Legible Compact Calligrams

Changqing Zou[1,2*] Junjie Cao[3,1*] Warunika Ranaweera[1] Ibraheem Alhashim[1]
Ping Tan[1] Alla Sheffer[4] Hao Zhang[1]

[1]Simon Fraser University [2]Hengyang Normal University [3]Dalian University of Technology [4]University of British Columbia

Abstract

A calligram is an arrangement of words or letters that creates a visual image, and a *compact calligram* fits one word into a 2D shape. We introduce a fully automatic method for the generation of *legible compact calligrams* which provides a balance between conveying the input shape, legibility, and aesthetics. Our method has three key elements: a path generation step which computes a global layout path suitable for embedding the input word; an alignment step to place the letters so as to achieve feature alignment between letter and shape protrusions while maintaining word legibility; and a final deformation step which deforms the letters to fit the shape while balancing fit against letter legibility. As letter legibility is critical to the quality of compact calligrams, we conduct a large-scale crowd-sourced study on the impact of different letter deformations on legibility and use the results to train a letter legibility measure which guides the letter deformation. We show automatically generated calligrams on an extensive set of word-image combinations. The legibility and overall quality of the calligrams are evaluated and compared, via user studies, to those produced by human creators, including a professional artist, and existing works.

Keywords: calligram, legibility, shape deformation

Concepts: •**Computing methodologies** → **Parametric curve and surface models; Shape analysis;**

1 Introduction

A calligram is an arrangement of words or letters that creates a visual image [Wikipedia 2014]. The beauty and elegance of calligrams is well appreciated in many languages. Visually expressing words or phrases while reflecting their meanings has a strong intellectual and artistic appeal. Well-designed calligrams must simultaneously convey the input image and be both readable and appealing [Maharik et al. 2011]. Such calligrams require significant effort and time to create even for experts. An intriguing question is whether calligram generation could be automated.

What makes calligram generation an attractive pursuit in computer graphics is its geometric appeal: embedding an input text into a visual image inevitably involves modeling, as well as manipulating, both the *shapes* and *spatial arrangements* of the letters or strokes which make up the text to fit them into the shape of the input image. Such a processing of the letter or stroke shapes becomes more critical when the input consists of a short text. Long textual calligrams such as micrographies [Maharik et al. 2011] convey the input shapes by aligning lines of text with the shape outlines and only minimally deforming individual letters. For a *compact* calligram formed by one or few words, conveying the input shape requires placing and deforming the individual letters to fit the shape outline and this directly impacts readability or legibility of the words and individual letters. This key consideration poses an inherently different challenge to those addressed by long text layouts.

Figure 1: *A few compact calligrams generated by our algorithm fully automatically. Input images are shown as insets.*

In this paper, we introduce a fully automatic method to generate *legible compact calligrams*, providing a desired balance between conveying the input shape, legibility, and aesthetics. Figure 2 shows several calligrams which fulfill these criteria; they are called *word animals* and were designed by the professional artist Dan Fleming. Note the significant variability in how the letters are deformed and arranged. Coming up with a fully automatic solution to reach a comparable level of artistry is highly challenging.

Following the above goals, we first identify a set of geometric criteria to fulfill. To convey the input shape, we want the outer hull and interior coverage of the deformed letters to closely *fit* the input shape. Previous work [Maharik et al. 2011] indicates that texts are easier to read when letters are laid out along a *smooth low-curvature* path; we refer to this criterion as *text flow*. Next, we account for legibility and aesthetics at the word level, which point to a preference for the letters to be *evenly sized* and *consistently oriented*. Lastly and critically, we seek maximal *letter legibility*.

Aiming to satisfy all these criteria at once leads to an unwieldy optimization, which is impractical to solve directly. Instead, we

Figure 2: *Word animal calligrams designed by the artist Dan Fleming (images taken with permission from the artist).*

*Corresponding authors: {jjcao1231,aaronzou1125}@gmail.com

SIGGRAPH '16 Technical Paper, July 24 - 28, 2016, Anaheim, CA
ISBN: 978-1-4503-4279-7/16/07
DOI: http://dx.doi.org/10.1145/2897824.2925887

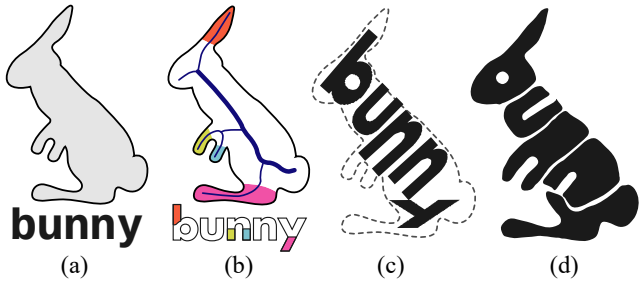

Figure 3: *An overview of our legible compact calligram generation method: (a) an input word and an outline image; (b) text layout path (thick blue curve), color-coded corresponding letter anchors, and outline protrusions; (c) correspondence-based letter alignment; (d) final result achieved by letter deformation driven by a legibility measure learned from crowdsourced data.*

note that our set of criteria naturally leads to a *coarse-to-fine* layout approach, where at each level, we solve an optimization which balances fit against legibility and aesthetics; see Figure 3.

Given a word and an input shape, which is assumed to be a solid binary image, we first search for a layout path whose induced text flow fits the input shape. With word legibility in mind, we focus on *single-line* text flows, like that of the bunny, monkey, or even elephant in Figure 2, but not the penguin or orangutan. The next challenge is to position the letters along the path while balancing the fit criterion against word legibility and aesthetics. For fit at this level, we distribute and scale the letters to align protruding letter features with their counterparts on the input shape boundary; see Figures 3(b-c). Through an optimization with mixed objectives, fit is balanced against legibility and aesthetics considerations reflected by letter sizes, distribution, and orientation.

At the finest and most critical level of our solution pipeline, we collectively deform the letters to locally optimize fit while maximally retaining legibility. This task is the most challenging and least researched. Few, if any, studies exist on the impact of shape deformation on letter legibility. Without any such insights, one cannot arrive at an algorithm to preserve legibility. Our work fills this void. We first conduct a large-scale crowd-sourced study on the impact of different deformations on letter legibility. We use the results of our study to train a letter legibility measure and utilize this measure in the letter deformation step to refine the calligram.

Contributions. Our overall contribution is the first fully automatic method for legible, compact calligram generation. Another contribution is our legibility measure learned from crowdsourced data, which is likely of interest to other text layout problems.

Letter legibility depends on letter shapes and on the types and degree of variations a reader can tolerate; it is thus alphabet- and font-dependent. In our work, we focus on using lowercase letters from the Latin alphabet and train the legibility measure with letters written in freeform handwriting style as well as a simple font suitable for calligram design. We expect our general framework for calligram generation and legibility learning to carry to other alphabets and font styles, but require an additional user study and different selection of features to account for the new letter shapes.

We show automatically generated calligrams on an extensive set of word-image combinations including a variety of shapes. Through several user studies, the legibility and overall quality of these calligrams are evaluated and compared qualitatively to those produced by human creators, including a professional artist, and the most closely related existing method [Xu and Kaplan 2007].

2 Related work

In general, our method for generating legible compact calligrams is related to and builds on existing works on text art layout, packing, 2D shape deformation, and text analysis and recognition.

Text art. Computer graphics research has addressed artistic text layout in a number of contexts. Technologies for placing texts along user-specified paths are widely available in commercial software such as Adobe Illustrator [2010]. These methods are designed for laying out long texts and do not address image fit. Algorithmic generation of text art has been explored in several other contexts. Xu et al. [2010] generate structure-based ASCII art by careful analysis of line structures. They approximate shapes using a limited set of symbols under rigid rules for alignment and positioning. Calligram generation is complementary to this approach as we allow letter shapes and placement to vary more freely to achieve fit. Several software packages, e.g. [Helmond 2010] generate grid-aligned text mosaics. Compact calligrams allow for a greater artistic freedom capturing not only the text but the geometry of an input shape.

Maharik et al. [2011] develop a method for generating micrography images formed by laying out long sequences of text inside a 2D contour. While the high-level criteria they use for the layout are directly applicable to compact calligrams, the technical challenges they face are significantly different. In particular, their core computational problem is to optimize the layout of multiple lines of text to balance aesthetics and readability of the text flows; letter shapes are only minimally altered. Our calligrams are single-line, where the key challenge instead is to embed and deform letter shapes, perhaps drastically if necessary, to achieve a balance between fit and readability of the word and the letters.

Calligraphic packing. To the best of our knowledge, the work of Xu and Kaplan [2007] is the only known prior attempt at generating compact calligrams. They solve the calligraphic packing problem using a semi-automatic technique: the initial position and orientation of the letters are specified by the user, and guide the subsequent segmentation of the input shape and the letter packing. In contrast, our framework automatically computes letter placement, packing, as well as deformation. Unless stated otherwise, all the results shown in the paper have been generated fully automatically. However, as discussed in Section 8, we can easily support user intervention. To account for letter legibility, their work merely employs a shape context-based similarity between the deformed and original letters. This measure leads to inferior results compared to our systematically learned legibility score; see Figure 17.

Non-textual packing and layout. Multiple methods address the problem of filling a region or shape with smaller elements. A photo collage, e.g., [Goferman et al. 2010], packs a set of photo segments into a rectangular frame where the segments may be trimmed or expanded. The work by [Kim and Pellacini 2002] produces a jigsaw mosaic for an arbitrarily-shaped input image by selecting and packing a set of pre-defined image tiles, also of arbitrary shapes. Gal et al. [2007] abstract a given 3D object using an assembly of related shapes. These methods share some similarity with our work in terms of the need to embed input elements into a container, but do not address any of the readability concerns that are critical in the context of calligram generation and text layout in general.

Character recognition. There has been a great deal of work on OCR, as well as the recognition of handwritten digits and characters. The primary goal of recognition or classification is to discriminate between different characters, e.g., the letters ‘q’ vs. ‘g’, rather than answering questions about what makes a given letter more or less legible over shape variations. Our work focuses on the lat-

ter, in particular, the quantification and improvement of legibility of printed, deformed letters. That said, legibility and recognizability are closely related. However, the kind of letter shape distortions OCR or other character recognition methods typically deal with are quite different from those in compact calligrams.

Letter legibility. We are not aware of previous studies on learning or rating letter legibility, and specifically reader sensitivity to different types of alphabet deformation. Works from the psychology community, e.g.,[Loomis 1990; Sheedy et al. 2005; Cai et al. 2008], focus on identifying and testing factors, e.g., fonts, colors, and spatial complexity, which may impact character legibility. For example, through a small-scale user study, Cai et al. [2008] examined how several image descriptors for letters in Times New Roman may influence legibility. The learned factors may help enrich the way we generate training data for our legibility study, but these works do not propose ways to quantitatively measure legibility. Legibility was one of the many *font* attributes considered in [O'Donovan et al. 2014]. Through crowdsourcing, they learn a set of high-level and descriptive attributes to facilitate font selection. Our focus is on shape-based letter legibility. In our work, we also rely on crowdsourcing for data collection. We ask pairwise comparison queries, rank representative inputs, and learn a prediction model, much like Zhu et al. [2014], with the goal of rating the legibility of new letters to drive their deformation.

CAPTCHA. The purpose of CAPTCHA, e.g., [Chellapilla et al. 2005], is to alter images which contain characters so that the characters are easily recognized by humans but hardly recognizable by a machine in an economic way. Letter shape deformations are involved, but the focus is more on manipulating image content around the letters which affects OCR. Similar to psychology research, CAPCHA works do not propose legibility measures.

2D deformation. Many 2D shape deformation algorithms have been proposed over the years, e.g., [Igarashi et al. 2005; Lipman et al. 2008; Weber et al. 2009; Jacobson et al. 2011], just to name a few. These methods largely focus on interactive, free-form deformation of a *single* 2D shape, in the context of shape editing, keyframe animation, or morphing. Our framework deforms multiple letters to fit them within a container. The problem setting also adds two new dimensions to the deformation problem. First, our deformation is controlled by letter legibility, which has a much more complex relation to shape geometry than those considered by common geometric transformations. Second, we need to deform multiple letters in the container in tandem, simultaneously optimizing their shapes under a multitude of constraints.

3 Overview

Our algorithm takes as input a lowercase text in the Latin alphabet and a 2D outline shape $\mathcal{S}$, and automatically produces a legible calligram, via a coarse-to-fine approach; see Figure 3.

Path generation. At the coarsest level of processing, our input text can be viewed as a horizontal rectangle which we wish to fit into our outline image with minimal deformation. The fit criterion simply refers to having sufficient space along the layout path to place the input word. So we map the horizontal middle axis of the rectangle to the skeleton of our outline. Effectively, this mapping takes the skeleton as an approximate text layout path.

Letter alignment. At the next level of our hierarchy, we need to position the letters along the path. From a pure legibility and aesthetics perspective, the optimal positioning is given by an arc-length spacing in which each letter is allocated equal-sized space. This naïve solution does not account for the needs of deforming individual letters to fit into large protrusions in the outline, such as animal legs. To balance readability and fit, we correspond protrusions on the input shape outline with protruding features, or simply, anchors, on the individual letters to roughly position the letters. We also adjust for evenly distributed letter size, consistent letter orientation, and minimal letter overlap, through a weighted optimization, while maintaining a smooth text flow.

Letter deformation. Given the approximate locations, scales and orientations of all the letters, the finest-level step of our method deforms the individual letters to best satisfy our fit criteria, while preserving their legibility. In developing a method that addresses this task, our key challenge is a lack of a well defined measure that can evaluate the impact of deformation on legibility. We address this challenge by developing a perception-based, viewer-validated legibility measure. To define this measure we use a range of features inspired by typography literature and learn their relative importance using crowdsourced input. We then propose a deformation scheme that employs this measure as a guide.

4 Outline image analysis

We analyze the input outline image to extract a text layout path and some protrusions to align the input letters to the outline.

Protrusion detection. Deep protrusions on the outline require special processing to fit without significantly affecting legibility. We decompose the outline image into several parts with the method in [Luo et al. 2015]. Among these parts, we choose deep and relatively narrow regions as protrusions, because shallow protrusions can be fitted by only slight letter deformation, while large ones can be fitted using multiple letters with the distortion evenly distributed between them. Given the area $Area_{\mathcal{S}}$ of the input shape, we expect the average letter size to be $Area_{\mathcal{S}}/N$, where N is the letter count. We thus filter out any protrusion whose area is larger than $2 \cdot Area_{\mathcal{S}}/N$. We similarly filter out small protrusions with areas less than $0.2 \cdot Area_{\mathcal{S}}/N$. For each protrusion, we fit a bounding box aligned with the corresponding skeletal branch, and filter out too shallow ones when $W/H > 3/4$, where H and W are the height and width of the box. We denote the boundary separating a protrusion from the main 'body' of the input shape as a *separator*.

Path computation. We compute a skeleton of our outline to approximate the text layout path, since it is long, of low curvature, and sufficiently distant from the contour to allow readably-sized text. We first use the thinning tool in [Kovesi 2006] to extract the shape skeleton. We take the longest branch which is not covered by a protrusion as the seed path and greedily grow it to include consecutive branches. A branch is added if the angle between it and the current path tangent directions at the intersection point is smaller than $5\pi/6$, and it is not covered by a protrusion. Figure 4 shows some sample skeletons and the paths we extract from them. The text goes from left to right (or top to bottom) along this path, which separates the outline image into *below* and *above* portions.

5 Letter alignment

We match letter anchors, i.e., protruding shape features on the input letters, to outline protrusions detected in the previous section. A rough text layout is computed from this correspondence, which is further refined to minimize letter distortion during deformation.

Traditional typography provides an extensive classification of the different strokes that constitute a letter. We simplify this classification and abuse the terminology a bit here to better fit to our ap-

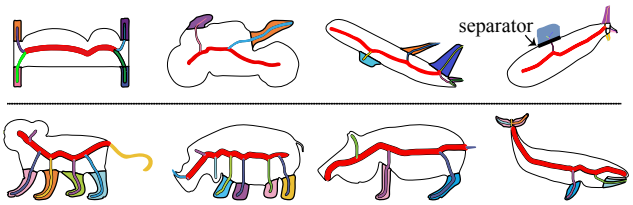

Figure 4: *Detected protrusions (with colors different from the main body) and layout paths (red) on a few shapes.*

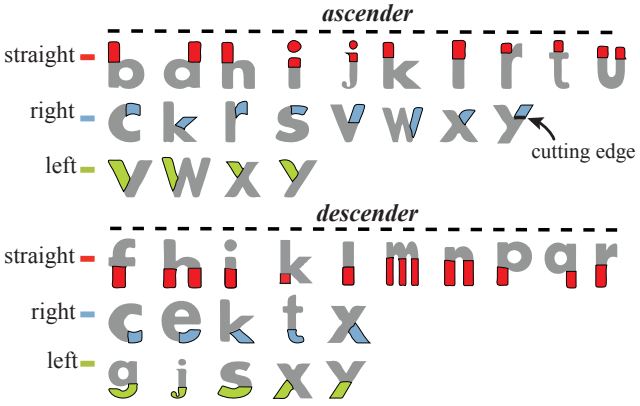

Figure 5: *Manually defined letter anchors on the lowercase Latin letters. The anchors are highlighted with different colors marked below their category labels.*

plication. We define protruding features on letters as *anchors* and classify letter anchors as *ascenders* and *descenders* depending on their relative positions on the letter. As shown in Figure 5, *ascenders* and *descenders* extend upward or downward of a letter.

We manually specify the extent of each identified anchor, placing the *cutting line* (see Figure 5) so as to separate it from the rest of the letter. We further introduce an orientation attribute for each letter anchor, which can be "straight", "left", or "right". Figure 5 shows the anchors, with marked orientation attributes, of the lowercase Latin letters. Three letters, 'a', 'o', and 'z', without anchors are not included in this figure. Throughout the paper we use the "Futura Bold" font to facilitate deformation and other computations, though other fonts might be used too. While manual, this analysis is required only once per alphabet per font.

5.1 Anchor-Protrusion Correspondence

Our algorithm matches identified outline protrusions with letter anchors in the input text. Intuitively, we would like to compute a corresponding anchor for each protrusion that satisfies two goals. On the one hand, we wish to minimize the local letter deformation necessary to fit a letter's anchor to the protrusion. On the other hand, we also wish to minimally change the letter spacing along the path. In particular, we seek matches that minimally shift letter locations from a default even letter spacing, and forbid matches that switch the letter order. We formulate this as an assignment cost with a set of constraints. The cost of matching a letter anchor to an outline protrusion is dependent on two functions: the relative positional compatibility, and the type and orientation compatibility.

Positional compatibility. Relative positional compatibility between a protrusion $\mathbf{a}_i$ and a letter anchor $\mathbf{s}_j$ is defined as a function

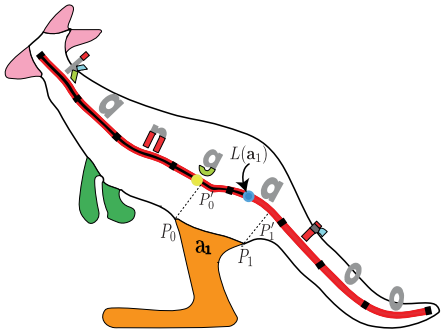

Figure 6: *Alignment of letters in the outline image. The ideal letter position is computed by uniformly segmenting the layout path. For example, the position of the letter 'g' is marked by a yellow dot. The approximate position of a protrusion,* e.g., *the kangaroo's leg* $\boldsymbol{a}_1$*, is computed by projecting the end points* (i.e., P_0 *and* P_1*) of its* separator *to the path. Its position* $L(\boldsymbol{a}_1)$ *is set at the middle of the two projections* P_0' *and* P_1'*.*

of the letter's optimal position $L(\mathbf{s}_j)$ and the approximate location of the protrusion along the path $L(\mathbf{a}_i)$. Suppose the input word has N letters and suppose the letter anchor $\mathbf{s}_j$ is on the j-th letter. The optimal position $L(\mathbf{s}_j)$ is the position of the j-th letter using arc-length parameterization of the path. We divide the path into N equal segments, and specify the optimal position of the jth letter as the midpoint of the jth segment. For example, the position of the letter 'g' is marked by a yellow dot in Figure 6 on the skeleton. For each outline protrusion, we compute its approximate location along the path by first projecting the end points of its *separator* to the closest locations on the path, and then using the mid-point of the two projections as its approximate location (see $L(\mathbf{a}_i)$ in Figure 6). The positional compatibility of the assignment is then given by the Gaussian function

$$P_{loc}(\mathbf{a}_i, \mathbf{s}_j) = \exp\left(-(\frac{L(\mathbf{a}_i) - L(\mathbf{s}_j)}{2 \cdot W/N})^2\right), \qquad (1)$$

where W is the length of the text path.

Type and orientation compatibility. We evaluate the type and orientation compatibility of an outline protrusion and letter anchor with a discrete function P_{sim}. We classify each protrusion as ascending or descending based on whether it is above or below the layout path. We use PCA to compute the major axis of each protrusion, and classify its orientation as "right", "straight", or "left", based on the angle θ between the major axis and the text path direction. The orientation is set to "right", "straight", or "left", if $\theta <= \frac{\pi}{3}$, $\frac{\pi}{3} < \theta <= \frac{2\pi}{3}$, or $\frac{2\pi}{3} < \theta < \pi$, respectively. If the anchor and the protrusion are on different sides (*e.g.*, an ascending anchor and a descending protrusion), we set $P_{sim} = 0$. If the two are on the same side, we evaluate their orientation compatibility. If the orientations match (*e.g.*, right and right), we set $P_{sim} = 1$. If the mismatch is minor (*i.e.*, one of the two orientations is "straight") we set $P_{sim} = 0.75$; and if it is major, *i.e.*, one is right and one is left, then we set $P_{sim} = 0.5$.

Final cost and non-inversion constraints. The overall compatibility cost for matching a protrusion $\mathbf{a}_i$ and a letter anchor $\mathbf{s}_j$ is:

$$c(\mathbf{a}_i, \mathbf{s}_j) = 1 - P_{loc}(\mathbf{a}_i, \mathbf{s}_j) \cdot P_{sim}(\mathbf{a}_i, \mathbf{s}_j). \qquad (2)$$

Given a set of image protrusions $\mathbf{A} = \{\mathbf{a}_i\}(i = 1, ..., M)$ and a set of letter anchors $\mathbf{S} = \{\mathbf{s}_j\}(j = 1, ..., N_s)$, we find the best corre-

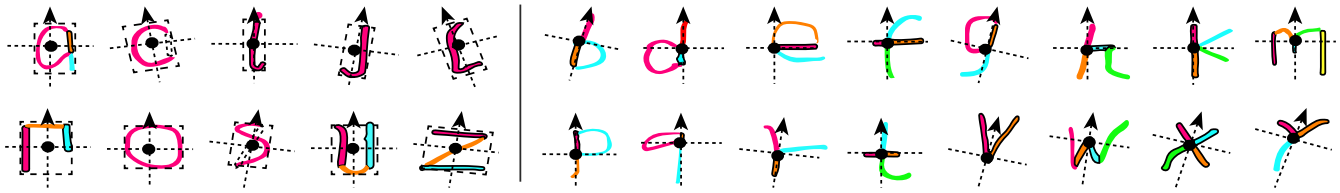

Figure 7: *Stroke decompositions and local coordinate frames for all letters. The origin of the local coordinate frame is marked by a black dot, and is either located at the center of the minimum bounding rectangle (MBR) (left) or at the center junction of strokes (right). The y-axis is chosen as an edge of the MBR (left) or according to some reference strokes (right). The reference strokes are marked with a black outline.*

sponding anchor for each protrusion by minimizing the combined assignment cost, subject to non-inversion constraints,

$$c(\mathbf{A},\mathbf{S}) = \sum_{i=1}^{M} c(\mathbf{a}_i, \mathbb{M}(\mathbf{a}_i)), \tag{3}$$

$$s.t. \quad L(\mathbb{M}(\mathbf{a}_i)) < L(\mathbb{M}(\mathbf{a}_j)), \text{ if } \quad L(\mathbf{a}_i) < L(\mathbf{a}_j). \tag{4}$$

Here, $\mathbb{M}(\mathbf{a}_i)$ is the corresponding letter anchor for the protrusion $\mathbf{a}_i$. $\mathbb{M}(\mathbf{a}_i)$ could be a void anchor to allow protrusions without corresponding anchor. We define the compatibility $P_{loc}(\cdot,\cdot), P_{sim}(\cdot,\cdot)$ between a void anchor and a protrusion to be 0.

We could use a Markov Chain optimization to obtain the best solution. However, our search space is typically small, with less than a dozen protrusions and less than two dozen anchors, so that an exhaustive search which quickly discards invalid and poor solutions is equally effective. Specifically, for each protrusion $\mathbf{a}_i$, we first limit its correspondence to the top three anchors with largest location compatibility score $P_{loc}(\cdot,\cdot)$ and exclude anchors whose location compatibility score is smaller than 0.1. After that, there is typically a few thousand different combinations to be evaluated.

5.2 Initial alignment

After finding correspondences between letter anchors and protrusions, we use their matches to align the letters to the outline image. To minimize letter overlaps in this initial layout, we set the initial scale of each letter to a portion of its expected final size. Specifically, we compute per-letter scales such that the area of the scaled letter equals to $Area_{\mathcal{S}}/2N$. We first position letters whose anchors have corresponding protrusions and then position the rest, keeping them as evenly spaced as possible. For each letter with an anchor that corresponds to a protrusion, we place the letter by aligning the centers and orientations of the anchor's cutting line (Figure 5) with the protrusion's separator (Figure 4). The orientation and location of letters with more than one corresponding protrusions are decided by the largest one.

After fixing letters with corresponding protrusions, we insert the remaining letters in-between them. Suppose n consecutive letters need to be inserted between two fixed letters. We divide the path between the two fixed letters into n intervals, and put one letter at each interval. The orientations of these letters are then set to match the normal of the path (see *e.g.* the letter 'u' in Figure 3).

5.3 Refined alignment

The coarse layout from the initial alignment is readable and fairly evenly spaced, but the letter boundaries remain relatively far from the image outline. We improve the fit by adjusting the position, orientation, and scale of the letters. This step produces a good starting point for the deformation in Section 7, which is controlled by a full legibility measure learned from crowdsourced data.

This refinement is driven by three key criteria. First, we aim to compute the per-letter transformations that optimize fit, or outline alignment. Second, while we allow letters to scale along both axes, to maintain readability, we minimize changes in their aspect ratios. Finally, to maintain both readability and aesthetics, we seek smooth letter flow. That is, we want the letters to have similar overall scale and orientation, with minimal overlap.

We formulate these criteria as follows. We encode the preference for preserving letter aspect ratio as:

$$score_l = \sum_{i=1}^{N} \left| \ln \frac{r_d(l_i)}{r_o(l_i)} \right| / N, \tag{5}$$

where $r_o(l_i)$ and $r_d(l_i)$ are the aspect ratios of a letter l_i before and after refinement respectively.

The fit criterion is encoded as:

$$score_g = \left| \ln \frac{Area_{\mathcal{S}}}{\overline{Area}_{(L,\mathcal{S})}} \right|, \tag{6}$$

where $Area_{\mathcal{S}}$ is the area of the input outline image $\mathcal{S}$ and $\overline{Area}_{(L,\mathcal{S})}$ is defined as the difference between two areas, *i.e.*, $\overline{Area}_{(L,\mathcal{S})} = B - A$. Here, B is the area of the intersection of letters L and the outline $\mathcal{S}$, and A is the area of the outline $\mathcal{S}$ uncovered by the letters L. We define $\overline{Area}_{(L,\mathcal{S})}$ in this way to maximize the intersection of L and $\mathcal{S}$, and at the same time to minimize the uncovered region of $\mathcal{S}$.

The smooth flow is encoded as:

$$score_w = \omega_z Var_s(L) + \omega_o Var_o(L) + \omega_b \Sigma_i \frac{Overlap_{(l_i,l_{i+1})}}{Area_{\mathcal{S}}}. \tag{7}$$

Here, $Var_s(L)$ denotes the variation of the areas of all letters in a word L. When evaluating this variation term, each letter $l_i \in L$ is normalized using the area $Area_{\mathcal{S}}/N$ to make the result independent of the size of the input outline image. $Var_o(L)$ is the variation of the orientations (in radians) of all letters in L. $Overlap_{(l_i,l_{i+1})}$ denotes the overlapping area of two neighboring letters. The combination weights ω_z, ω_o, and ω_b are all fixed at 1.

The combined layout energy is

$$S = \lambda score_l + score_g + \gamma score_w. \tag{8}$$

We empirically set $\lambda = 0.4$, and $\gamma = 0.6$ in all experiments, prioritizing fit and flow over aspect ratio preservation.

This function has no continuous derivatives, and thus requires a discrete solver to optimize. We use hill-climbing [Russell and Norvig 2010] to find the location, orientation, and vertical and horizontal scales of each letter that minimize this function. This iterative optimization stops when the change in energy function is smaller than a prefixed threshold ($\varepsilon = 0.001$), for two consecutive iterations.

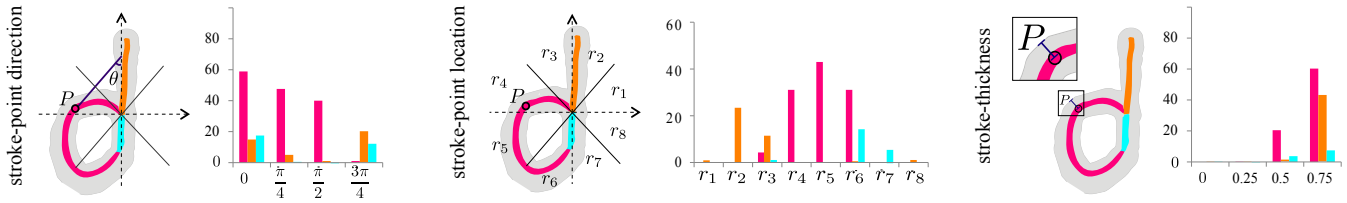

Figure 8: *Stroke features. From left to right, they are the histogram features of stroke orientation, position, and thickness. Histograms bins are colored according to the color of the corresponding stroke in the letter 'd'.*

6 Letter legibility

A key to the final deformation step, which adjusts the letter shapes to best fit the image outline, is the ability to evaluate the impact of these shape changes on letter legibility. There are no existing methods that can assist us with this measurement. Moreover, due to the structural variance between the different letters, it is not clear if one can develop a unified legibility function that works for all letters. For example, simple letters such as 'o' appear less sensitive to deformation, while pairs of easier to confuse letters, such as 'h' and 'n', seem to be more sensitive to edits. We therefore choose to learn an individual legibility measure for each letter in the alphabet by the relative attribute ranking method [Parikh and Grauman 2011] with a large-margin approach [Joachims 2002].

Feature space. The first question to address is how to represent the deformed letters for our legibility measurement. We need an effective representation that can be generalized to work for all Latin letters, and has a moderate dimensionality since high dimensional features usually require more training data [Jin and Wang 2012]. We encode letters using the popular, see e.g. [Phan et al. 2015], skeleton-plus-thickness-profile representation. Specifically, we separate each letter into its strokes and define a feature vector for each stroke, representing its *skeleton* and *thickness profile*. We describe each letter by concatenating the features from all strokes. For example, the feature representation of letter 'd' is shown in Figure 8, where the skeletons of different strokes are color-coded.

Feature extraction. For each letter, we set up a local coordinate system to evaluate its features based on skeleton length, direction, position, and stroke thickness. These coordinate systems are chosen to make the extracted features invariant to translation, scale, and rotation of letters. Generally, the origin is at the center of a letter and strokes are uniformly distributed in different quadrants. In the end, we design these coordinate systems for different letters as shown in Figure 7, while different choices might work equally well. The origins are either at the center of a letter's bounding box, or at a stroke junction nearest to the letter center. The y-axis is then set to an axis of the bounding rectangle, or with regards to some reference strokes, *e.g.*, perpendicular to a reference stroke, or bisecting two reference strokes. Please refer to Figure 7 for the local coordinates of different letters.

We define length, direction, position, and thickness features for each stroke. The length feature is the stroke's skeleton length normalized by the total length of all strokes. The direction feature is an orientation histogram. Specifically, we take each pixel on a stroke as a sample point and compute the angle between the y-axis and the skeleton's tangent direction at each sample point. This angle takes on a value between $[0, \pi]$ and we quantize it to four equal sized bins, *i.e.*, $[0, \pi/8] \cup [7\pi/8, \pi]$, $[\pi/8, 3\pi/8]$, $[3\pi/8, 5\pi/8]$, and $[5\pi/8, 7\pi/8]$, to build a direction histogram. This histogram is further normalized by the total number of points as shown in the left of Figure 8. The position feature is also a histogram. Specifically, we divide the local coordinate plane into eight octaves, count the number of skeleton points in each octave, and normalize this histogram by the total number of points, as shown in the middle of Figure 8. The thickness of a stroke is also encoded as a histogram. We compute the minimum distance between each stroke sample point to the stroke's boundary. This distance is then normalized by the minimum of the letter's skeleton width and height, and quantized into four equal sized bins between $[0, 1]$ to define the thickness feature as shown in the right of Figure 8.

Finally, a stroke's feature vector consists of:

- The relative *length*, represented by a scalar.
- The *orientation histogram*, represented by a 4D vector.
- The *position histogram*, represented by an 8D vector.
- The relative *thickness histogram*, represented by a 4D vector.

Therefore, a stroke is described by a 17D vector. We concatenate the feature vectors computed for all the strokes to form the feature of each letter. Taking a letter with three strokes as an example, its feature is a 51D vector.

Training objective. For each letter l, we learn a ranking function r_l. We first collect a set of training images $\mathbf{I}_l$ containing different deformed variations of the same letter. We then obtain two sets of crowdsourced pairwise visual comparisons on those deformed letters. Pairwise comparison is used, since it is much harder to mark absolute metric scores [Parikh and Grauman 2011].

The first set $\mathbf{O}_l = (i, j)$ contains ranked pairs for which the first image i is more legible than the second image j. The second set $\mathbf{S}_l = (i, j)$ consists of unordered pairs for which both images have the same legibility. We wish to learn a ranking function

$$r_l(\mathbf{x}) = \mathbf{w}_l^T \mathbf{x} \tag{9}$$

with the feature vector $\mathbf{x}$ satisfying the training constraints:

$$\forall (i, j) \in \mathbf{O}_l : \mathbf{w}_l^T \mathbf{x}_i > \mathbf{w}_l^T \mathbf{x}_j$$
$$\forall (i, j) \in \mathbf{S}_l : \mathbf{w}_l^T \mathbf{x}_i = \mathbf{w}_l^T \mathbf{x}_j.$$

We directly solve the optimization by the large-margin approach [Parikh and Grauman 2011], obtaining a score in $[-1, 1]$ for each deformed letter with higher scores implying higher legibility.

Warped letter generation. We collect more than 30,000 deformed shapes for all the 26 lowercase Latin letters. There are more than 1,500 shapes for most of the letters. Less shapes are used for simple letters, such as 'i' and 'l'. The various deformed letters are generated by either stroke thickness transformation or projective transformation. Taking the letter 'd' for example, we generate in total 1,535 shapes, where 80% of them come from stroke thickness transformation (according to 307 human sketched letter skeletons), and 20% of them come from projective transformation

Figure 9: *Deformed letter generation (to train the legibility measure). The first row shows letters generated by applying randomly sampled thickness profiles to a human sketched skeleton. The second row are letters generated by applying random projective transformations to the standard letter.*

of the standard form, see Figure 9. Stroke thickness transformation varies the thickness of each stroke in a human sketched skeleton. We concatenate points of all strokes into an array $p_i, i = 1, ..., n$. New letter shapes are generated by the union of disks with different radiuses centered at these points. The radiuses are controlled by a cubic spline curve defined in the range $[1, n]$. The spline curve interpolates n_s thicknesses uniformly distributed in $[1, n]$ and $n_s \in \{4, 6, 8, 10\}$. The thickness of each data point is specified by a base thickness plus a random variation in the range of minus to plus half of the base thickness. The base thickness is also determined randomly, in $[0.8, 1.5]$ of the default stroke thickness.

Once the data from the thickness transformation is ready, we add the remaining 20% of the data from random projective transformations of a standard letter. For each standard letter, we randomly perturb the four corners of its bounding box. The perturbed position of each corner is confined to a disk centered at its original location and with a radius of 0.4 times the minimum of the letter's width and height. The four perturbed points define a projective transformation which warps the original letter. Finally, 40% of the dataset is selected for training and 60% for testing.

Crowdsourced data. We collect crowdsourced pairwise legibility comparisons of deformed letters via the Amazon Mechanical Turk service. At each time, a participant is shown two deformed letters side by side. The participant is asked to choose if the left or right letter on display is more legible or if both have similar legibility. The webpage we used for data collection is included in the supplementary material. About 200 random pairs were generated for each letter, and each pair was compared by 5 participants. Less queries are generated for simple letters such as 'i' and 'o', and more for complex ones such as 'b' and 'k'. In total, $20,000$ (200 pairs $\times$ 20 letters $\times$ 5 comparisons) pairwise legibility comparisons are collected for all the 26 letters from 90+ participants. The comparison tasks are divided into 20 batches, such that each batch can be finished by a participant in about half an hour.

As suggested by [Joachims 2002; Liang and Grauman 2014], we validate our learned legibility measure using prediction accuracy and Kendall's τ index [Kendall 1938]. The average accuracy and τ for all the letters are about 0.8 and 0.6 respectively. These values are not very high since our labeled data is noisy with many inconsistencies. Hence, we further empirically validate the learned legibility measure using the test data in Figure 10, which shows letter ranking consistent with human perceived legibility.

7 Legibility-preserving letter deformation

The final step of our calligram computation deforms the individual letters to best satisfy fit, while minimally affecting legibility. Our

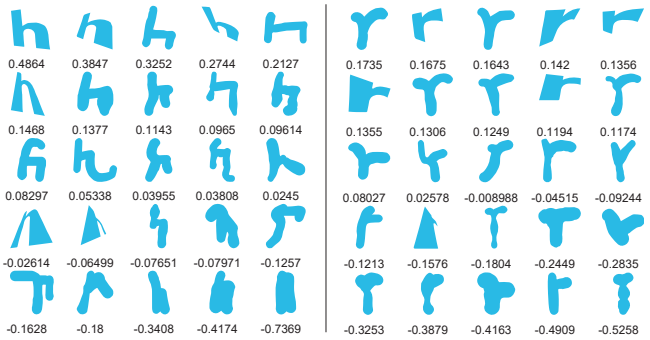

Figure 10: *Some deformed letters of 'h' and 'r', ranked according to our legibility measure.*

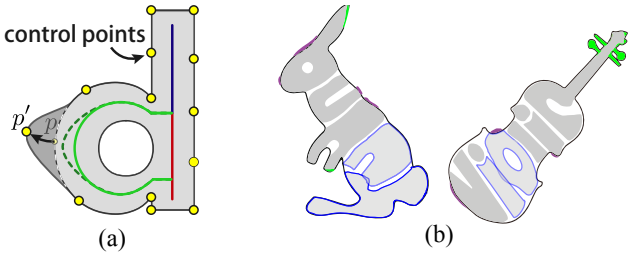

Figure 11: *Deformation of a letter boundary (a) and post-processing after letter deformations (b).*

optimization uses the same criteria as the refined letter alignment, balancing legibility, fit, and textual flow. We therefore use the same energy function (Equation 8) but replace the coarse, aspect ratio based legibility measure $score_l$ with a more sophisticated alternative $\hat{score}_l$ that accounts for local deformations,

$$\hat{score}_l = (1 - \sum_{i=1}^{N} (r(l_i) + 1)/2N), \tag{10}$$

where $r(l_i)$ is the ranking score for letter l_i.

Letter boundary discretization. To allow fine-scale deformation of our letters, we represent the boundary of each letter using Catmull-Rom splines [DeRose and Barsky 1988] (see Figure 11 (a)), and use their control points to deform the letter boundary. To determine the control point density, we use a distance parameter μ which is progressively refined for fine-tuning. Specifically, we use $\boldsymbol{\mu} = [10, 5, 3]$ pixels at different iterations. The initial set of control points includes the corner points of the letter strokes, detected using the feature detection method of [Harris and Stephens 1988]. We iteratively insert a new control point at the middle of two consecutive control points whose distance is larger than 2μ. This insertion of control points is applied after each iteration of deformation.

Legibility-driven deformation. We deform the boundary control points to minimize the energy function Equation (8) with the learned letter legibility measure Equation (10). Suppose there are m control points along the letter contour. At each iteration, we generate $2m$ samples by randomly moving each control point inward or outward along its normal direction, with the step size bounded by our distance parameter μ. If a control point is within distance μ from the input outline, we consider fit as fully satisfied locally and prevent the point from moving further. An example of the process is shown in Figure 11 (a), where a deformed letter is generated by moving the control point p along the normal of the boundary. Some

sampled deformations can lead to self-intersections of letter boundaries. We discard those samples and pick the optimal solution that minimizes Equation (8).

Evaluating the legibility measure in Equation (10) requires tracking the motion of the letters' skeleton. We compute the initial skeleton using the skeletonization method of [Kovesi 2006]. We use Green coordinates [Lipman et al. 2008] to track the motion of the skeleton during deformation. We compute the feature vector $\mathbf{x}(l_i)$ as described in Section 6 and obtain the letter legibility score with Equation (9) and Equation (10).

For each step size μ, we repeat this iteration until $\|S^n - S^{n+1}\| < \epsilon$, where $\epsilon = 0.001$ in our implementation. Once the iteration converges, we reduce the step size μ and refine the deformation.

While the deformation control mechanisms adopted at the training and legibility-driven calligram construction stages are different, we generally obtain similar letter shapes. During calligram generation, a more refined deformation, by manipulating control points, is necessary for the fine-grained inter-letter constraints. During training, we aim to sparsely cover the space of letter deformations, where a coarse deformation mechanism is better suited. Fine-level deformation would require many more samples to obtain coverage.

Post-processing. Up to this point, the deformed letters still may not fit the image outline perfectly, as shown in Figure 11(b), we perform post-processing to address the following three issues. First, part of the letters may protrude outside the outline; see purple regions in Figure 11(b). We simply remove these protruding regions. Secondly, the deformed letter might not reach the outline precisely, (green gaps in Figure 11 (b)). We simply add those green regions to the corresponding letters, if the green gap is between the outline contour and some *critical edges* of a letter. The critical edges are those within a five-pixel distance from the convex hull of the deformed letter. Finally, two letters may be too close to (even overlapping with) each other (blue overlaps in Figure 11 (b)). We define a minimal gap – six pixels in our implementation, between letters. Once overlapping letters are detected, we shrink both letters' boundaries by half of the minimal gap.

8 Results and evaluation

We have tested our method on a range of input shapes, including organic ones and those of man-made artifacts. All the results conform to our aesthetics and legibility considerations and they have been obtained all under the *same* default parameter setting (Figures 1 and 12-17) unless otherwise specified. We compare our method to calligraphic packing and to manual designs in terms of the legibility and overall quality of the generated calligrams. We also evaluate our objective function and legibility scoring.

Parameters. Our method has five tunable parameters: weight λ for letter legibility, weight γ for word legibility, and with respect to word legibility, the weights w_z, w_o, and w_b, which influence letter scale, orientation, and interrelations, respectively. All the calligram results shown in the paper have been obtained fully automatically, under the default parameter setting: $\lambda = 0.4$, $\gamma = 0.6$, and $w_z = w_o = w_b = 1.0$, unless stated otherwise.

Word animals. Word animal calligrams designed beautifully by professional artist Dan Fleming provided the initial motivation to our work. We have tested our method on all the word animal shapes we could find. Figure 12 shows, side-by-side, some calligrams created by the artist and by our method, whose input shape is the one which tightly bounds the artist's design. As one can observe, while in some cases, automatically generated results still lack the elegance originating from an artist's touch, by and large, the overall quality of the results are visually quite close and essentially indistinguishable in some cases, e.g., the pig, bunny, and camel.

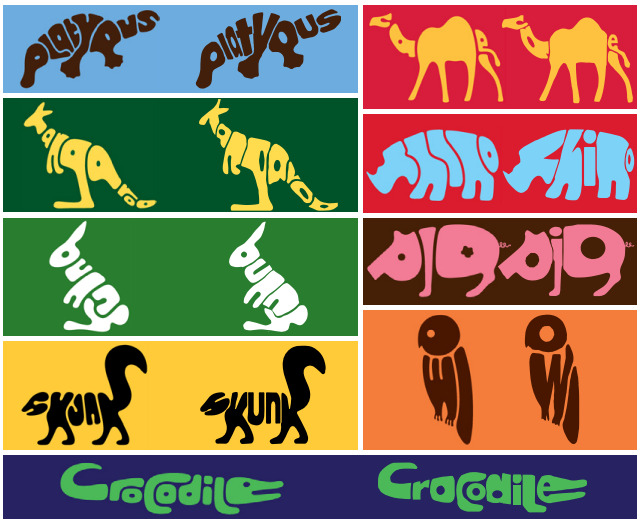

Figure 12: *A gallery of word animals generated by our automatic algorithm (right image in each pair), compared to designs from Dan Fleming (images taken with permission). In most cases, the overall quality of the results are quite close.*

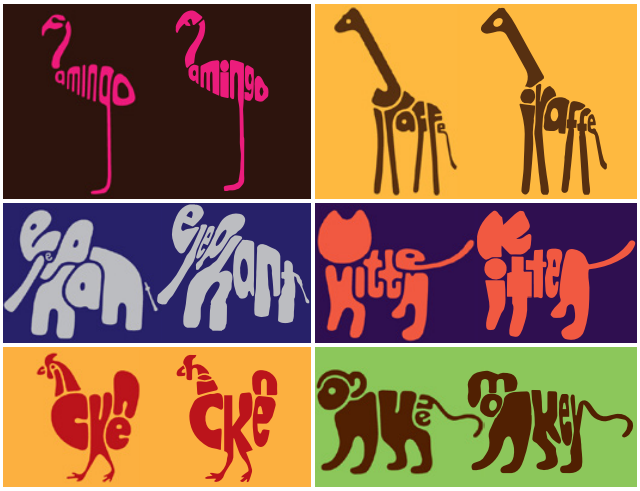

Figure 13: *Additional examples comparing algorithmic generation with expert design, showing more noticeable discrepancies. Yet, our results are still seen as providing reasonable alternatives.*

Expert designs shown in Figure 13 exhibit more significant letter deformations and arrangement variations. Extreme examples include the elephant's eye using the letter 'e', which is significantly smaller than the other letters, the kitten's head given entirely by the diacritic of the letter 'i', and the monkey skull by a drastically deformed 'm'. Our current method does not attain such solutions and its results deviate more noticeably from professional designs; this is dictated by the criteria set for low-curvature text flow, as well as letter and word legibility. The leg of the flamingo was not assigned to 'g' but to 'n' since our current alignment scheme slightly favors mapping the long leg to a stem in 'n' than to a descending anchor in 'g'. Nevertheless, the automatically generated calligrams are generally seen as offering reasonable alternatives.

Figure 14: *A gallery of compact calligrams obtained by our method on input shapes obtained via Google image search.*

On "wild" inputs. Results for word animals have all been obtained on input shapes converted from expert designs. Figure 14 shows a set of calligrams generated by our method on input shapes obtained "from the wild" via Google image search. Most of them represent man-made artifacts and have not been associated with calligram designs before. To assess the robustness and generality of our method, we embedded words of varying lengths, from "bat" to "batmanforever", to the same image. Another stress test is on the embedding of a very long "word" (bottom of Figure 14).

Legibility test. The validation presented in Section 6 serves to assess the learned legibility measure for individual letters. In our first user study, we evaluate how consistent our combined (word and letter) legibility score, as defined by $\lambda score_l + \gamma score_w$ in Equation (8), is with respect to human judgment.

The study consists of 20 pairs of calligrams: each pair embeds the same word, but the word is not spelt out. For each pair, a participant was asked to choose which one he/she thinks is more legible or the assessment that the two calligrams are equally legible. The 20 word-shape combinations for the study were randomly chosen from the set of all available examples. The pair of calligrams were chosen at random stages over our optimization process, ensuring that the gap between their legibility scores varies.

In the end, 30 participants provided feedback. These participants are either graduate students from computer science and engineering or professionals with similar backgrounds. Among a total of 593 user responses (one participant did not finish), **70.1%** agree with legibility rankings which would result from our score. Data, user instructions, and further details about *all* the user studies can be found in the supplementary material.

Overall quality test. The second user study involves ranking the overall quality of word calligrams and serves three purposes: a) to compare our method with calligraphic packing; b) to compare our method with designs by a professional artist; c) to evaluate our objective function (8). The study consists of a total of 25 pairs of calligrams: each pair embeds the same word, but the word is not spelt out. For each pair, a participant was asked to choose which one he/she thinks is of higher overall quality or the assessment that the two calligrams are of equal quality. The participants were told to judge the quality based on how well the calligram conveys the input shape, along with legibility and aesthetics. The same 30 participants from the first user study provided feedback.

Four of the 25 pairs compare our results to those produced by calligraphic packing; outcome related to these examples is reported later. The 9 pairs of algorithmic and artist-designed calligrams shown in Figure 12 are also included in the study. Among a total of 270 responses, **50.74%** favored our method over artist's designs or indicated that they are of equal quality (9.56%).

Each of the remaining 12 pairs is composed of two algorithmically generated results. The input word-shape combination was again randomly chosen from the set of examples and the two results were chosen at random stages over our optimization process. These 12 pairs serve to assess the consistency between our objective function and human judgment when ranking quality of calligrams. Among a total of 360 responses, **57.8%** are consistent with the ranking based on our objective function.

Participants' feedback revealed that generally, it was not easy to judge the overall quality of the calligrams due to the combination of factors. Personal and stylistic preferences were clearly influential. Some user judgments were affected by certain local elements that were thought of as either aesthetic or unpleasant.

Algorithmic vs. manual. In the final user study, we ask viewers to rank the overall quality of calligrams created by our method and by humans, including Dan Fleming. To collect results from human creators, who had not participated in any of our user studies, we provided instructions on what legible compact calligrams are and showed them two examples designed by Dan Fleming. The human creators who participated in our study are all graduate students in computer science or engineering or professionals with postgraduate degrees in these disciplines. About half of them claim specialization in fields related to visual computing and about half possess varying degrees of design experience. They were asked to design calligrams on paper, taking as much time as they wish until they were satisfied. Their creations on paper were then digitized. We collected more such creations than we needed and chose the ones which were clearly of higher quality for the study. All the human creations can be accessed in the supplementary material.

The input shapes tested and the queries belong to two groups:

- The first group consists of five word animals: skunk, kangaroo, chicken, elephant, and kitten. Note that the last three (Figure 13) are challenging cases. Each query set is formed by four calligrams (in random order): two created by humans, one by our method, and one by Fleming. The viewers were asked to rank the calligrams based on the overall quality.

- The five test cases for the second group are shown in Figure 15. Note that the sedan, joypad, and submarine have relatively simple boundaries; mermaid and motorbike are more complex. Each query set in this group consists of three calligrams (in random order): two were created by humans and one by our method. Note that we have no access to professional designs for these inputs. The viewers were asked to rank all three calligrams based on their overall quality.

On average, a human creator took **16.55** minutes to complete a calligram from the two groups explained above.

A separate group of viewers, the 30 who provided feedback for

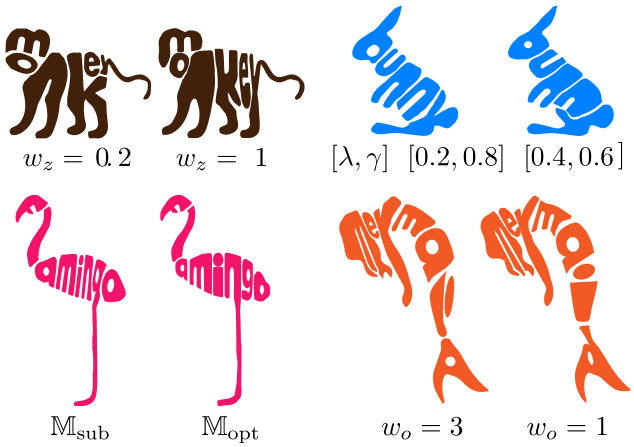

Figure 16: *Result variation for calligram generation. Flamingo: alternative letter alignments, where $\mathbb{M}_{opt}$ and $\mathbb{M}_{sub}$ indicate the optimal and one close sub-optimal matching correspondences, respectively. Bunny: influences of weight γ on word legibility. Monkey: influences of weight w_z on even letter scales. Mermaid: influences of weight w_o on consistency of letter orientations.*

Figure 17: *Our results (in color) vs. calligraphic packing results (black) generated with initial user placement and orientation of letters, on four examples taken from their original paper. The Mona Lisa and "successful" examples each has two separate layout paths, requiring user strokes as initialization. Our results were otherwise generated automatically with default parameters.*

the first two studies, were asked to rank the two query sets of calligrams. Among the 150 ranking responses for the word animal group, 54% ranked the Fleming design first and among these responses, 37.3% ranked our method in second place. In 21.3% of the time, our method was ranked first. Overall, **41.44%** responses ranked our method ahead of *all* human creators. On the other hand, we also observe some noise in the participant's feedback, as about 25% responses did not rank the professional designs first.

Among the 150 responses for the second group, 34.66% ranked our method in first place and 44% placed it second. It is interesting to note that for the more complex examples, the mermaid and motorbike, our method received the most first-place votes. It did not fare as well, in users' judgment, on the simpler inputs: joypad, sedan, and submarine. The human creation for the submarine, with the overlapping of 'b' and 'm', as shown in Figure 15, was well-liked, receiving 24 first-place votes out of 30. It would appear that local features can play a big role in user judgment.

Result variations. One option to vary the final result is to consider multiple letter alignments, as no alignment objectives can work perfectly in all cases. As shown in Figure 16, adopting a sub-optimal alignment, which is accessible during our search, allows the leg of the flamingo to be assigned to the letter 'g'. Interesting variations can also be obtained by tuning parameters. De-emphasizing letter legibility by reducing λ from 0.4 to 0.2 places more weight on word legibility. As a result, letters of the bunny exhibit a more even distribution, in terms of the scales of the two 'n' letters and less overlap between 'n' and 'y'. While holding other parameters at default, decreasing w_z from 1.0 to 0.2 tolerates more variations in the scales of the letters, as can be observed for the monkey calligram. In another example, increasing w_o from 1.0 to 3.0 emphasizes more the consistency in letter orientation, leading to a different result for the mermaid, particularly for the letters 'i' and 'd'.

Multi-line calligrams with user assistance. Our set goal in this work is to develop a fully automatic method. However, few user-drawn strokes as *suggestive* layout paths can allow us to produce *multi-line* calligrams such as the Mona Lisa and "successful" results in Figure 17. Note that a user only needs to provide the quick strokes, splitting of the input text is done automatically by the algorithm. If the letters are rather scrambled, as for the orangutan in Figure 2, user-assisted letter placement may be necessary.

Comparison to calligraphic packing. Figure 17 compares our results to user-assisted outputs from calligraphic packing [Xu and Kaplan 2007]. The four input shapes and the calligraphic packing results were taken directly from their paper. Only these inputs would allow the two methods to be comparable; all the other examples from their paper scrambled the ordering of letters. For the Mona Lisa and "successful", as the calligrams are necessarily multi-line, user strokes were provided, after which our method ran fully automatically with default parameters. From the user study on calligram quality, when viewers were shown side-by-side results from Figure 17 (in random order and the same color scheme), out of a total of 120 answers, **70%** indicated preference for our method, while 5 responses could not distinguish. The majority of votes that went to their method are for the elephant and the Mona Lisa.

Timing. Our current implementation is entirely MATLAB-based. All steps of the method, except for the final deformation step, take less than 30 seconds to execute. The iterative deformation is the most time-consuming, typically requiring hundreds of iterations. With the un-optimized MATLAB code executed on a single-core PC with Inter(R) Dual Core(TM) i5 CPU 540@2.53GHz, it takes between 10-17 minutes to obtain all results shown in the paper. Average run time for the 10 examples in the final user study was 11.8 minutes, while human creators took 16.55 minutes. With a C/C++ implementation, a significant speedup is expected [Andrews 2012; Aruoba and Fernandez-Villaverde 2014].

9 Discussion, limitation, and future work

We develop a method for automatic generation of compact and legible calligrams. Extensive tests demonstrate the effectiveness and robustness of our method. Preliminary studies, based on viewer feedback, indicate that in terms of overall quality, the calligrams

Figure 15: *Several calligrams designed by human creators (black and white), contrasted with results by our automatic method (colored). Human creations shown here were the selected best ones.*

generated by our fully automatic method are comparable to those produced by humans. Note that the human creators in our study generally possess above-average artistic skills and we selected the best of their creations when comparing to our results.

Even though we were seeking a fully automatic method originally, our current pipeline can easily incorporate user assistance, e.g., stroke hints for layout paths, as shown in Figure 17. Our ultimate goal is certainly not to replace or supercede professional artists for calligram design, a highly creative and artistic endeavor. The best way to utilize our method is for it to automatically generate design suggestions which may inspire an artist or novice user.

"Calligramification". Not all shapes of a monkey can result in an elegant word animal; probably most monkey poses cannot. From a calligram *design* point of view, the major limitation to our current solution arises from the fixed input shape. A professional calligram design typically involves the design of the boundary shape as well. An interesting extension to our method would be to allow the input shape to deform to improve the quality of the generated calligram. That is, the input shape is "calligramified". To achieve this, one needs to constrain the shape deformation to ensure its semantic validity while striking a balance with letter shape deformation.

Potential use of legibility. Data and results from our letter legibility study may serve other applications beyond calligram generation. In connection with the recent work by Campbell and Kautz [2014], a legibility measure may be factored into the generative model they have developed for fonts. In the work by Zitnick [2013], a user's handwriting can be beautified in real time by applying an averaging operation to letter instances appearing in other sample writings by the same user. To improve the legibility of a piece of handwriting, without other writing samples, a model of letter legibility would be necessary.

Co-constrained deformation. In addition to contributions by the legibility study, we believe that the notion of "co-constrained deformation" arising from our letter deformation problem is of interest beyond calligram construction. It adds a new dimension to classical shape deformation and represents a general problem setting: how to deform multiple entities in a confined space where the deformations constrain each other. Our solution is admittedly preliminary but can potentially stimulate future work.

Limited training data. In terms of data, currently, we believe that the letter samples used for training still only cover a limited space of letter deformations. Preparing for extensive and quality training data can be non-trivial and costly, e.g., to generate suitable word variations with letter rotations or to allow disconnections in letter shapes. With much more training data gathered, more advanced

Figure 18: *Two "failure" cases due to inadequate automatic layout path generation. Automatic result for the penguin (purple) is less revealing than an artist's creation (Figure 2). User strokes (green and red) allow multi-line calligrams to be generated, bringing closer resemblance to the artist's result. Automatically generated layout path for the snail does not lead to a satisfactory calligram; this can also be fixed with a user hint (green).*

schemes, e.g., via the use of deep convolutional neural networks, are possible. We leave these possibilities for future work.

Technical limitations. As discussed before, one limitation is in terms of the alphabet and fonts our method adopts. Extensions to other alphabets and fonts should be straightforward under the same framework. That said, an alphabet- and font-specific method for calligram generation is not unexpected since an aesthetic calligram is unlikely formed by mixing letters from different alphabets or drastically different fonts. On a more technical front, a main limitation is our assumption that the layout path is a single low-curvature line. In Figure 18, we show an automatically generated penguin calligram that may not be as elegant as the artist's design, which encompasses three or four low-curvature layout paths. Multi-line calligrams can be generated with the aid of user strokes.

Our method may also fail to produce an elegant result, e.g., for the snail, when the automatically computed layout path is inadequate. Again, a user stroke can help, as shown in Figure 18. Also, our current method does not account for symmetry of the letters, e.g., the symmetry between the two stems of the letter 'n'. A consideration of symmetry may prevent the letter 'n' from being assigned to the leg of the flamingo (Figure 14). Finally, our method currently does not handle input shapes with high genuses.

Future works. Aside from addressing the limitations and speeding up the search, we could also explore a few other future directions. Instead of providing both a word and an image as input to calligram generation, a user may only provide a textual description, e.g., "a jumping tiger", for the image. It is also interesting to compute *reusable* letter shapes or segments thereof which can contribute to several input shapes. Finally, a fun future problem is to develop a technique to *morph* between two word calligrams, shape-to-shape and letter-to-letter, in a visually pleasing way.

Acknowledgements

We first thank all the reviewers for their valuable comments and feedback. We owe our gratitude to Mr. Dan Fleming for his creative work which inspired our research and for his generosity in allowing us to use images of his works in this paper. Thanks also go to Daniel Cohen-Or and Nicholas Vining for proofreading, as well as to students from Hengyang Normal University and Dalian University of Technology for their hard work on data collection and labeling. This work is supported in part by grants from NSERC Canada, the Young Scientists Fund of the National Natural Science Foundation of China (Grant No. 61502153), the National Natural Science Foundation of China (Grant No. 61363048), and the Program of Key Disciplines in Hunan Province.

References

ADOBE, 2010. Illustrator CS5 adobe.com/products/illustrator.

ANDREWS, T. 2012. Computation time comparison between matlab and C++ using launch windows. *Research report submitted to American Institute of Aeronautics and Astronautics, California Polytechnic State University San Luis Obispo*.

ARUOBA, S. B., AND FERNANDEZ-VILLAVERDE, J. 2014. A comparison of programming languages in economics. *National Bureau of Economic Research, Working Paper 20263*.

CAI, D., CHI, C.-F., AND YOU, M. 2008. Assessment of english letters' legibility using image descriptors. *Perceptual and motor skills 107*, 2, 618–628.

CAMPBELL, N. D., AND KAUTZ, J. 2014. Learning a manifold of fonts. *ACM Transactions on Graphics (TOG) 33*, 4, 91.

CHELLAPILLA, K., LARSON, K., SIMARD, P., AND CZERWINSKI, M. 2005. Designing human friendly human interaction proofs (HIPs). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI'05, 711–720.

DEROSE, T., AND BARSKY, B. A. 1988. Geometric continuity, shape parameters, and geometric constructions for catmullrom splines. *ACM Trans. Graph. 7*, 1, 1–41.

GAL, R., SORKINE, O., POPA, T., SHEFFER, A., AND COHEN-OR, D. 2007. 3d collage: expressive non-realistic modeling. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, ACM, 7–14.

GOFERMAN, S., TAL, A., AND ZELNIK-MANOR, L. 2010. Puzzle-like collage. In *Computer Graphics Forum*, vol. 29, Wiley Online Library, 459–468.

HARRIS, C., AND STEPHENS, M. 1988. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, 147–151.

HELMOND, A., 2010. Textaizer mosaizer.com/Textaizer.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3, 1134–1141.

JACOBSON, A., BARAN, I., POPOVIC, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph. 30*, 4, 78.

JIN, C., AND WANG, L. 2012. Dimensionality dependent pac-bayes margin bound. In *NIPS*, 1043–1051.

JOACHIMS, T. 2002. Optimizing search engines using clickthrough data. In *KDD*, 133–142.

KENDALL, M. G. 1938. A new measure of rank correlation. *Biometrika 30*, 1/2, 81–93.

KIM, J., AND PELLACINI, F. 2002. Jigsaw image mosaics. *ACM Transactions on Graphics 21*, 3, 657–664.

KOVESI, P., 2006. Matlab and octave functions for computer vision and image processing. http://people.csse.uwa.edu.au/pk/Research/MatlabFns/index.html.

LIANG, L., AND GRAUMAN, K. 2014. Beyond comparing image pairs: Setwise active learning for relative attributes. In *2014*, 208–215.

LIPMAN, Y., LEVIN, D., AND COHEN-OR, D. 2008. Green coordinates. *ACM Trans. Graph. 27*, 3.

LOOMIS, J. M. 1990. A model of character recognition and legibility. *Journal of Experimental Psychology: Human Perception and Performance 16*, 1, 106–120.

LUO, L., SHEN, C., LIU, X., AND ZHANG, C. 2015. A computational model of the short-cut rule for 2d shape decomposition. *IEEE Transactions on Image Processing 24*, 1, 273–283.

MAHARIK, R., BESSMELTSEV, M., SHEFFER, A., SHAMIR, A., AND CARR, N. 2011. Digital micrography. *ACM Trans. on Graph 30*, 4, 100:1–100:12.

O'DONOVAN, P., LIBEKS, J., AGARWALA, A., AND HERTZMANN, A. 2014. Exploratory Font Selection Using Crowdsourced Attributes. *ACM Transactions on Graphics (Proc. SIGGRAPH) 33*, 4.

PARIKH, D., AND GRAUMAN, K. 2011. Relative attributes. In *ICCV*, 503–510.

PHAN, H. Q., FU, H., AND CHAN, A. B. 2015. Flexyfont: Learning transferring rules for flexible typeface synthesis. *Computer Graphics Forum 34*, 7, 245–256.

RUSSELL, S. J., AND NORVIG, P. 2010. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

SHEEDY, J. E., SUBBARAM, M. V., ZIMMERMAN, A. B., AND HAYES, J. R. 2005. Text legibility and the letter superiority effect. *Human Factors 47*, 4, 797–815.

WEBER, O., BEN-CHEN, M., AND GOTSMAN, C. 2009. Complex barycentric coordinates with applications to planar shape deformation. In *Computer Graphics Forum*, vol. 28, Wiley Online Library, 587–597.

WIKIPEDIA, 2014. Calligram — wikipedia, the free encyclopedia. [Online; accessed 9-October-2014].

XU, J., AND KAPLAN, C. S. 2007. Calligraphic packing. In *Proceedings of Graphics Interface 2007*, ACM, 43–50.

XU, X., ZHANG, L., AND WONG, T.-T. 2010. Structure-based ascii art. *ACM Trans. Graph. 29* (July), 52:1–52:10.

ZHU, J.-Y., AGARWALA, A., EFROS, A. A., SHECHTMAN, E., AND WANG, J. 2014. Mirror mirror: Crowdsourcing better portraits. *ACM Trans. on Graph 33*, 6.

ZITNICK, C. L. 2013. Handwriting beautification using token means. *ACM Trans. on Graph 32*, 4, 53:1–53:8.