illustration of uniform distance sampling is shown in Figure 2. Notice uniform distance sampling needs to be fine-grained in order to capture writing details.

While uniform distance or temporal sampling may be adequate for handwriting recognition, for stroke refinement we would like stroke samples to possess two additional properties. First, we want a concise representation to reduce memory requirements. Second, for two sequences of stroke samples representing different instances of the same written word or shape, samples in the same position in the sequence should roughly correspond to the same position in the written word or shape. As we demonstrate in following sections, having stroke samples aligned in this manner allows for their efficient matching.

To accomplish both of these goals, we adapt a curvature-based approach to resampling [Whitney 1937; Mokhtarian and Mackworth 1992; Dudek and Tsotsos 1997]. We compute a stroke representation $\Phi^c = \{\phi_1^c, \ldots, \phi_n^c\}$ with the same parameterization as Φ, i.e. $\phi_i^c = \{x_i, y_i, p_i\}$. Instead of sampling at uniform distances, we sample based on the amount of curvature. That is, we increase the sampling density in areas of high curvature and reduce it in areas of low curvature. For instance a straight line with no curvature may be represented by a single sample, where a circle with high curvature will need many samples. We sequentially resample the strokes by mapping each of the raw samples ϕ_i to a sample ϕ_j^c in the curvature-based representation. The mapping is computed using $j = \lfloor z_i \rfloor$, where

$$z_i = z_{i-1} + \min(1, \frac{\alpha \Delta_\theta \beta_j}{2\pi}). \quad (1)$$

$\Delta_\theta \in [0, \pi]$ is the absolute difference between orientations θ_{i-1} and θ_i from samples ϕ_{i-1} and ϕ_i. The value of α controls the density of sampling. Specifically for the case of drawing a circle, α samples will be generated. We found $\alpha = 24$ to produce minimum visual artifacts, while still producing a concise sampling. The value of z_i is incremented by at most 1 to avoid entries in Φ^c with zero valued magnitudes. To account for errors introduced by the discretization of the stylus position, β_j reduces the increase in z if the stroke magnitude r_j is currently small, $\beta_j = \max(0, \min(1, r_j - \sqrt{2}))$. Given $j = \lfloor z_i \rfloor$, the raw stroke sample $\phi_i = \{x_i, y_i, p_i\}$ is added to $\phi_j^c = \{x_j, y_j, p_j\}$ to create a new sample $\{x_i + x_j, y_i + y_j, (r_i * p_i + r_j * p_j)/(r_i + r_j)\}$. To avoid the mixing of stylus inputs that are touching and not touching the screen, if a change in surface contact is detected the value of z_i is incremented by one. Finally while the stylus is not in contact with the screen, z_i is not incremented.

In Figure 2, we show an example of letters written by a user. Notice that the curvature-based resampling requires fewer samples and the stroke samples are better aligned than when using uniform distance resampling. Next, we describe how we group the stroke samples Φ^c into tokens for handwriting beautification.

3.2 Refining strokes

In this section, we describe how a set of stroke samples are refined. When a user writes they generate a large set of stroke samples, denoted Φ (for the rest of the paper we assume a curvature-based sampling and drop the superscript c.) From Φ we create overlapping fixed length sequences of stroke samples called *tokens*, Figure 1. Each token contains n stroke samples. A token T_i represents the stroke samples from ϕ_i to ϕ_{i+n-1}, $T_i = \{\phi_i, \ldots, \phi_{i+n-1}\}$. For notational convenience, we refer to the jth sample in T_i as $t_{i,j} = \phi_{i+j}$. A token is created for every i, resulting in each sample ϕ_i belonging to n tokens.

Choosing the correct length n for the tokens is a critical decision. We found that sequences of stroke samples spanning two or three characters provides enough information to localize similarly written tokens, while still generalizing across multiple words. That is, a single letter would not be sufficient, since how a user writes a letter is generally dependent on the one or two letters proceeding it. If larger windows are used that cover entire words or multiple words, the likelihood of finding matching tokens would decrease greatly, since words such as "mountain" rarely occur. However, its subparts such as "oun" or "ain" do commonly occur. We use $n = 31$,

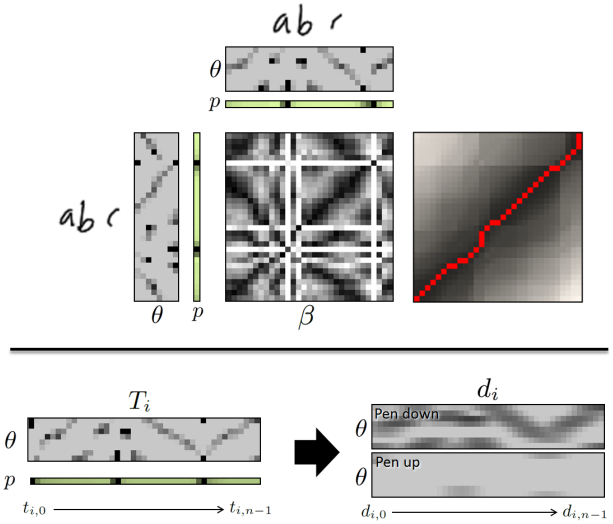

Figure 3: *Illustration of the fine alignment of two tokens and the creation of a descriptor. (top) The stroke samples' magnitude and orientation for each token are illustrated in grey and the pressure in green. The cost matrix β is shown to the left with the minimum cost path on the right in red. (bottom) The stroke samples are shown ordered temporally from left to right (left). The vertical dimension is their orientation and the darkness indicates their magnitude. The pressure of each stroke sample is shown in green. The descriptor d_i used for coarse matching splits the magnitudes into pen down and pen up samples (right).*

which roughly corresponds to two or three characters when writing text as shown in Figure 1. It is worth noting that our token representation models both strokes (pen down) and the spacing between strokes (pen up), unlike previous approaches [Lu et al. 2012] that only beautify the strokes but not the spacing between strokes. In the following sections, we use normalized stroke magnitudes, $\hat{r}_i = r_i/\eta_i$, $\hat{y}_i = y_i/\eta_i$ and $\hat{x}_i = x_i/\eta_i$, to remove scale variation between tokens drawn by the user. η_i is a Gaussian weighted running average of the visible stroke magnitudes in Φ.

Our approach to stroke beautification relies on first finding sets of similar tokens, followed by averaging them to reduce variation in the user's handwriting. We hold off on discussing our approach for automatically localizing similar tokens to Section 3.3. In this section we assume similar tokens have been found and we describe our two stage process for combining similar tokens. First, we finely align the stroke samples in the tokens. When using the curvature-based representation the strokes are coarsely aligned, but small amounts of drift may result from subtle differences in how the tokens are drawn or discretized. Second, we compute our refined token by averaging the aligned stroke sets.

Fine-scale alignment We use an efficient dynamic programming technique for fine-scale alignment, similar to approaches that have been used for stereo vision [Belhumeur 1996] and for handwriting recognition [Bahlmann and Burkhardt 2004; Jain and Namboodiri 2003]. Let us assume we have two tokens T_i and T_j that correspond to the same shape or letters. First, we compute an $n \times n$ matrix of match costs β. The match cost $\beta_{k,l}$ measures the similarity of two stroke samples $t_{i,k}$ and $t_{j,l}$ with a lower score indicating a better match. We compute an alignment by finding the least cost path from $\beta_{1,1}$ to $\beta_{n,n}$ using three types of moves $\{(0,1),(1,0),(1,1)\}$. A move to index (k,l) has cost $\beta_{k,l} + \xi$, where ξ is used to favor diagonal moves. Specifically, $\xi = 0$ for a $(1,1)$ move, and $\xi = 0.2$ otherwise. The globally optimal path corresponding to the mapping of stroke samples from T_i to T_j can be found using dynamic programming with forward and backward passes in $O(n^2)$ time. An example with the computed optimal path is shown in Figure 3.

The match cost $\beta_{k,l}$ is found using a linear combination of three features,

$$\beta_{k,l} = \Delta_{\hat{r}} + \Delta_\theta + \delta_p, \tag{2}$$

computed from $t_{i,k}$ and $t_{j,l}$. $\Delta_{\hat{r}}$ is the absolute difference between $\hat{r}_k$ and $\hat{r}_l$. Δ_θ is the absolute angular distance between θ_k and θ_l. δ_p measures whether both strokes have consistent visibility. That is, $\delta_p = 1$ if $p_k = 0$ and $p_l = 0$, or $p_k > 0$ and $p_l > 0$, and $\delta_p = 0$ otherwise.

Merging stroke sets Once two or more tokens are aligned, we can merge them by averaging the stroke samples. For two stroke samples t_{ik} and t_{jl}, their average is $\{(\hat{x}_k + \hat{x}_l)/2, (\hat{y}_k + \hat{y}_l)/2, (\hat{r}_k * p_k + \hat{r}_l * p_l)/(\hat{r}_k + \hat{r}_l)\}$. In Figure 1, we show several examples of rendered tokens after merging. Notice the improvement in the quality of the token, while still maintaining the writing style of the user.

3.3 Finding similar tokens

In the previous section, we described how we combine matching tokens to compute a refined token. In this section we describe how we find corresponding tokens that may be merged for token refinement in real-time. We accomplish this using a two staged approach. First, a coarse search is performed on sets of clustered tokens. Second, each cluster found during the coarse search is verified and assigned a confidence weighting. The final result is found using a weight average of the clusters' means and the user's written token, Section 3.2. We begin by discussing how we cluster tokens.

Token clustering Using the stroke samples Φ generated by the user, we cluster tokens T_i for all i. For clustering, a distance metric needs to be defined between the tokens representing the cluster means and the tokens written by the user. We could use a simple distance measure such as that used by Equation (2), but the stroke samples would need to be finely aligned, which would be computationally prohibitive. Instead, we use an L^2 distance between token descriptors that blurs the values temporally to provide robustness to small temporal shifts.

Our descriptor d_i linearly splits the stroke sample magnitudes $\hat{r}$ into histograms based on their orientation θ, and applies Gaussian blur along the temporal dimension, Figure 3. Two histograms are created corresponding to strokes when the stylus is in contact with the screen (pen down) and when it is not (pen up). Since changes in small strokes are in many cases as visually salient as changes in large strokes, we use the logarithm of the blurred magnitudes to equalize changes across scale. Finally, the descriptor's values are weighted by a temporally centered Gaussian.

Clustering is performed using an online approach in which a token is merged with a cluster if its distance is below a threshold τ, otherwise a new cluster is created. For a matching cluster c_j, we merge the token T_i with the cluster's mean token $\Psi_j = \{\psi_{j,0}, \ldots, \psi_{j,n-1}\}$, where $\psi_{j,k}$ is the kth stroke sample in Ψ_j. Merging is performed using the same technique described in Section 3.2 using fine-alignment and averaging. A new cluster descriptor is computed given the updated mean token.

Coarse search Given the most recently drawn token T_i by the user, we compute a descriptor d_i and find its L^2 distance to the descriptors of the clusters' token means. As we describe later, in practice we use temporal prediction to help reduce the computational cost of this step. If a cluster is within a distance τ_v, $\tau_v > \tau$, of the

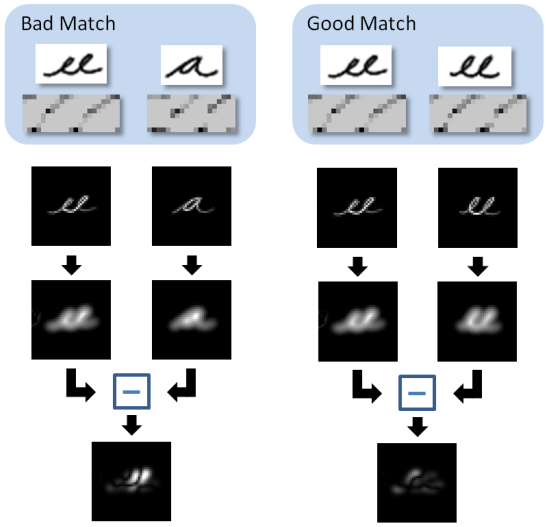

Figure 4: *An example of a bad and good match between tokens. The descriptors d_i are very similar for a cursive "a" and "ee". However, if we perform a raster rendering of both tokens and blur them, their difference is large. In contrast, a correct match results in a smaller difference between the blurred raster renderings.*

cluster mean, the cluster is added to a list of potential matches, m_i. We use a threshold slightly greater than τ to obtain a larger set of potential candidates for later verification. In practice, the average number of potential candidates is typically between 4 and 12 out of a potential of $1,000$'s of clusters.

Token verification For verification, we compute whether a written token T_i and a cluster mean Ψ_j are indeed similar. A match confidence score $\lambda_{i,j}$ is found using a complementary token descriptor to d_i. Our previous token descriptor d_i encodes online temporal information based on the order in which the stroke samples were drawn. However, two descriptors d_i and d_j might be quite similar but the tokens may produce visually different renderings. For instance, Figure 4 shows an example of two tokens that are written using similar stroke orderings, but a slight change in the stroke sample angles changes the visual appearance from an "a" to "ee". To account for this phenomenon, we verify matching strokes using a descriptor based on how the tokens are actually rendered.

We compute our confidence $\lambda_{i,j}$ in a match between T_i and Ψ_j using a low-resolution rendering of the tokens, Figure 4. The intensity of the rendered strokes are weighted by their temporal distance to the center of the token. We compute the L^2 distance between the renderings after applying a small amount of blur. The confidence score is computed using a normal distribution on the L^2 distance. Before the L^2 distance is computed, both of the rendered tokens are spatially centered.

Refining stroke samples Since we create tokens from overlapping windows of stroke samples, a stroke sample ϕ_i belongs to n different tokens, i.e. $T_{i-(n-1)}$ to T_i. Each of these tokens T_j with $j \in [i-(n-1), i]$ has its own set of candidate cluster matches m_j with corresponding confidence values $\lambda_{j,k}, k \in m_j$. For the cluster mean Ψ_k, the sample $\psi_{k,l} \in \Psi_k$ with $l = i - j$ will contribute to the refinement of stroke sample ϕ_i. The weight w_{ijk} assigned to sample $\psi_{k,l}$ is computed using,

$$w_{ijk} = \sum_{j \in [i-(n-1),i]} \sum_{k \in m_j} \lambda_{j,k} \mathcal{N}(l; \frac{n}{2}, \sigma), \tag{3}$$

where $\mathcal{N}$ is the normal distribution with mean $n/2$ and standard deviation $\sigma = n/6$. The use of a Gaussian weighting ensures a smooth transition between the contribution of various clusters in the refinement. Using a weighted average, the refined stroke sample $\tilde{\phi}_i = \{\tilde{x}_i, \tilde{y}_i, \tilde{p}_i\}$ is computed using

$$\tilde{x}_i = \frac{\hat{x}_i + \sum_{j \in [i-(n-1),i]} \sum_{k \in m_j} w_{ijk} s_k \hat{x}_l}{1 + \sum_{j \in [i-(n-1),i]} \sum_{k \in m_j} w_{ijk} s_k}, \tag{4}$$

and similarly for $\tilde{y}_i$. s_k provides higher weight to larger clusters, and is equal to the square root of the cluster's size. The pressure $\tilde{p}_i$ is computed similarly.

3.4 Temporal cluster prediction

As described above, when refining strokes we find the nearest cluster to the currently drawn token. Done naïvely, we would need to compare the current token to every cluster mean. Since the number of cluster means may range in the $10,000$'s, this can be computationally expensive. To improve performance, we use temporal information to predict which clusters are likely to occur next.

Our task is to find the closest clusters to a written token T_i. When performing online stroke refinement, we assume that the closest cluster c_j to T_{i-1} has already been found. Since our tokens are created using a sliding window, T_i and T_{i-1} have $n-1$ overlapping, but shifted stroke samples. Similarly, we may predict the clusters that match T_i by finding clusters means that match the mean Ψ_j of cluster c_j shifted by one sample. When a new cluster is added, we find a set of likely adjacent clusters by shifting its descriptor temporally by one stroke sample and comparing it to other clusters. Any cluster whose mean is less than a distance of τ_a, $\tau_a > \tau$, is then added to a list of adjacent clusters. Thus, when searching for possible matches to T_i, we only look at clusters predicted by c_j. In practice we find this approach greatly reduces the number of clusters that need to be compared against from thousands to 40-60.

The remaining computationally expensive process is adding a new cluster, since we would need to compare the new cluster mean to all of the other cluster means to find a set of adjacent clusters. To reduce computation, we use a random sampling approach to finding adjacent clusters. Given a new cluster c_k, we randomly select 300 other clusters. If a cluster mean Ψ_j is close to Ψ_k, we search all of c_j's adjacent clusters to find adjacent clusters for c_k. In practice, this reduces the number of mean comparisons to 300 to 500 when adding new clusters with minimal loss in accuracy.

4 Online rendering

In the previous section, we described how we compute a refined set of stroke samples $\tilde{\Phi}$ from the original samples Φ. We now describe our method for rendering the refined samples to the user. When we compute the refined sample $\tilde{\phi}_i$, we need samples from Φ that occur after $\tilde{\phi}_i$ to find similar tokens, Section 3.3. Thus, there is a temporal lag between when a user writes a stroke sample and when its refined version is computed. To account for this, we use a method that slowly warps the stroke samples Φ to $\tilde{\Phi}$ as the user writes. Specifically, we linearly blend between Φ and $\tilde{\Phi}$ over 30 stroke samples. The result may be viewed in the supplementary video.

When rendering the refined strokes, we desire them to occupy the same rough spatial position and scale as the original strokes. Before rendering, the magnitudes of the refined stroke samples are multiplied by η_k to return them to their original scale. Due to the accumulation of small differences in the stroke samples it is possible for the refined stroke samples to drift from the original stroke

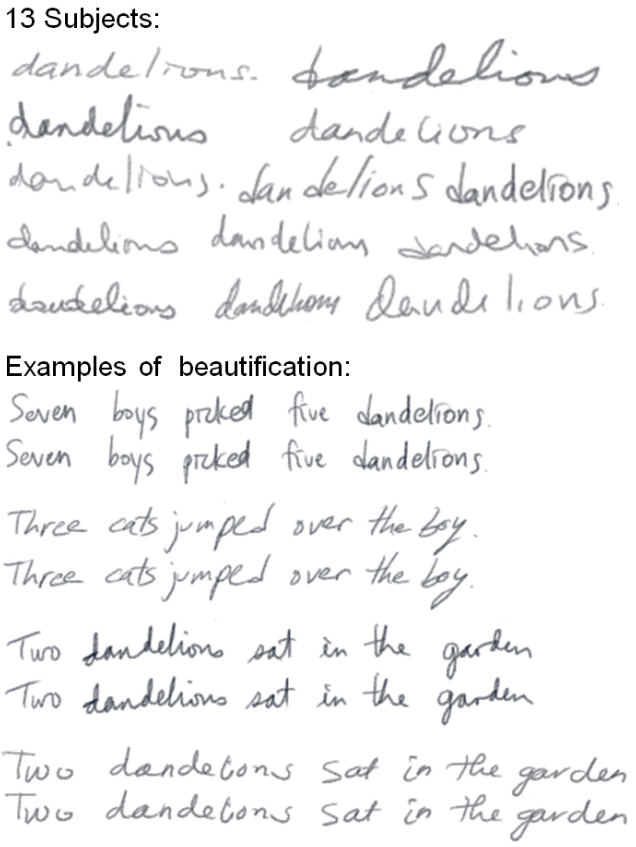

Figure 5: *Examples of the handwriting user study. (Top) 13 examples of the different subjects writing "dandelions". Notice the wide variation in writing styles. (Bottom) Examples of text beautification. The top sentence is the original and beautification has been applied to the bottom sentence.*

sample absolute positions. To account for this we compute the average position of the original strokes and the refined strokes, and shift the refined strokes by their difference. This average ensures the low-frequency placement of the strokes are the same without removing the higher-frequency and visually salient changes of the refined strokes.

5 Results

In this section, we discuss results on improving handwritten text, shapes and doodles. We begin by showing quantitative results of a user study in which we collected a wide variety of writing samples. Next, using a larger database of writings, we analyse the memory and computational performance of our approach. Finally, we show some qualitative results.

In all of our experiments we used the following parameter settings, which were experimentally found to produce accurate results while maintaining computational efficiency. For the descriptor d the magnitudes are split into 8 orientation bins, and temporally blurred with a Gaussian, $\sigma = 1.5$ samples. The descriptor is weighted temporally with a Gaussian, $\sigma = 15$ samples. The resulting descriptor has $n \times 16$ values, Figure 3.

For verifying matching tokens, we compute the pixel difference between their renderings, Figure 4. Before computing the difference we Gaussian blur the renderings with a $\sigma = 1.5$ pixels. The intensity of the rendered strokes is weighting using a temporally centered

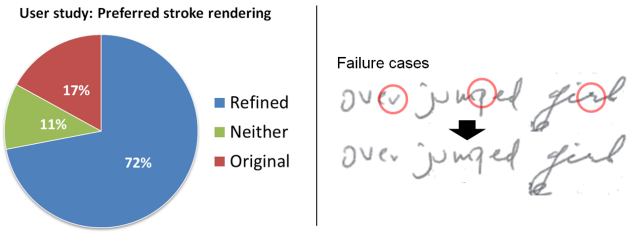

Figure 6: *The results of the handwriting user study. (Left) Chart showing users preference for the original strokes and beautified strokes. 72% of the refined or beautified renderings where preferred over the original stroke renderings, where 17% of the original renderings were preferred. For each of the 13 handwriting examples, a majority of subjects preferred the refined strokes. (Right) Example failure cases from the user study.*

Gaussian with $\sigma = 5$ samples. The confidence score is computed using a normal distribution on the L^2 distance with $\sigma = 0.25$.

The values of the cluster merging threshold τ and token size n are critical parameters. τ should be chosen carefully to ensure tokens representing different letters or shapes are not merged together, while still merging tokens as often as possible to maintain computationally efficiency. We used a value of $\tau = 1.5$ and found values ranging from $\tau = 1.4$ to 1.8 produced reasonable results. We use values of $\tau_v = 1.1\tau$ for finding sets of tokens for verification, and $\tau_a = 1.2\tau$ for computing cluster adjacency sets.

5.1 Handwriting user study

We designed a user study to see if our approach to handwriting beautification would work across different writing styles, including both printed and cursive text. We asked 13 subjects to write 20 sentences. The sentences where chosen to contain similar words, so that similar tokens could be found. The subjects used a Wacom Cinteq 24HD pen display. The subjects' strokes were rendered in a pencil style on ruled paper. Figure 5 shows examples from the 13 subjects. Notice the large amount of variation in their writing styles. We applied our online approach to handwriting beautification to each of the examples. We then asked 8 of the original 13 subjects to judge whether the original unmodified strokes or the refined strokes produced by our approach was "easier to read and more appealing to look at." The last 6 out of 20 sentences written by the 13 subjects were shown for comparison. The subjects had to pick between example "A", "B" or "Neither". The original and refined strokes were randomly assigned to "A" and "B" and the ordering of the examples was randomized. For each of the 13 handwriting examples, a majority of subjects preferred the refined strokes. As shown in Figure 6, the subjects liked the refined strokes in 72% of the examples, they didn't have a preference in 11% of the examples, and they favored the original strokes in 17% of the examples. Given the relatively small number of sentences written by the subjects, it is impressive that even when refining strokes with a small number of potentially matching tokens an improvement was seen. Except for a few words such as "the", most words only occurred 4-6 times. Examples of our text beautification and failure cases are shown in Figures 5 and 6.

5.2 Performance evaluation

In this section, we analyse both the memory and computational performance of our approach. Specifically, we study how the size of the database and number of clusters grows as a user writes more strokes. To study these questions, we asked a subject to write the first 2,000 words of "Crime and Punishment" by Fyodor Dos-

toyevsky using 9 separate sessions. This resulted in over 75,000 stroke samples.

We begin by looking at the number of clusters created. After 75,000 samples our online clustering algorithm finds over 15,000 clusters. However, a vast majority of these clusters are created but never used for stroke refinement, since they are never matched to any other token. We measured the percentage of weights w_{ijk} in Equation (4) that contribute to stroke refinement in the top k clusters, $k = \{100, 1000, 5000, 10000\}$. We found that after 75,000 stroke samples 97% of the contribution to stroke refinement is made by the top 5,000 clusters, while 31%, 69% and 100% is made by the top 100, 1,000 and 10,000 clusters respectively. Given this finding, we periodically remove clusters when adding new strokes. Specifically we limited the number of clusters to 6,000. Another way to judge how many clusters should be used is to look at the number of letter triplets possible in the English language. By this measure we should have $26 \times 26 \times 26 = 17,576$ clusters. Obviously, many of these triplets will never be used, some may be written in multiple ways, and many tokens may not correspond to written text. However, having between 5,000 and 10,000 clusters appears adequate for capturing most of the variation in a user's writings and drawings.

Using 6,000 clusters our approach can process new strokes at a rate of 300Hz to 500Hz. This corresponds to roughly 500 words per minute, which is at least an order of magnitude faster than a person can write. Experiments were run on an Intel i5 2.53GHz CPU with 4GB RAM. The implementation is written for efficiency, but is not highly optimized, more specifically no hardware acceleration was used. The memory requirements for storing the database of clusters and their descriptors is less than 20MB.

5.3 Qualitative results

In Figure 1, we show several qualitative results of our approach before and after stroke refinement. The figure shows words, shapes, and doodles, cut from a larger document. Notice how the beautified results across a wide range of different input styles are more consistent, and for words easier to read. To illustrate how our online approach progressively improves written tokens, we show several examples in Figure 7. Notice how initially the database is empty so no stroke refinement is performed. However, as the same word or shape is written multiple times the algorithm finds matching tokens and the strokes are refined. For a better visualization of the differences and more results, please see the supplementary video.

6 Discussion

Currently, our approach only matches sets of stroke samples within individual tokens. In future work, longer range and more complex interactions could be explored. For instance, spelling correction may automatically be performed if we took advantage of relations between multiple tokens. While we developed our efficient search algorithm for beautification, the approach may be used for other purposes as well. For instance a user may search their notes using stylus input. The clusters may also be useful if handwriting recognition was desired, since recognition could be performed on the clusters instead of individual tokens.

One area that we did not explore in this paper is whether a stroke database created by one user may be useful for beautifying the written strokes of another user. Currently, our approach has a "cold start" problem, in that we cannot beautify a person's strokes until we have a database large enough to find matching tokens. However as shown in Figure 7, only a few matching tokens are needed to start seeing improvement. One possible failure case is the system may not be able to reliably match hastily written strokes by a user to those more careful written if the differences are too large.

In this paper we propose a general purpose method for handwriting beautification. This includes numerous forms of written text and shapes. This is enabled by the insight that the average of multiple matching tokens is in general more appealing than individual instances. This holds true not just for text, but for shapes and other drawings. As a result, our approach does not need predefined shapes [Pavlidis and Van Wyk 1985; Igarashi et al. 1997; Arvo and Novins 2000], handwriting recognition [Plamondon and Srihari 2000] or other parsers [Zanibbi et al. 2001; Simard et al. 2005]. We demonstrated the effectiveness of our approach with a user study showing beautification across a diverse collection of handwriting styles and by showing numerous qualitative examples.

References

ARVO, J., AND NOVINS, K. 2000. Fluid sketches: continuous recognition and morphing of simple hand-drawn shapes. In *In Proc. of UIST*, ACM.

BAHLMANN, C., AND BURKHARDT, H. 2004. The writer independent online handwriting recognition system frog on hand and cluster generative statistical dynamic time warping. *PAMI 26*, 3.

BAHLMANN, C., HAASDONK, B., AND BURKHARDT, H. 2002. Online handwriting recognition with support vector machines-a kernel approach. In *Proc. of Work. on Frontiers in Handwriting Recognition*, IEEE.

BARAN, I., LEHTINEN, J., AND POPOVIĆ, J. 2010. Sketching clothoid splines using shortest paths. In *Computer Graphics Forum*, vol. 29.

BELHUMEUR, P. 1996. A bayesian approach to binocular steropsis. *IJCV 19*, 3.

BUADES, A., COLL, B., AND MOREL, J. 2005. A non-local algorithm for image denoising. In *In Proc. of CVPR*, vol. 2.

COLE, F., GOLOVINSKIY, A., LIMPAECHER, A., BARROS, H., FINKELSTEIN, A., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2012. Where do people draw lines? *ACM Trans. Graph. 55*, 1.

DIXON, D., PRASAD, M., AND HAMMOND, T. 2010. icandraw: using sketch recognition and corrective feedback to assist a user in drawing human faces. In *Proc. of the Int. Conf. on Human Factors in Computing Systems*, ACM.

DUDEK, G., AND TSOTSOS, J. 1997. Shape representation and recognition from multiscale curvature. *Computer Vision and Image Understanding 68*, 2.

EITZ, M., HAYS, J., AND ALEXA, M. 2012. How do humans sketch objects? *ACM Trans. Graph. 31*, 4.

GRAVES, A., LIWICKI, M., FERNÁNDEZ, S., BERTOLAMI, R., BUNKE, H., AND SCHMIDHUBER, J. 2009. A novel connectionist system for unconstrained handwriting recognition. *PAMI 31*, 5.

GUYON, I., SCHOMAKER, L., PLAMONDON, R., LIBERMAN, M., AND JANET, S. 1994. Unipen project of on-line data exchange and recognizer benchmarks. In *In Proc. of ICPR*, vol. 2, IEEE.

IGARASHI, T., MATSUOKA, S., KAWACHIYA, S., AND TANAKA, H. 1997. Interactive beautification: a technique for rapid geometric design. In *In Proc. of UIST*.

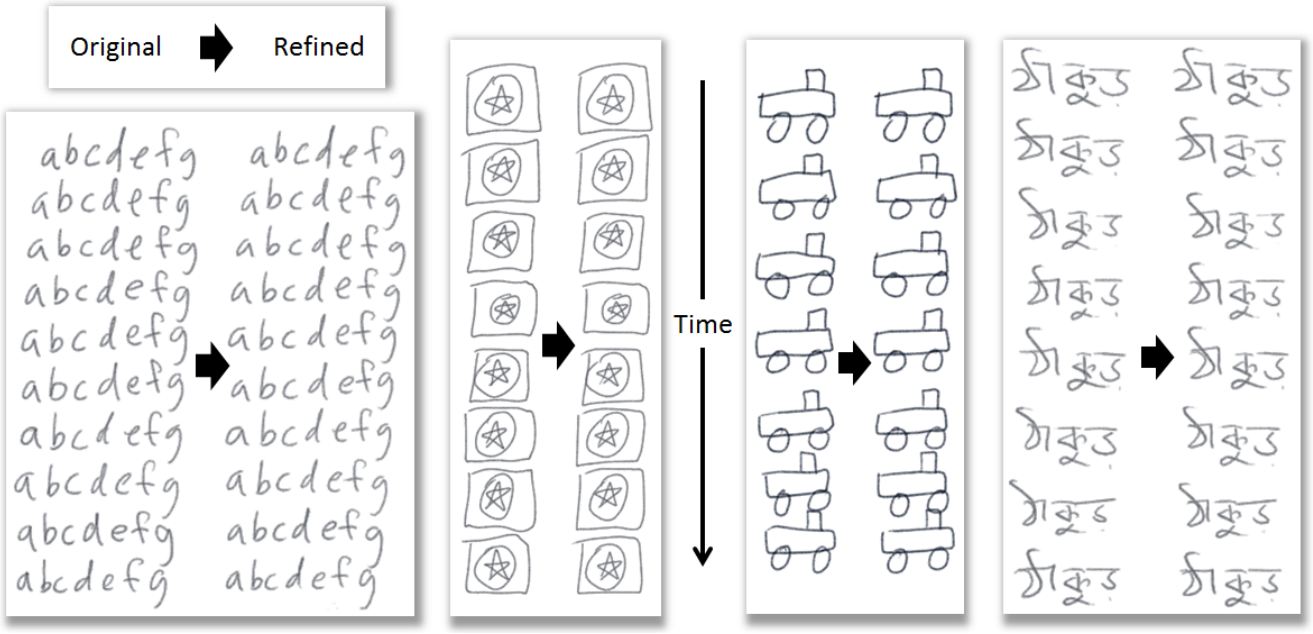

Figure 7: *Several examples of using token means for beautification. Each example was drawn from top to bottom. Notice how the later drawn examples have a higher level of refinement while still maintaining some individuality.*

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *Proc. of the Conf. on Computer Graphics and Interactive Techniques*.

JAIN, A., AND NAMBOODIRI, A. 2003. Indexing and retrieval of on-line handwritten documents. In *Conf. on Document Analysis and Recognition*.

JAWAHAR, C., BALASUBRAMANIAN, A., MESHESHA, M., AND NAMBOODIRI, A. 2009. Retrieval of online handwriting by synthesis and matching. *Pattern Recognition 42*, 7.

KARPENKO, O., AND HUGHES, J. 2006. Smoothsketch: 3d free-form shapes from complex sketches. In *ACM Trans. Graph.*, vol. 25.

LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. 1998. Gradient-based learning applied to document recognition. *Proc. of the IEEE 86*, 11.

LEE, Y., ZITNICK, C., AND COHEN, M. 2011. Shadowdraw: real-time user guidance for freehand drawing. In *ACM Trans. Graph.*, vol. 30.

LIMPAECHER, A., FELTMAN, N., TREUILLE, A., AND COHEN, M. 2013. Real-time drawing assistance through crowdsourcing. *ACM Trans. Graph. 32*, 4.

LU, J., YU, F., FINKELSTEIN, A., AND DIVERDI, S. 2012. Helpinghand: example-based stroke stylization. *ACM Trans. Graph. 31*, 4.

MOKHTARIAN, F., AND MACKWORTH, A. 1992. A theory of multiscale, curvature-based shape representation for planar curves. *PAMI 14*, 8.

OLSEN, L., SAMAVATI, F., SOUSA, M., AND JORGE, J. 2009. Sketch-based modeling: A survey. *Computers & Graphics 33*, 1.

ORBAY, G., AND KARA, L. 2011. Beautification of design sketches using trainable stroke clustering and curve fitting. *TVCG 17*, 5.

PAVLIDIS, T., AND VAN WYK, C. 1985. An automatic beautifier for drawings and illustrations. *Computer Graphics 85 19*, 3.

PLAMONDON, R., AND SRIHARI, S. 2000. Online and off-line handwriting recognition: a comprehensive survey. *PAMI 22*, 1.

PLÖTZ, T., AND FINK, G. 2009. Markov models for offline handwriting recognition: a survey. *Int. J. on Document Analysis and Recognition 12*, 4.

RIVERS, A., ADAMS, A., AND DURAND, F. 2012. Sculpting by numbers. *ACM Trans. Graph. 31*, 6.

SCHMID, J., SENN, M., GROSS, M., AND SUMNER, R. 2011. Overcoat: an implicit canvas for 3d painting. *ACM Trans. Graph. 30*, 4.

SENIOR, A., AND ROBINSON, A. 1998. An off-line cursive handwriting recognition system. *PAMI 20*, 3.

SIMARD, P., STEINKRAUS, D., AND AGRAWALA, M. 2005. Ink normalization and beautification. In *Conf. on Document Analysis and Recognition*, IEEE.

THIEL, Y., SINGH, K., AND BALAKRISHNAN, R. 2011. Elasticurves: exploiting stroke dynamics and inertia for the real-time neatening of sketched 2d curves. In *In Proc. of UIST*.

VINCIARELLI, A., AND PERRONE, M. 2003. Combining online and offline handwriting recognition. In *Conf. on Document Analysis and Recognition*.

WHITNEY, H. 1937. On regular closed curves in the plane. *Compositio Mathematica 4*.

ZANIBBI, R., NOVINS, K., ARVO, J., AND ZANIBBI, K. 2001. Aiding manipulation of handwritten mathematical expressions through style-preserving morphs. Graphics Interface 2001.