DecoBrush: Drawing Structured Decorative Patterns by Example

Jingwan Lu[1,3] Connelly Barnes[2] Connie Wan[1] Paul Asente[3] Radomir Mech[3] Adam Finkelstein[1]
[1]Princeton University [2]University of Virginia [3]Adobe Research

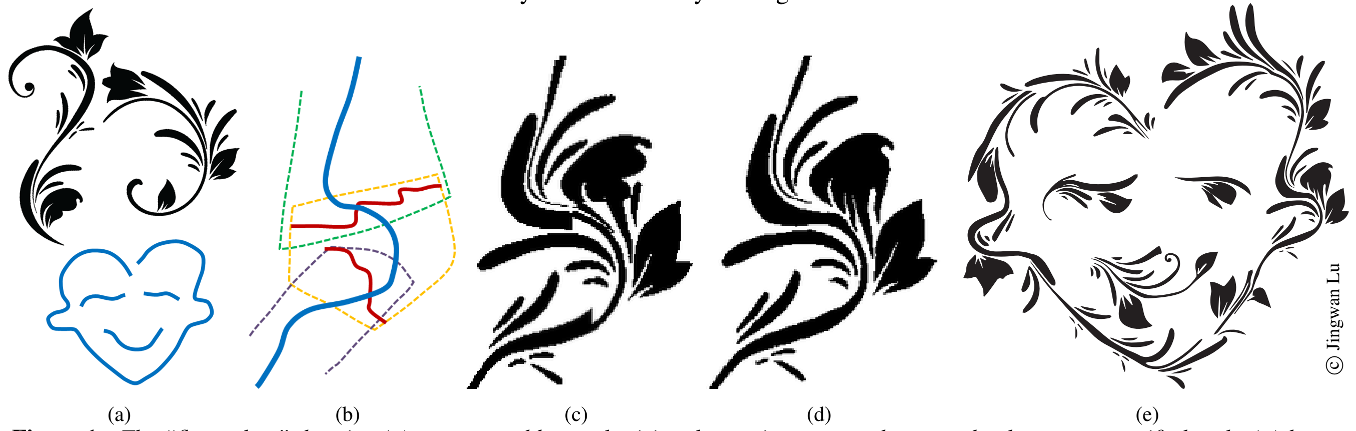


Figure 1: *The "flower boy" drawing (e) was created by synthesizing decorative patterns by example along user-specified paths (a)-bottom. (a)-top and Figure 2d show a subset of the exemplars for this floral style. (b), (c), and (d) show a closeup of the right ear. In (b), portions of the exemplars, shown outlined with dashes in green, yellow and purple, are matched along the user's strokes. Graph-cuts shown in red reduce but do not completely eliminate artifacts in the overlap regions, as seen in (c). Starting with this initialization, we run texture synthesis by example (d) and then vectorize the result (e).*

Abstract

Structured decorative patterns are common ornamentations in a variety of media like books, web pages, greeting cards and interior design. Creating such art from scratch using conventional software is time consuming for experts and daunting for novices. We introduce DecoBrush, a data-driven drawing system that generalizes the conventional digital "painting" concept beyond the scope of natural media to allow synthesis of structured decorative patterns following user-sketched paths. The user simply selects an example library and draws the overall shape of a pattern. DecoBrush then synthesizes a shape in the style of the exemplars but roughly matching the overall shape. If the designer wishes to alter the result, DecoBrush also supports user-guided refinement via simple drawing and erasing tools. For a variety of example styles, we demonstrate high-quality user-constrained synthesized patterns that visually resemble the exemplars while exhibiting plausible structural variations.

CR Categories: I.3.4 [Computer Graphics]: Graphics Utilities

Keywords: stroke, stylization, data-driven, decorative patterns

Links: DL PDF WEB VIDEO DATA

1 Introduction

Designers rely on structured decorative patterns for ornamentation in a variety of media such as books, web pages, formal invitations, commercial graphics and interior design. Digital artists often create such patterns from scratch in software like Adobe Illustrator. The artwork is typically represented as a set of outlines and fills using vector primitives like Bézier curves. In conventional software, the creation process can take hours to complete, even for skilled designers, and represents a steep learning curve for non-experts. Moreover, restructuring or repurposing an existing pattern can be painstaking, as it can involve many low-level operations like dragging control points.

This paper introduces "DecoBrush", a data-driven drawing system that allows designers to create highly structured patterns simply by choosing a style and specifying an approximate overall path. The input to our system is a set of pre-designed patterns (*exemplars*) drawn in a consistent style (Figure 1a). Off-line, we process these exemplars to build a stroke library. During the drawing process, the user selects a library, and then sketches curves to indicate an intended layout. The system transforms the query curves into structured patterns with a style similar to the exemplar library but following the sketched paths. DecoBrush allows both experts and non-experts to quickly and easily craft patterns that would have been time consuming for the experts and difficult or impossible for novices to create from scratch.

Our work is inspired by recent progress in example-based drawing methods, such as the work of Lu et al. [2013] as well as work enabling structural variation like that of Risser et al. [2010]. The kinds of highly-structured decorative patterns shown in this paper present a particular challenge for example-based texture synthesis, where most of the methods have been optimized for continuous-tone imagery where artifacts are largely hidden by fine-scale texture.

Our pattern synthesis works as follows. We first divide the query path into segments that are matched to similar segments among the exemplars, using dynamic programming to efficiently optimize this assignment. We then warp the segments using as-rigid-as-possible deformation to approximate the query path. Next at the *joints* where neighboring segments overlap, we rely on graph cuts (shown red in Figure 1b) to form a single, coherent figure that approximates the hand-drawn path. Despite the use of nearly-rigid warps combined with graph cuts at the joints, the resulting figure generally

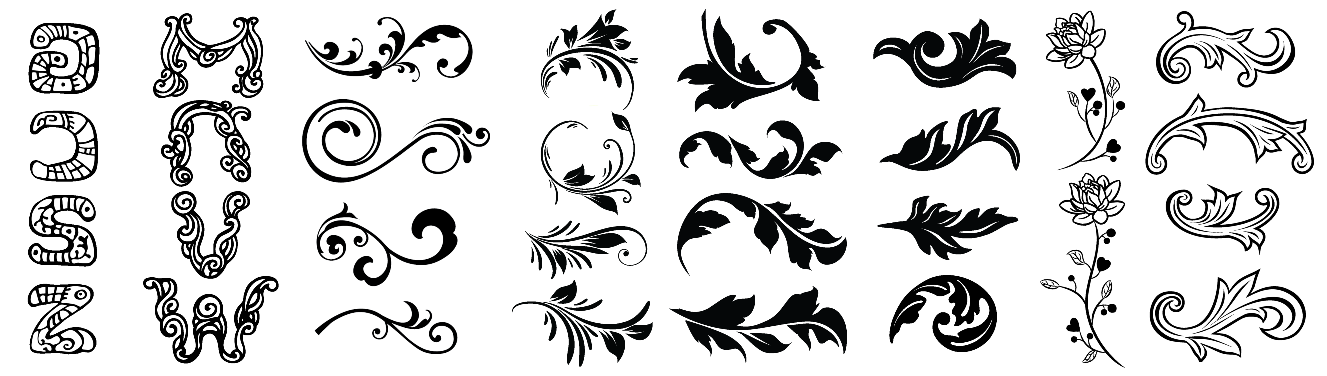


Figure 2: *Structured stroke patterns used as exemplars. Copyrights: (a) Depositphotos.com/Rud-Volha, (b) Depositphotos.com/pimonova, (c-d) Webdesignhot.com, (e) Ozerina Anna/Shutterstock.com, (f) Ornate Vector Florals, (g) Depositphotos.com/to_mua_to, (h) Daichi Ito.*

suffers from small stretching artifacts from the warp as well as misalignments at the joints (Figure 1c). Therefore we adapt structure-preserving hierarchical texture synthesis techniques to repair, refine and diversify the query stroke appearance. Our texture synthesis pipeline both ameliorates feature distortion within the warped exemplar segments and repairs broken local structures at the joints (Figure 1d). We also show that when synthesizing from coarse levels the pipeline brings additional structural variation into the design, consistent with the style of the exemplars. If the designer is not completely satisfied with the result of this process, our system also supports user-guided refinement of the synthesized result – a few new strokes with brush or eraser tools form the input to a supplemental texture synthesis step that seamlessly incorporates the newly-drawn constraints with earlier results. Finally, we obtain a vector representation of the synthesis results (Figure 1e). The proposed synthesis pipeline is efficient, which facilitates applications such as user-guided pattern diversification and sketch-based decoration of confined regions.

Our primary contribution is the idea of synthesizing structured decorative patterns along user-sketched paths, thereby generalizing the conventional digital "painting" concept beyond the scope of natural media to incorporate art traditionally represented in vector form. We show that starting from an initial figure carefully constructed from exemplar segments, a hierarchical texture synthesis pipeline can efficiently produce structured decorative patterns. DecoBrush is the first system to be able to synthesize by example structured patterns like those shown in this paper. Moreover, our pipeline allows the designer to specify both the overall shape of the resulting pattern as well as user-guided refinement to control fine detail.

2 Related Work

There have been **procedural approaches** for synthesizing decorative patterns. The early pioneer work by Wong et al. [1998] use procedural rules to grow decorative patterns for the application of automatically filling a confined region. Other than reshaping and resizing the target region, they do not investigate other types of user interactions. Other procedural approaches allow growing structured patterns along a user-specified path ([Chen et al. 2012] and [Anderson and Wood 2008]), but the range of supported styles is heavily limited by the employed simplistic procedural rules. Měch and Miller [2012] introduce an interactive procedural modeling framework, which generates complex decorative patterns. The procedural rules target plant-like structures and are hand-crafted by professional artists. Defining procedural rules for faithful reproduction of general structured patterns remains a challenging problem.

A variety of research has addressed **data-driven stroke synthesis**, where the general goal is to synthesize the appearance of the query stroke by example based on a 1D input path. The *skeletal strokes* of Hsu et al. [1993] warp pieces of the same structured texture to stylize lines. They focus on controlling the deformation of the selected features on the picture and do not introduce structural variations into the synthesis results. *Curve analogies* [Hertzmann et al. 2002] introduces a statistical model for synthesizing new curves by example. However, they target 1D curves instead of the shape and texture of 2D strokes. Previous work use small patches of stroke exemplars to stylize users' input lines ([Kim and Shin 2010] and [Ando and Tsuruno 2010]). The work of Lu et al. [2013] improves the synthesis quality by warping and blending long exemplar segments. They also synthesize the natural media stroke interactions by examples. Other work ([Zhou et al. 2013] and [Lukáč et al. 2013]) support exemplars with repetitive small-scale local structures. We target multiple exemplars that contain high-level large-scale structures and characteristic appearance at both ends of the strokes. For this purpose, we borrow the piecewise matching idea from Lu et al. [2012], [2013] as the first step of our synthesis pipeline, which gives a good initial guess of the possible query stroke structure.

Another related line of research is **structured texture synthesis**. Markov Random Field-based texture synthesis approaches, stemming from the pioneering work of Efros and Leung [1999], make the assumption that the appearance of a pixel is only dependent on a local spatial neighborhood regardless of the rest of the image. This model works well for homogeneous continuous-tone imagery. However, this assumption is weakened for highly structured textures where large-scale geometric features such as long connected lines are present, especially in binary or vector imagery where artifacts are difficult to hide. Previous work introduce hierarchical tile-based synthesis approaches, which better handle structured textures. Instead of synthesizing the image structure sequentially pixel by pixel, which often results in unrecognizable structures, previous work ([Lefebvre and Hoppe 2005] and [Lefebvre and Hoppe 2006]) explore the idea of tiling the exemplars and introducing structural variations by coordinate jitter and correction. They focus on synthesizing larger textures from a single small exemplar. Risser et al. [2010] introduce structure-preserving jitter and showed further evidence that the hierarchical tile-based synthesis framework is able to preserve, modify and repair image structures. They also extend the synthesis pipeline to consider multiple source exemplars. However, their goal is to synthesize variations of the exemplars without any user constraints. We, on the other hand, target synthesizing new structured patterns following a user specified path. We borrow ideas from texture synthesis methods and follow the hierarchical upsampling and correction pipeline. The

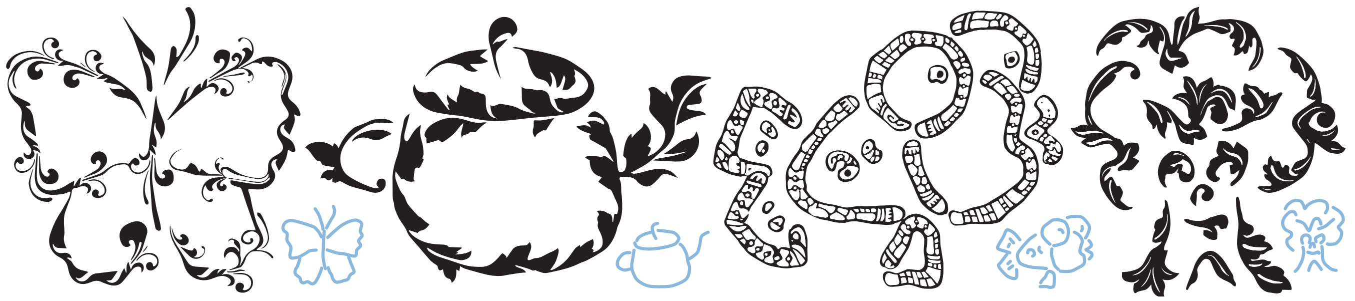


Figure 3: *Drawing results. From left to right, the butterfly, teapot, fish and tree lady are synthesized using the exemplars in Figure 2c, 2e, 2a, 2f. The blue curves in the lower right corner show the input query paths.* ©*Jingwan Lu*

major difference is that we enforce user constraints by initializing the synthesis field using the matched exemplar segments that follow a similar path to the query. We also modify the upsampling and correction steps to suit our goals. The synthesized patterns contain meaningful structures and closely follow the input query path.

There has been work to use texture synthesis techniques to **arrange discrete elements**, [Barla et al. 2006], [Ijiri et al. 2008], [Hurtut et al. 2009], [Ma et al. 2011], [Kazi et al. 2012], [Landes et al. 2013] and [AlMeraj et al. 2013]. The general idea is to define relationships between elements on a graph and synthesize new patterns by searching for input elements with graph-based neighborhood matching and pasting them to target locations. These approaches only support discrete primitives that do not intersect each other. However, the exemplars we have contain long, curly, intersecting lines and complex layout which pose additional challenges for defining inter-element relationships.

3 Algorithm Overview

Recent advances in sketch-based stroke synthesis ([Lu et al. 2013] and [Lukáč et al. 2013]) demonstrate promising results for synthesizing natural media strokes. The input strokes are not structured and can be combined using simple alpha blending. These approaches generate results that closely follow the user's sketches, but cannot be directly applied to decorative patterns due to the difficulty of combining exemplar segments of very different structures. On the other hand, there has been remarkable progress in texture synthesis for structured exemplars ([Lefebvre and Hoppe 2006] and [Risser et al. 2010]) . However, the synthesis of high-level semantic contents following user constraints remains a challenging problem. We combine the advantages of both types of approaches for the new application of "painting" decorative patterns.

To prepare the exemplars for synthesis, we need to extract a stroke-like semantic structure for each exemplar. We optionally convert the exemplars to a signed distance field representation (Section 4.1), which can preserve lines better for some exemplars. We then semi-automatically parameterize the exemplar stroke (Section 4.2). We also precompute the neighborhood information at every pixel so that runtime synthesis is efficient (Section 4.3).

At runtime, given a query stroke spine specified by the user, we look for long segments of exemplar strokes that have similar shape (Section 5.1) and apply as-rigid-as-possible deformation to align them with the query path (Section 5.2). We then apply graph cut to optimize the transition boundaries between the nearby exemplar segments (Section 5.3). The result of graph cut provides a good initial layout for the query stroke. We then use hierarchical texture synthesis approach to further reduce the warping artifacts and repair the broken image structures (Section 5.4). Finally, we vectorize the synthesized result using Adobe Illustrator.

4 Exemplar Processing

We target synthesizing structured patterns such as the ones in Figure 2. The exemplars can have either a raster or vector representation. We have focused on synthesizing decorative florals, stylized fonts and other structured vector patterns. The exemplars share common characteristics: 1) They are composed of variable width curves and other simple solid-colored geometric shapes; 2) They have stroke-like structures with a clear directionality indicated by a center curve and/or the orientations of a group of curves; 3) They have unique appearance at the beginning and the end of the stroke.

We collect several libraries of structured patterns. Each library contains between 8 and 12 strokes all following a consistent design. The design is characterized by the choice of geometric primitives, the branching structures, the thickness and curviness of lines, etc. We collect strokes of different shapes, lengths and spine curvatures, which are then semi-automatically processed to create a library. Each library stroke is limited to lie within a square of resolution 512x512. We rasterize the exemplar strokes and optionally compute Signed Distance Field (SDF) representation (Section 4.1). Then we parameterize the exemplars (Section 4.2) in preparation for the runtime synthesis. Finally, we also pre-compute per-pixel neighborhood information on the raster exemplar (Section 4.3) for efficient runtime synthesis.

4.1 Signed Distance Field

Before further processing, we rasterize the input exemplars and optionally calculate the Signed Distance Field (SDF) [Leymarie and Levine 1992]. In the SDF, black pixels have negative distances to the shape boundary and the white pixels have positive distances. The SDF effectively thickens the feature lines and introduces gradient information enriching the neighborhood information, which facilitates the piecewise matching process (Section 5.1) and the texture synthesis process (Section 5.4). Before the final vectorization, we simply threshold the SDF to extract a level set. However, as shown in Figure 4, the use of SDF can prove detrimen-

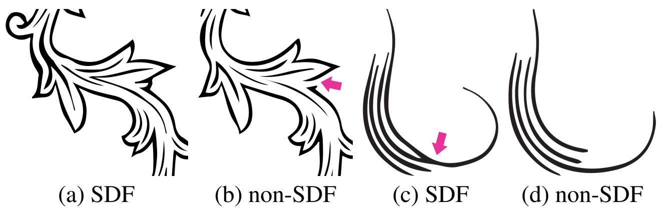


Figure 4: *Tradeoff of using SDF vs not using SDF. Lines are often joined better when using SDF (a,b), however this can also sometimes make spurious joins between unrelated lines (c,d).*

tal on some exemplars, where it can join unrelated lines. Thus we leave SDF usage as optional (we note in the supplementary where it has been used). The inset figure on the right shows an example of the SDF representation.

4.2 Exemplar Parameterization

Each decorative stroke has a clear directionality. The directionality is sometimes suggested by a continuous line that runs from the start to the end of the stroke. Other times, a number of lines collectively hint where the center branch is. To conduct synthesis, we need to extract the location of the main branch as a sequence of sample positions. To simplify the problem, we manually specify the start and end points of the main branch. Note that the start and end points do not have to be located on the tips of the structure or even on any of the feature lines (Figure 5a). The curls at the ends of the stroke serve for the purpose of decoration more than direction indication. They are part of the style that should be preserved in the synthesis results. By straightening the main branch towards the ends of the stroke, we can effectively simulate such curly appearance even when the query path is relatively straight.

With the two end points specified, we solve for the main branch by optimizing a path between them. Our goal is that the path does not wiggle too much, roughly follows the edge tangent flow (ETF) ([Kang et al. 2007]) of the exemplar, and is smooth. We find a polyline found by running Dijkstra's algorithm on a pixel grid containing the exemplar, where nodes of the graph are the pixels, and edges connect each pixel to neighbors distributed roughly uniformly in 32 directions. Each edge carries a weight w based on the vector v between the two adjacent pixels as $w = |v| + w_f(1 - f^p) + w_c(1 - c)$ where: f is the dot product of the v with the ETF sampled at its midpoint; c is the difference in curvature at the ends of this edge; and w_f, w_c and p are user-specified constants that control the behavior of these terms. (We found $w_f = 10$, $w_c = 3$ and $p = 4$ to work well in practice.) Finally, we fit a quartic polynomial to the vertices of the polyline to produce a smooth main branch that fits the artwork.

This typically produces a good match to the artwork and requires very little user effort. However, sometimes it fails to match the artwork well, especially where semantic understanding of the shape is required. In these cases, it is straightforward for the user to sketch a preferred path for the main branch directly over the artwork. The optimized main branch (red to green indicates the stroke directionality in Figure 5a) coincides with the "center" of the pattern, slightly straightened up at both ends of the stroke. We then follow [Lu et al. 2013] to obtain the uv parameterization of the whole stroke (Figure 5b). Each stroke consists of between 30 and 100 samples, where a sample $\mathbf{t} = \{\hat{\mathbf{x}}, \hat{\mathbf{l}}, \hat{\mathbf{r}}\}$ consists of the 2D positions of the spine, left and right outline samples respectively.

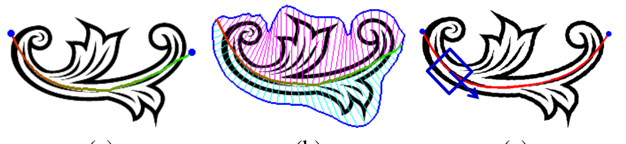


Figure 5: *Parameterization. (a) The user specifies the blue end points and the spine. The transition from red to green indicates the directionality of the stroke from the start to the end. (b) The system then automatically parameterizes the stroke. (c) Given the end points, we can optionally optimize the red spine automatically. The blue square and arrow indicate the search neighborhoods are aligned with the stroking orientation.*

4.3 Per-Pixel Processing

For efficient runtime synthesis, we further pre-process the exemplar image (or SDF if used) to extract hierarchical per-pixel information. We calculate three Gaussian pyramid levels for each exemplar. Note that since we do not jitter the coordinates, the Gaussian stack [Lefebvre and Hoppe 2005], which requires more memory and computation, is not needed. At each level, we only consider the pixels inside the automatically extracted stroke outline. We calculate a per-pixel orientation $\omega^e = (\omega^e_x, \omega^e_y)$, by interpolating the stroking orientations at the spine and the outline samples. The orientation of the spine is defined as $\bar{\mathbf{x}}_i = \hat{\mathbf{x}}_i - \hat{\mathbf{x}}_{i-1}$. The orientation of the outline is defined as $\perp \bar{\mathbf{l}}_\mathbf{i}$ and $\perp \bar{\mathbf{r}}_\mathbf{i}$, where $\bar{\mathbf{l}}_i = \hat{\mathbf{l}}_i - \hat{\mathbf{x}}_i$ and $\bar{\mathbf{r}}_i = \hat{\mathbf{r}}_i - \hat{\mathbf{x}}_i$. We then calculate an oriented 5x5 local neighborhood for each pixel with the upright direction (blue square and arrow in Figure 5c) of the neighborhood aligned with the per-pixel orientation ω_e.

5 Query Stroke Synthesis

Here we describe our process for converting a user-drawn stroke into a decorative pattern of roughly the same overall shape. First, we consider how to match segments from the exemplar set to overlapping portions of the query path. Next, we warp their shapes to better match the path. Because the warped shapes overlap at joints between the segments, we use graph-cuts to seek a seamless boundary between neighboring segments. Finally, to address distortion artifacts and mismatched boundaries, we use texture synthesis to find a shape similar to the output from warping and graph-cuts but everywhere locally matches the exemplar set.

5.1 Piecewise Matching

Given a query spine (blue curve in Figure 6a), we estimate a rough 2D shape (based on pressure sensing) and the uv parameterization (Section 4.2). We then adapt the piecewise matching algorithm proposed by Lu et al. [2013] to find a sequence of exemplar segments from the library and merge them together for the query stroke. To collect candidate nearest neighbors from the exemplars, for each query sample, we use the same feature vector as proposed by Lu [2013] that includes the turning angle, the stroke width and the distance to the endpoints. In the dynamic programming step, we heavily penalize the ends of the query stroke being matched to the middle parts of the exemplars. For structured exemplars, it is especially critical to avoid cutting short the ends, since it might destroy important features and lead to broken lines that are hard to repair. We also avoid matching the middle parts of the query stroke to the end parts of the exemplar strokes and heavily penalize any jumps between library segments that have very different appearance or have very different stroke thickness at the segment boundaries (thickness is the rib length in the uv parameterization). After finding the appropriate library segments, we extend each segment in both directions to overlap the nearby matched segments. In Figure 6, the query stroke is matched to three exemplar segments.

5.2 As Rigid As Possible Deformation

To synthesize decorative patterns closely following the user's intent, we need to align the spines of the exemplar segments with the input query spine. The warping method introduced by [Lu et al. 2013] leads to noticeable texture distortion for the structured exemplars (Figure 6a). We instead use As Rigid As Possible Deformation ([Igarashi et al. 2005]) to reduce the distortions. The query stroke's spine samples (blue curve in Figure 6a) are fixed at the input locations. The locations of the outline samples (green curve in Figure 6b) are optimized to reflect the original shape of the

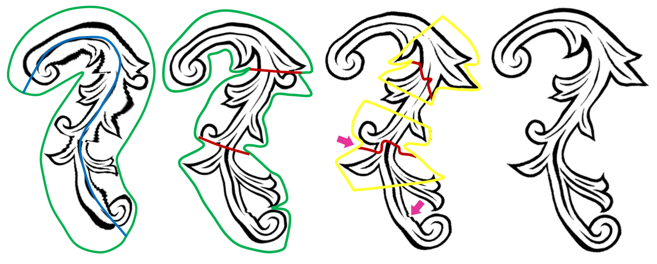

(a) (b) (c) (d)

Figure 6: *Synthesis Pipeline. (a) The warping introduced by [Lu et al. 2013] results in noticeable distortions. (b) We apply As Rigid as Possible Deformation to recalculate the shape of the query stroke. (c) We find the optimal cuts (red curves) within the overlapped regions (outlined in yellow). (d) We apply hierarchical texture synthesis to reduce the residual artifacts.*

exemplar segments. The use of As Rigid As Possible Deformation minimizes the amount of structural distortion and therefore leaves an easier task for the texture synthesis step (Section 5.4) to remove the residual warping artifacts. Note that though it is later changed, the query stroke's input outline (green curve in Figure 6a) is utilized for the piecewise matching process in constructing the feature vector. Thus, if thin query strokes are drawn with little pressure, then thin exemplar segments are likely to be matched.

5.3 Graph cut

After the adjacent library segments are extended and deformed, we detect the region where they overlap (outlined by yellow curves in Figure 6c). We apply a 2-label planar graph cut algorithm [Schmidt et al. 2009] to find the least cost cut that smoothly transitions from one library segment to another. We define a graph on the overlap region where each pixel is a node and each node has four edges connecting the adjacent nodes. Each pixel (j, k) corresponds to two stroke textures, L and R. Then the cost of each node $\mathbf{E}_c$ and edge $\mathbf{E}_d$ is defined as:

$$\mathbf{E}_c = w_c(2 - L_{j,k} - R_{j,k}) + C \tag{1}$$

$$\mathbf{E}_d = \begin{cases} (L_{j,k} - R_{j,k+1})^2 & \text{going up} \\ (L_{j,k} - R_{j,k-1})^2 & \text{going down} \\ (L_{j,k} - R_{j+1,k})^2 & \text{going right} \\ (L_{j,k} - R_{j-1,k})^2 & \text{going left} \end{cases} \tag{2}$$

$\mathbf{E}_c$ penalizes lengthy cuts and cuts that pass through black feature lines on the exemplar, which might result in undesirable changes in line thickness. $\mathbf{E}_d$ represents the difference in intensity of the two adjacent pixels along the edge direction from the two library segments respectively. We use $C = 0.02$ and $w_c = 0.01$. Figure 6c shows the result after applying graph cut, in comparison to that of applying a naive cut along the stroke rib in the middle of the overlap region in Figure 6b. The red curves indicate the cuts.

The output of the piecewise matching and the graph cut steps include, for every query pixel, a local orientation $\omega^q = (\omega^q_x, \omega^q_y)$ (Section 4.3) and a 3D texture coordinate $\mu = (\mathbf{i}, \mathbf{x}, \mathbf{y})$, where $\mathbf{i}$ indicates the index of the exemplar from which this pixel originates.

5.4 Texture Synthesis

The graph cut step (Section 5.3) finds good transition boundaries most of the time, but at times the results might contain undesirable distortions and broken or jagged line structures (Figure 8a). We therefore apply a fast hierarchical texture synthesis step to fix the distortion and the broken structures, inspired by the approach proposed by Lefebvre et al. [2005]. Their key idea is to initialize, synthesize and upsample the exemplar coordinates instead of the pixel values. During the upsample step, the finer level pixels inherit and offset the coordinates of the coarser level. Using coordinate inheritance better preserves the sharp image features compared to applying bilinear interpolation on the pixel values. We follow their synthesis pipeline, but modify the steps to suit our need. The synthesis pipeline is initialized using the exemplar coordinates produced by graph cut (Section 5.4.1). Then, we improve the image structures by iteratively replacing the initialized exemplar coordinates with the center coordinates of the appropriate exemplar neighborhoods (Section 5.4.2). The synthesized exemplar coordinates are then upsampled to the next finer level (Section 5.4.3). We alternate the correction and upsample steps until we reach the finest level and skip the upsample step at the finest level. We found three synthesis levels produce good results (Figure 6d).

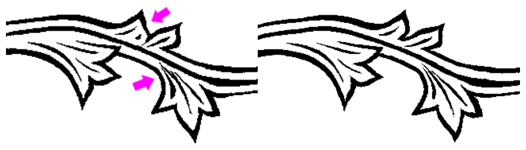

(a) 1 orientation (b) 11 orientations

Figure 7: *Searching over more orientations slightly improves the synthesis results.*

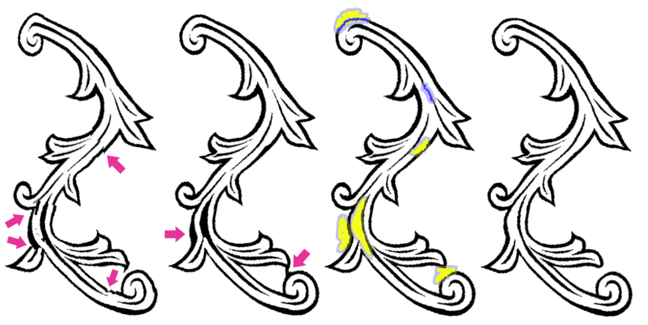
5.4.1 Initialization

We initialize the texture synthesis pipeline by performing the graph cut step with the exemplars at low resolution (usually 128x128). The graph cut outputs an exemplar coordinate to initialize every synthesis pixel.

5.4.2 Correction

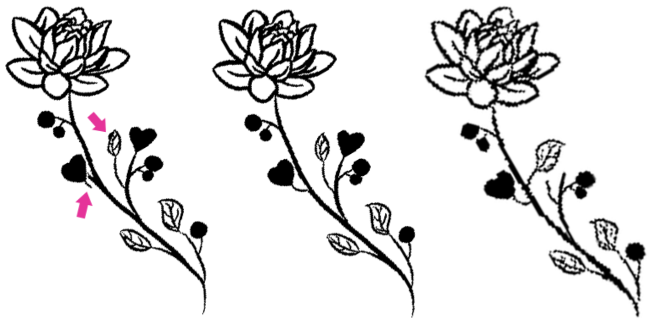
The essential step that repairs the broken image structures is inspired by the correction step proposed by Lefebvre et al. [2005], which we briefly review. During correction, the exemplar coordinate of each query pixel is replaced by the center coordinate of an exemplar neighborhood that is most similar to the query neighborhood measured by L^2 distance. The correction step also uses the idea of coherent synthesis ([Ashikhmin 2001]), and considers the 3x3 immediate neighbors of the current query pixel. Correction can be applied to non-adjacent pixels independently, which allows fast parallel processing. Several iterations of this step for each pixel at each synthesis level ensure that the synthesis result is locally similar to the exemplars everywhere and therefore faithful to the style.

We adapt this step in several ways. We sample rotated query neighborhoods with the up axis aligned with the per-pixel orientation, ω^q (calculated in the same way as in Section 5.3). Rotating both the exemplar and the synthesis neighborhoods to align with the stroking direction increases the chances of success for neighborhood matching. In the overlap region of the adjacent exemplar segments (red curves in Figure 6c), the image structures are the most challenging to repair. We therefore apply approximate nearest neighbor search [Muja and Lowe 2009] to find the most similar 5x5 exemplar neighborhood among all exemplar strokes for each 5x5 query neighborhood. For each query pixel, we additionally gather eight coherent exemplar neighborhoods by applying an appropriate offset to the coordinate of each pixel in the 8-connected neighborhood. We favor choosing the closest coherent neighborhood unless the L^2 distance to this neighborhood is significantly larger (5 times or more) than the L^2 distance to the approximate nearest neighbor. We find the use of strong coherence very important for fixing image structures and maintaining clean feature lines. For the non-overlap region, we find it satisfactory to only select the best one among the eight coherent neighborhoods for faster synthesis performance. To better



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

Figure 8: *Synthesis refinement. (a) The graph-cut step sometimes finds sub-optimal transition boundaries. (b) The texture synthesis step reduces but cannot completely remove the artifacts. (c) Users can apply refinement strokes (yellow indicates erasing and blue indicates adding). The light blue regions indicate the masks inside of which the synthesis pipeline modifies the exemplar coordinates. (d) The final synthesis result. Notice that though the user's scribbles are noisy, the synthesis pipeline produces smoother curves.*



(a) Graph-cut (b) Final result (c) Low resolution

Figure 9: *Texture synthesis improves on graph-cuts, if performed at sufficient resolution. (a) Graph-cut gives good initialization at the cost of small artifacts. (b) Texture synthesis removes the local artifacts and connects broken lines. (c) Using exemplars at lower resolution of* 256×256 *cannot preserve thin line structures in the synthesis results compared to using* 512×512.

tolerate the distortion introduced by the warping step (Section 5.2), we can optionally sample a few query neighborhoods rotated at several different angles around and including the stroking direction, ω^e. We perform the search for each rotated query neighborhood and select the exemplar coordinate μ and the optimal query orientation ω^e that gives the lowest L^2 neighborhood distance. Figure 7 demonstrates slight quality improvement when orienting the query neighborhoods at 11 different orientations around the stroking angle. Due to the diminishing return of searching over different orientations, all results of the paper are generated searching only one orientation exactly aligned with the stroking orientation.

5.4.3 Upsample

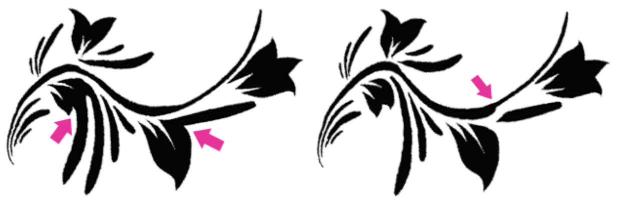
To initialize the synthesis field of the next finer level, we inherit the exemplar coordinates obtained by the correction step. To turn one center pixel into four surrounding pixels of the finer level, we use the optimal orientation ω^e to calculate the four offsets to be applied to the inherited exemplar coordinates. Specifically, each child pixel of the finer level l inherits the parent coordinate of level $l-1$ in the following way: $\mathbf{S}_l[\mathbf{p}+\Delta] := \mathbf{S}_{l-1}[\mathbf{p}] + \mathbf{J}_e^T(\mathbf{p})\mathbf{J}_q(\mathbf{p})\Delta$, $\Delta = (\pm 1/2 \ \pm 1/2)^T$, where $\mathbf{J}_q(\mathbf{p}) = \begin{pmatrix} \omega_x^q & \omega_y^q \\ \omega_y^q & -\omega_x^q \end{pmatrix}$ and $\mathbf{J}_e(\mathbf{p}) = \begin{pmatrix} \omega_x^e & \omega_y^e \\ \omega_y^e & -\omega_x^e \end{pmatrix}$ denotes the Jacobian matrices for the query and the exemplar neighborhoods respectively.

5.5 User-guided Refinement

Often the result of texture synthesis is satisfactory. However, in some cases, the graph cut step finds a sub-optimal cut due to the dissimilarity between the nearby segments (Figure 8a), which poses a challenging problem for texture synthesis. On top of the synthesized result (Figure 8b), users can optionally refine the design or fix the remaining artifacts by scribbling refinement strokes. The refinement strokes are drawn as black or white discrete pixels (visualized as blue or yellow pixels in Figure 8c) at the highest synthesis level, which facilitate adding or removing features respectively. To improve upon user's imprecise refinement strokes and obtain clean feature curves, we apply a slightly modified texture synthesis pipeline. We first dilate the refined pixels to obtain a binary mask $\mathbf{M}$ (visualized as light blue region in Figure 8c), which indicates the region of pixels to be changed. We downsample the refinement strokes and the binary mask for two levels until the coarsest synthesis level and seed the synthesis pipeline. At the coarsest level, we disable the coherence search for the user refined pixels, since there are no corresponding exemplar coordinates at these locations. We find the corresponding exemplar coordinates by searching for nearest exemplar neighborhood. We then upsample the updated exemplar coordinates to the next level and continue the usual synthesis pipeline (Section 5.4). At each synthesis level, the pixels outside the mask are locked and remain unchanged. At the coarsest level, the pixels inside the mask are updated to reflect the user's refinement constraints, and at other levels they are unconstrained. The hierarchical synthesis pipeline effectively smooths user's refinement strokes by matching to the exemplar. The synthesized curves are smoother than the initial refinement strokes, which is useful for novices who have difficulty drawing clean lines. All results of the paper, except the capital letters and Figure 8 are generated without synthesis refinement.

6 Results

The results in the paper are all synthesized with exemplars of a maximum resolution of 512×512 for performance and memory considerations. Figure 9 shows that for datasets which contain very thin lines, using resolution higher than 256×256 is crucial for maintaining thin line structures. On the other hand, for the exemplars in Figure 2b, 2e and 2f, we find the resolution of 256×256 is enough for synthesizing reasonable structures. The whole synthesis pipeline takes about 1-2 seconds to synthesize a stroke similar to the one in Figure 6. At 512×512, the synthesis is about 5 times slower. On average, each stroke in the paper takes about 8 seconds to finish. We did our performance measurement with the following



(a) 64×64 (b) 128×128

Figure 10: *Tradeoff of starting the texture synthesis from different resolutions. Starting the synthesis with exemplars at lower resolution improves the smoothness of the curves, but aggravates the problem of incorrectly connecting nearby sub-structures.*

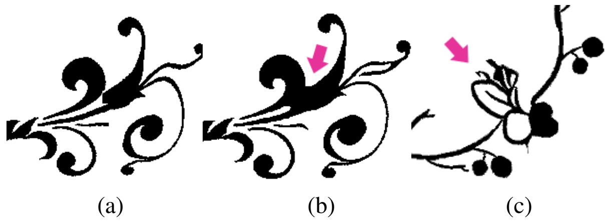


Figure 11: *Synthesis limitations: Even though graph-cut results in separate structures (a), synthesis sometimes falsely connects them together (b). Details can appear in inappropriate places, like the partial flower in the middle of the stroke (c).*

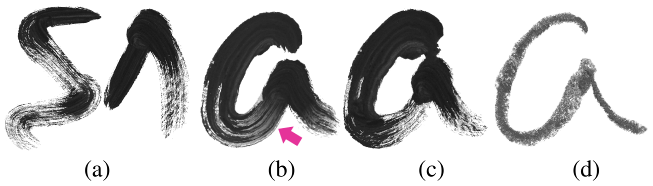


Figure 12: *Natural media synthesis. (a) Dry watercolor exemplars. (b) RealBrush with alpha blending. (c) DecoBrush using same exemplars. (d) DecoBrush using pastel exemplars.*

hardware, Intel Core i7 2.3 GHz CPU. We believe the performance can be improved to real-time by implementing the synthesis correction step on GPU ([Lefebvre and Hoppe 2005]).

The synthesis starting level can be determined by the user based on the quality of the graph cut result. For minor texture distortion (uneven and jagged curves), starting the synthesis using the exemplar resolution of 256 $\times$ 256 can sufficiently smooth out the feature curves. With more severe texture distortion or broken lines, synthesizing three hierarchy levels starting from 128 $\times$ 128 improves the results. Starting from even lower resolution presents a trade-off in the result quality. With the exemplar resolution of 64 $\times$ 64, close-by features are sometimes merged together undesirably (magenta arrows in Figure 10a), while on the other hand, the synthesized curves are usually smoother (magenta arrow in Figure 10b). This reveals a limitation (Figure 11a,b) of our approach. Curves that are near each other might be represented by only a few pixels at the coarse levels and are sometimes integrated into a single component through the synthesis correction steps.

We tested the robustness of our system by sketching strokes of different lengths and shapes synthesized with eight different exemplar libraries (see supplementary materials). We also made simple drawings and decorative designs shown in Figure 1e, 3, 20 and 19. Each of the designs was created by novice with less than 20 strokes in total. The blue input query paths are shown inset. The resulting drawings contain complex structures in the styles of the exemplars. We vectorized our drawing results using Adobe Illustrator.

If gray-scale image exemplars are used, our approach is also able to synthesize structured natural media exemplars with improved quality. Figure 12 demonstrates that compared to RealBrush, our synthesis results are free of blending artifacts and contain realistic variations in the stroke thickness. Figure 13 shows that our system also addresses colored exemplars. We convert the colored exemplars to grayscale image and synthesize them using the regular pipeline. Since the main synthesis component works by referencing texture coordinates, we obtain the synthesized colors by directly reading them from the exemplars.

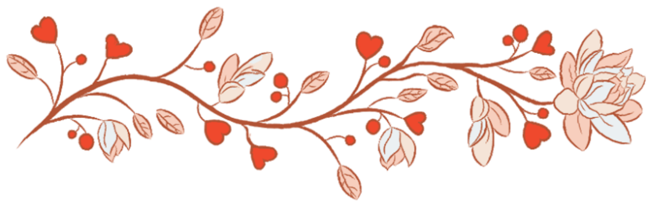


Figure 13: *Color synthesis. With simple extension to the synthesis pipeline, DecoBrush can also synthesize colored strokes.*

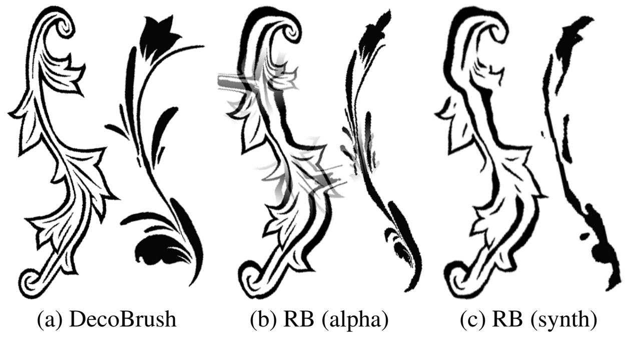


Figure 14: *Comparison with RealBrush [Lu et al. 2013], using exemplars in Figure 2h and 2d. (a) DecoBrush. (b) RealBrush using alpha blending. (c) RealBrush using texture synthesis.*

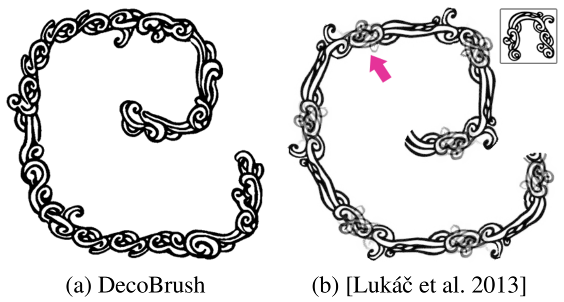


Figure 15: *Comparison with the Painting By Feature method of Lukáč et al. [2013]. (a) DecoBrush, using exemplar library in Figure 2b. (b) Method of Lukáč et al., using single exemplar shown in the upper right corner (note blending artifacts).*

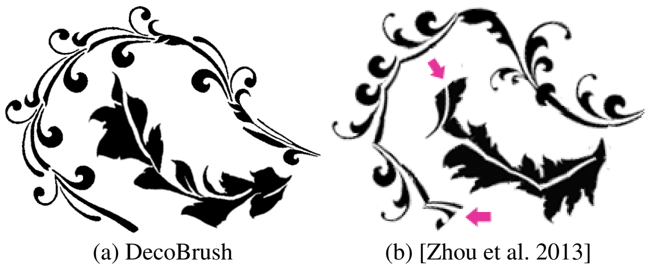


Figure 16: *Comparison with the work of Zhou et al. [2013]. The two strokes are synthesized with the exemplars in Figure 2c and 2e. (b) shows jagged spine and feature cutoff at the end.*

In Figures 14, 15 and 16, we compare with RealBrush [Lu et al. 2013], Painting by Feature [Lukáč et al. 2013] and the work of Zhou et al. [2013]. In Figure 14, the RealBrush alpha blending result introduces noticeable distortions, and the texture synthesis module removes delicate line structures, since it was designed for highly textured media such as sponge or glitter. In Figure 15, Painting by Feature introduces ghosting artifacts and does not handle the stroke ends differently from the middle. Figure 16 shows that the method of Zhou et al. chops the exemplars into smaller pieces, cuts off the exemplar features at the end of the query stroke, and generates jagged spines that do not closely follow the user's input path.

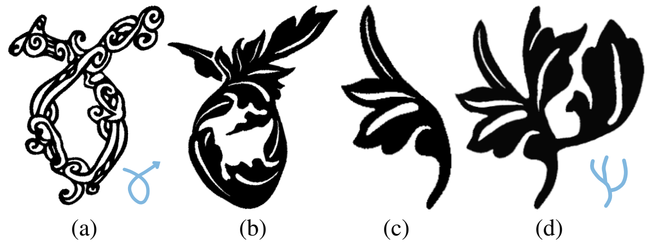


Figure 17: *Stroke Intersection and branching. (a-b) demonstrate the self-intersecting strokes. (d) shows the result of branching two strokes from the main spine in (c). (a) is synthesized with "doodle" in Figure 2b. (b-d) are synthesized with "palm" in Figure 2f,*

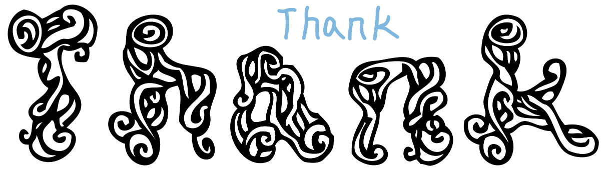


Figure 18: *The word "Thank" is synthesized using the exemplars in Figure 2b demonstrating intersection and branching appearance.*

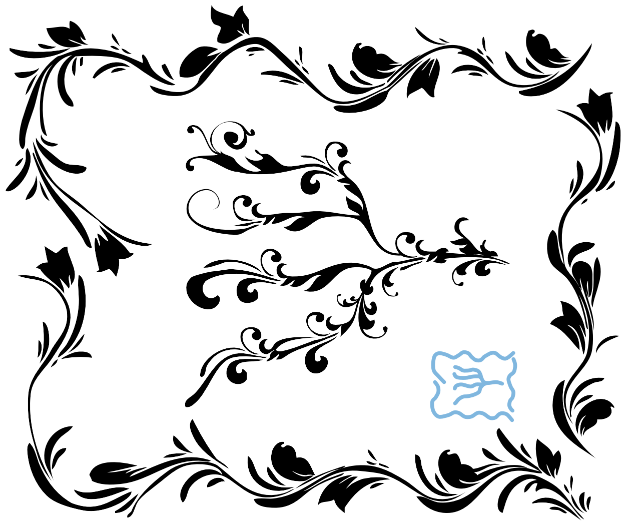


Figure 19: *The decorative frame is synthesized with Figure 2d. The "wine glass" inside is synthesized with Figure 2c. ©Jingwan Lu*

7 Limitations and Future Work

ecoBrush is the first system that can synthesize by example structural patterns like those shown in this paper along user-specified paths. Nevertheless there are a number of limitations to our system, and many of these suggest opportunities for future work.

Global structure limitations. The texture synthesis pipeline can only remove small local artifacts, but cannot fix artifacts on a more global scale, for example curvature discontinuities at a transition boundary that can lead to wobbling in the output that is not characteristic of the exemplars or the query curve. A more global strategy for synthesis could potentially address these artifacts, but we find that the most straightforward approach of deepening the synthesis hierarchy is not effective.

Variation in output. Because the initialization to our texture synthesis process comes from piecewise-matching portions of exemplars to the query path, the statistics of the resulting texture can be skewed by the shape of the query. For example, a near perfect circular query may cause a single, similar-curvature portion of an exemplar to match repeatedly along the query, leading to repetition. Lukáč et al. [2013] introduced an efficient algorithm for matching the statistics of an exemplar set when selecting them for a linear sequence. It may be possible to adapt their algorithm for our setting where we need to take special care at the endpoints of curves and also match the rough shape of the query. More broadly, our system currently has no random component, so the same input always produces the same output. It would be nice to offer a mode suitable for *ideation* where several different variations are presented and the designer chooses among them.

Parameterization challenges. There are other general-purpose parameterization techniques ([Sun et al. 2013] and [Schmidt 2013]), that are more complex than the one introduced in Section 4.2. Additional parameterization challenges exist for our particular problem. For example, when drawing a curve around a sharp corner, branches that extend towards the curve can intersect in the corner. We believe that a more sophisticated parameterization of the path could ameliorate these artifacts. Moreover, our current framework cannot handle some forms of decorative art – those with very large features relative to the size of the strokes, or which do not have an obvious directionality. To extend the kinds of artwork that could be handled well, it would be desirable to be able to express secondary branches off the main spine. During sketching this could provide additional control over where such features appear. In addition it would be ideal if there is some mechanism whereby a user could indicate regions of the exemplars that should remain intact (such as the head of the flower in the "rose" dataset), so that artifacts shown in Figure 11c can be avoided.

Vectorization from SDF. As discussed in Section 4, many of our example styles work best when we synthesize a signed distance field (SDF) rather than binary image as output. In such cases, our current approach is to use Adobe Illustrator to vectorize from a level set in the SDF. However, the synthesized SDF contains much more data (highly redundant) than simply the values neighboring the level set, and this information might be used to address small topological problems that are difficult to repair directly from the level set. One strategy might be to use the gradient and SDF values sampled at many patches in the neighborhood of the level set to generate point and normal samples at the boundary, and then reconstruct a boundary from this (noisy) data using Poisson reconstruction [Kazhdan et al. 2006].

Stroke Intersection and Branching. When a long stroke intersects itself, the newly drawn part overwrites the previously drawn parts. The texture synthesis pipeline refines all pixels in the query stroke in the same way, including the overlap region. We can synthesize meaningful intersections for datasets that contain intersection-like structures, like "doodle" and "palm" (Figure 17a,b). In addition, we can emulate branching by starting a new stroke from the main branch of a previously synthesized stroke. The texture synthesis step merges the new structures with the existing structures, creating a branching appearance. Figure 17c shows a synthesized stroke before branching. Figure 17d demonstrates the effect of adding two branches. Figure 18 and the other decorative capital letters also demonstrate branching: the T, D, h, k, etc., are generated with two strokes, where the beginnings of the second strokes overlap the first strokes. To better handle intersections and branching for more complex datasets such as "floral" and "wings", one should design explicit synthesis procedures for the overlap regions based on exemplars that contain proper crossing and branching structures. Figure 20a shows that the current synthesis pipeline does not synthesize optimal structure in the overlap region between the two strokes of the digit "4". Figure 20b shows the first stroke before intersecting with the second stroke.

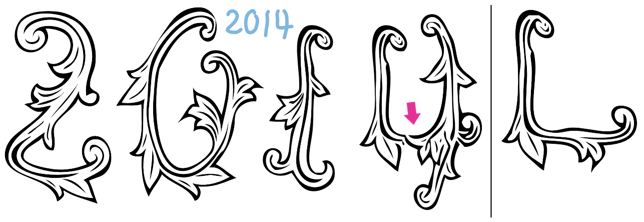


Figure 20: *Stroke Crossing. Left: the number "2014" is synthesized with the "wings" exemplars in Figure 2h. The digit "4" is drawn with two crossing strokes. Our current pipeline cannot synthesize proper crossing appearance due to the lack of crossing structures in the exemplars. Right: the first stroke of the digit "4".*

Acknowledgements

We thank the reviewers for helpful feedback, and Daichi Ito who provided the "wings" dataset. This research was sponsored in part by generous gifts from Adobe and Google.

References

AlMeraj, Z., Kaplan, C. S., and Asente, P. 2013. Patch-based geometric texture synthesis. In *Proceedings of the Symposium on Computational Aesthetics*, ACM, CAE '13.

Anderson, D., and Wood, Z. 2008. User driven two-dimensional computer-generated ornamentation. In *Proc. International Symposium on Advances in Visual Computing*.

Ando, R., and Tsuruno, R. 2010. Segmental brush synthesis with stroke images. In *Proc. Eurographics – Short papers*.

Ashikhmin, M. 2001. Synthesizing natural textures. In *Proceedings of Symposium on Interactive 3D Graphics*, ACM.

Barla, P., Breslav, S., Thollot, J., Sillion, F., and Markosian, L. 2006. Stroke pattern analysis and synthesis. In *Computer Graphics Forum (Proc. of Eurographics 2006)*.

Chen, Y.-S., Shie, J., and Chen, L.-H. 2012. A npr system for generating floral patterns based on l-system. *Bulletin of Networking, Computing, Systems, and Software 1*, 1.

Efros, A. A., and Leung, T. K. 1999. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, 1033–1038.

Hertzmann, A., Oliver, N., Curless, B., and Seitz, S. M. 2002. Curve analogies. In *Rendering Techniques*, 233–246.

Hsu, S. C., Lee, I. H. H., and Wiseman, N. E. 1993. Skeletal strokes. In *Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, ACM, 197–206.

Hurtut, T., Landes, P.-E., Thollot, J., Gousseau, Y., Drouillhet, R., and Coeurjolly, J.-F. 2009. Appearance-guided synthesis of element arrangements by example. In *Proc. of Non-Photorealistic Animation and Rendering*, ACM.

Igarashi, T., Moscovich, T., and Hughes, J. F. 2005. As-rigid-as-possible shape manipulation. In *Proc. of SIGGRAPH*.

Ijiri, T., Měch, R., Igarashi, T., and Miller, G. S. P. 2008. An example-based procedural system for element arrangement. *Comput. Graph. Forum 27*, 2, 429–436.

Kang, H., Lee, S., and Chui, C. K. 2007. Coherent line drawing. In *Proc. of Non-photorealistic Animation and Rendering*, ACM.

Kazhdan, M., Bolitho, M., and Hoppe, H. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*.

Kazi, R. H., Igarashi, T., Zhao, S., and Davis, R. 2012. Vignette: interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, ACM, 1727–1736.

Kim, M., and Shin, H. J. 2010. An example-based approach to synthesize artistic strokes using graphs. *Computer Graphics Forum 29*, 7, 2145–2152.

Landes, P.-E., Galerne, B., and Hurtut, T. 2013. A shape-aware model for discrete texture synthesis. In *Computer Graphics Forum*, vol. 32, Wiley Online Library, 67–76.

Lefebvre, S., and Hoppe, H. 2005. Parallel controllable texture synthesis. *ACM Trans. Graph. 24*, 3 (July), 777–786.

Lefebvre, S., and Hoppe, H. 2006. Appearance-space texture synthesis. In *Proc. of SIGGRAPH*, ACM, 541–548.

Leymarie, F., and Levine, M. 1992. Fast raster scan distance propagation on the discrete rectangular lattice. *CVGIP: Image Understanding 55*, 1, 84 – 94.

Lu, J., Yu, F., Finkelstein, A., and DiVerdi, S. 2012. Helpinghand: Example-based stroke stylization. *ACM Trans. Graph. 31*, 4 (July), 46:1–46:10.

Lu, J., Barnes, C., DiVerdi, S., and Finkelstein, A. 2013. Realbrush: Painting with examples of physical media. *ACM Trans. Graph. 32*, 4 (July), 117:1–117:12.

Lukáč, M., Fišer, J., Bazin, J.-C., Jamriška, O., Sorkine-Hornung, A., and Sýkora, D. 2013. Painting by feature: Texture boundaries for example-based image creation. *ACM Trans. Graph. 32*, 4 (July), 116:1–116:8.

Ma, C., Wei, L.-Y., and Tong, X. 2011. Discrete element textures. In *ACM Transactions on Graphics (TOG)*, vol. 30, ACM, 62.

Měch, R., and Miller, G. 2012. The *Deco* framework for interactive procedural modeling. *Journal of Computer Graphics Techniques (JCGT) 1*, 1 (Dec), 43–99.

Muja, M., and Lowe, D. G. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, 331–340.

Risser, E., Han, C., Dahyot, R., and Grinspun, E. 2010. Synthesizing structured image hybrids. In *Proc. of SIGGRAPH*, ACM, 85:1–85:6.

Schmidt, F. R., Toppe, E., and Cremers, D. 2009. Efficient planar graph cuts with applications in computer vision. In *IEEE Conf. Computer Vision and Pattern Recognition*, IEEE, 351–356.

Schmidt, R. 2013. Stroke parameterization. In *Computer Graphics Forum*, vol. 32, Wiley Online Library, 255–263.

Sun, Q., Zhang, L., Zhang, M., Ying, X., Xin, S.-Q., Xia, J., and He, Y. 2013. Texture brush: an interactive surface texturing interface. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, 153–160.

Wong, M. T., Zongker, D. E., and Salesin, D. H. 1998. Computer-generated floral ornament. In *Proc. of SIGGRAPH*, ACM, New York, NY, USA, SIGGRAPH '98, 423–434.

Zhou, S., Lasram, A., and Lefebvre, S. 2013. By-example synthesis of curvilinear structured patterns. *Computer Graphics Forum 32*, 2.