Painting by Feature: Texture Boundaries for Example-based Image Creation

Michal Lukáč[1]* Jakub Fišer[1] Jean-Charles Bazin[2] Ondřej Jamriška[1] Alexander Sorkine-Hornung[3] Daniel Sýkora[1]
[1]CTU in Prague, FEE [2]ETH Zurich [3]Disney Research Zurich

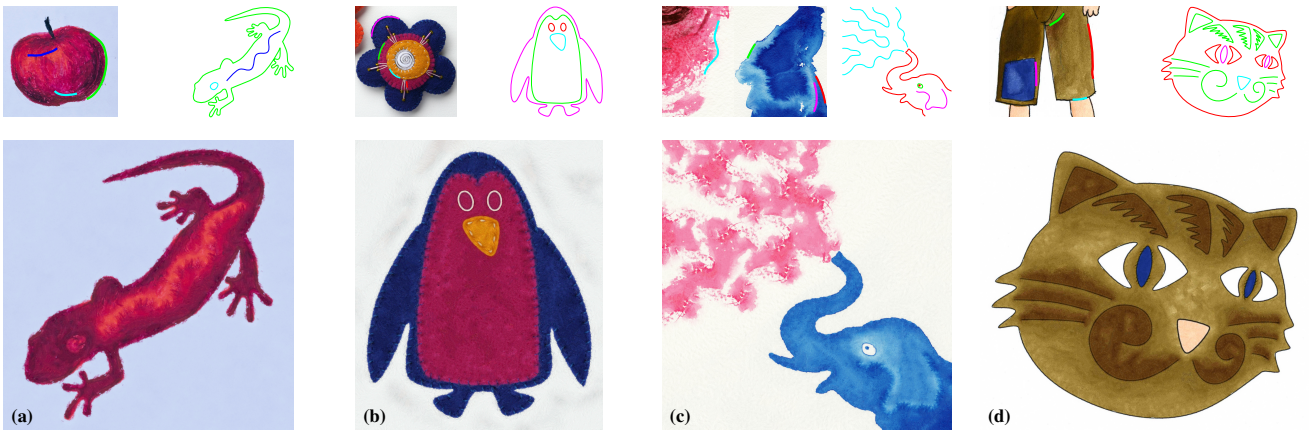

Figure 1: *Representative results generated by our proposed example-based painting framework. The user selects line features in a reference image (colored lines in the top left images, see also area features in the supplementary material) which are then immediately available as brushes for applications such as real-time painting or vector image stylization. The respective top right images depict the user's painted strokes in order to create the images in the bottom row. These demonstrate various use cases of our method: (a) complex paintings from a few input strokes, (b) painting detailed, structured boundaries, (c) watercolor, and (d) diffusion curve effects. Source credits: (a)* Sarah G *via flickr,* fzap *via OpenClipArt; (b)* Pavla Sýkorová*, clipartsy; (c)* bittbox *via flickr,* papapishu *via OpenClipArt; (d)* Anifilm*,* Pavla Sýkorová

Abstract

In this paper we propose a reinterpretation of the *brush* and the *fill* tools for digital image painting. The core idea is to provide an intuitive approach that allows users to paint in the visual style of arbitrary example images. Rather than a static library of colors, brushes, or fill patterns, we offer users entire images as their palette, from which they can select arbitrary contours or textures as their brush or fill tool in their own creations. Compared to previous example-based techniques related to the painting-by-numbers paradigm we propose a new strategy where users can generate salient texture boundaries by our randomized graph-traversal algorithm and apply a content-aware fill to transfer textures into the delimited regions. This workflow allows users of our system to intuitively create visually appealing images that better preserve the visual richness and fluidity of arbitrary example images. We demonstrate the potential of our approach in various applications including interactive image creation, editing and vector image stylization.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation I.3.4 [Computer Graphics]: Graphics Utilities—Paint systems I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

Keywords: example-based painting, stroke synthesis, painting-by-numbers, vector image stylization, non-photorealistic rendering

Links: DL PDF WEB

*e-mail:michal.lukac@fel.cvut.cz

1 Introduction

Strokes and lines are the most elementary primitives in painting, both digital and physical. The concept of drawing shapes by first sketching and developing object outlines seems to be so natural and intuitive that small children employ it just as artists and designers. Any existing image editor implements the basic *pencil* and/or *brush* tools, and various attempts have been made to enhance their expressive power, such as the calligraphic brush or the textured stroke. Similarly, vector-based image editors use *paths* as their most fundamental primitive for defining object boundaries.

Despite their importance for sketching the essential structures in an image, basic brush- or path-based tools are generally less suitable for creating a clean, richly textured image such as the ones shown in Figure 1. Researchers have long been aware of this gap between a sketch and production quality artwork, and proposed various ideas for converting simple sketches into richer and more expressive images [Ashikhmin 2001; Hertzmann et al. 2001; Ritter et al. 2006; Orzan et al. 2008].

Unfortunately, existing approaches often face difficulties when synthesizing images with significant structure, as the underlying algorithms generally focus on synthesizing 2D textured areas, without explicitly enforcing consistency to the boundaries of a shape. Due to the sensitivity of human vision to the contours of a shape [De-

Figure 2: *Comparison of results from different approaches given the input picture (a) as the reference image or* feature palette. *Result of (b) Image Analogies [Hertzmann et al. 2001], (c) Painting with Texture [Ritter et al. 2006], (d) Synthesizing Natural Textures [Ashikhmin 2001], (e) our approach. Source credit:* Wednesday Elf – Mountainside Crochet *via flickr*

Carlo et al. 2003], such artifacts become immediately apparent (see comparison in Figure 2).

This paper addresses these issues by modeling an image as a set of two classes of features. The first class corresponds to 1D *line features*, such as important contours, boundaries of textured regions, or salient strokes which are used to define the basic structure of the image. The second class corresponds to 2D *area features*, which represent regions filled with a nearly-stationary texture. For defining the visual style of an image, we introduce the metaphor of a *feature palette*, which is simply one or more example images of a desired visual style, in which the user selects line and area features with which to paint.

Our main technical contribution is a novel algorithm for interactive synthesis of line features (*brush* tool) which utilizes a randomized graph traversal mechanism with multi-level blending to seamlessly synthesize long, non-repetitive, textured strokes sampled from shorter exemplars located in the input image. For the transfer of area features (*fill* tool) we use a state-of-the-art texture synthesis algorithm [Wexler et al. 2007] which avoids visible discontinuities between painted line features and textured areas while preserving the richness of the original exemplar. Both tools provide immediate real-time feedback, making their use as intuitive and easy as an ordinary brush or fill tool. Creating complex, visually appealing drawings with our system requires similar effort as creating a simple contour sketch in standard drawing systems.

2 Related Work

One of the first works for example-based visual style transfer between images is the Image Analogies approach by Hertzmann et al. [2001]. They discuss the possibility of example-based painting using the texture-by-numbers paradigm where an input image is first segmented into multiple regions denoted by color labels, and then these labels are painted to form a new segmentation from which an output image is generated using their texture synthesis algorithm. While this approach provides a high degree of freedom in defining the output result, it is not clear how to support the concept of 1D structure elements such as contours. Moreover, the algorithm complexity prohibits an interactive implementation. A representative result is shown in Figure 2b.

Ritter et al. [2006] further extended Hertzmann et al.'s framework and created a nearly interactive texture-by-numbers painting program where boundary pixels are refined automatically thanks to an additional energy term which takes similarity of source and target boundaries into account. However, a key limitation is the lack of user control in the boundary forming process and the technique is inherently 2D, i.e., it does not preserve 1D structure of more complex boundaries. Although pixels are transferred from locations with a similar boundary shape, there is no guarantee that they will produce a 1D, visually continuous strip since the source pixels can be located on different parts of the boundary. Hence, the method produces convincing results only for textures which have a nearly constant cross-section profile along the boundary, producing artifacts otherwise (see Figures 2c and 9).

Similar considerations apply to other types of texture synthesis algorithms [Ashikhmin 2001; Efros and Freeman 2001; Kwatra et al. 2003] which partially also provide support for user constraints [Kwatra et al. 2005; Lefebvre and Hoppe 2005], or to matching based image manipulation and morphing techniques [Barnes et al. 2009; Shechtman et al. 2010; Darabi et al. 2012; Yücer et al. 2012]. All these methods provide very flexible and powerful tools for filling or transforming image areas with plausible and visually rich textures, but at the same time they are inherently 2D without support for user-controlled, real-time 1D structure transfer from a reference image. See Figure 2d for an exemplary result with the method of Ashikhmin et al. [2001]. The synthesis step of the above result took 130 seconds, whereas our approach provides instantaneous feedback.

Recently, other example-based content generation techniques have been proposed, which create new images from a user-provided set of examples [Risser et al. 2010; Assa and Cohen-Or 2012]. However, these techniques are non-interactive, global approaches which specialize in rapid generation of a large number of variations of the input image. Currently, the only way to influence this process is by providing a different choice of input images.

Sun et al. [2005] demonstrated the benefit of giving the user control over structural features in the context of image inpainting. They apply a constrained patch-based synthesis on the user-provided line features and then perform inpainting on the remaining areas that is consistent with the previously synthesized structures. A restriction of this approach is, however, that the employed energy minimization provides no guarantees that the global scale visual appearance of the synthesized line feature is consistent with its appearance in the respective source image. In texture synthesis, this problem is generally avoided by multi-scale synthesis, but this is not feasible for linear features, as they eventually disappear on lower resolutions. Using basic energy optimization without a sufficiently expressive model of a feature's global scale, artifacts are often perceptible as a periodic repetition of a pattern along the output path. On a related note, another example for the benefits of contour-based editing is the work of Fang et al. [2007] for detail preserving shape deformation in images.

For vector graphics editing Orzan et al. [2008] presented a technique for creating smooth color transitions between spline paths using Poisson interpolation. Due to the purely vector-based representation this approach is not suitable for style and texture transfer

between images. McCann and Pollard [2008] broke new ground by introducing a set of gradient-painting tools, designed to be fully interactive and directly controlled by the user. Notably, they introduce an edge brush tool which allows the user to select a path in a source image and map it to a path in the result using gradient-domain blending. Their approach, however, targets image editing, and their simple copying procedure offers no variation during feature synthesis, resulting in clearly visible periodicity when the output path is much longer than the source path of the respective feature. In our approach, we utilize a workflow similar to theirs for image creation. However, we introduce a generative line feature model to enable an indefinite extension of a source path without such artifacts.

Related to our algorithm for feature transfer is the work on video textures [Schödl et al. 2000]. They developed a feature model capable of extending a video in the temporal domain, where the frames are represented as graph nodes and the edge weights represent a measure of similarity between two frames. Thanks to this representation a permutation of video frames can be expressed as a low-cost traversal through the graph. Their approach served as an inspiration for our generative model for line feature synthesis. However, similar to the work of Sun et al. [2005] and McCann and Pollard [2008], a direct application of their loop-based synthesis algorithm would result in obvious periodic artifacts. Part of our contribution is a synthesis algorithm that resolves these issues.

3 Our Approach

As briefly outlined in the introduction, our proposed approach is based on three central concepts. The first two of them are the two different types of features and their corresponding tools:

- A **Line Feature** is a one-dimensional feature representing an arbitrary curvilinear structure, such as an edge or contour in an image. It typically represents a boundary between two textured regions, but can also represent other structures such as open curves. The corresponding tool for painting line features is the *Brush* tool.
- An **Area Feature** is a two-dimensional image region which has the semantics of a stationary texture rather than that of a one-dimensional structural element. It typically represents the interior of a region, but can also be a changing gradient or any other area sample. Its corresponding tool is the *Fill* tool.

Both tools consist of two parts, namely a *selection* component which allows the user to define a desired line or area feature, and a *synthesis* component which efficiently renders the corresponding output according to the user's drawing.

The third central concept is the **Feature Palette** and it concerns the feature selection process. Rather than requiring the user to define features in a cumbersome manual way, the basic idea is to regard an arbitrary set of input images as a palette for painting. The user may simply pick one or more input images that reflect a desired visual style, and our algorithm provides the selection tools to intuitively and efficiently define line features as well as area features. Hence, any image can be used as a palette for defining features.

These concepts are fundamentally different from merely building a static database of strokes and fill textures, as commonly done in vector image editors. In our process, the reference image(s) used as the feature palette permit effortless definition of a dynamically changing library of brushes and textures on-the-fly. This facilitates the replication of the desired visual characteristics of the reference images in one's own creation. The user directly benefits from the rich visual details that are typically present in paintings, drawings, or photographs. Just as a painter can efficiently mix colors on a physical color palette, our concept allows the user to intuitively and interactively modify and refine a feature with instant feedback while painting.

In the following section we describe how the respective selection and synthesis components of both the brush and the fill tools for line and area features are implemented.

3.1 Brush

Given a feature palette in the form of one or more input images, selecting a line feature such as an object contour requires the user to simply draw a path (the width of which can be manually adjusted) approximately along the desired feature. Since precise drawing of such a path would be tedious, our algorithm supports an assisted selection that refines the user's approximate path and aligns the selection closely to the actual line feature in the image. We found that a relatively simple gradient-based approach is reasonable in order to provide an active support for the user at a sufficient accuracy for our algorithm, hence we based our path selection on an Active Contours approach [Kass et al. 1988]. A considerable advantage of this approach is that it runs in real-time and gives an instant result, which is an important requirement for a responsive and intuitive user interface.

Once the user has defined a path over a line feature we require a real-time algorithm that synthesizes a corresponding line feature in the output image as the user paints. In the field of texture synthesis it has long been understood that synthesizing a larger texture simply by tiling a smaller example texture produces sub-optimal results. Thus, in order to avoid periodicity, some texture synthesis techniques [Lefebvre and Hoppe 2005] deliberately introduce a degree of randomness instead of tiling the texture. Likewise, our goal is to reproduce the local visual characteristics, i.e., the *look and feel* of a line feature, without introducing noticeable artifacts on a larger scale. Approaches such as [McCann and Pollard 2008] exhibit periodicity and cannot explicitly avoid visible discontinuities when stroke endpoints meet. We present a new algorithm for randomized line feature synthesis based on a graph model of the input feature to resolve such issues.

As line features such as object contours are one-dimensional and oriented, we found the graph formalism introduced by Schödl et al. [2000] for manipulating video over time to be an excellent basis for feature synthesis. We sample an input path at equidistant points and consider the direction of the feature to be equal to the direction of the user's stroke. Treating these samples as graph nodes and using the direction of the feature for ordering, we define a complete oriented graph, where the weight $w(i, j)$ of an oriented edge between nodes i and j is given by a dissimilarity measure. Specifically, we define $w(i, j) = SSD(p(i), p(j - 1))$, where $p(i)$ is a square image patch centered on the i^{th} sample and aligned with the path direction, and SSD denotes the sum of squared differences between patches (see Figure 4). We use $SSD(p(i), p(j - 1))$ rather than $SSD(p(i), p(j))$ because traversing to a consecutive sample on the original feature should be free, and thus $w(i, i + 1)$ should be equal to zero. The size of the patch is a user-configurable parameter which is intuitively equivalent to brush width and can be adjusted interactively. A walk in such a graph represents a permutation (with repetition) of input samples, which, if transferred to equidistant samples on a different path and rendered, would yield a variation of the source feature. The total cost of this walk is then representative for the amount of discontinuities in the output.

Given this representation the main concern is how exactly to generate a walk through this graph to satisfy all of our requirements and

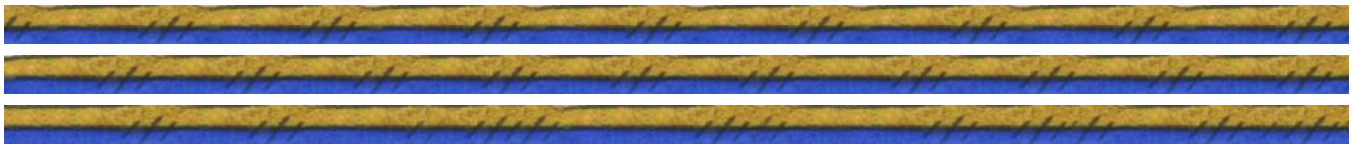

Figure 3: *A comparison of different walk synthesis approaches. Top to bottom: looping, dynamic programming and our randomized graph traversal. Note that finding the cheapest walk of a given length by dynamic programming provides the optimal result with respect to discontinuity cost, but it does so by finding the cheapest loop in the graph and thus introduces periodicity. A randomized approach, though not optimal with respect to the global cost, provides a more natural, varied look without noticeable visual discontinuities.*

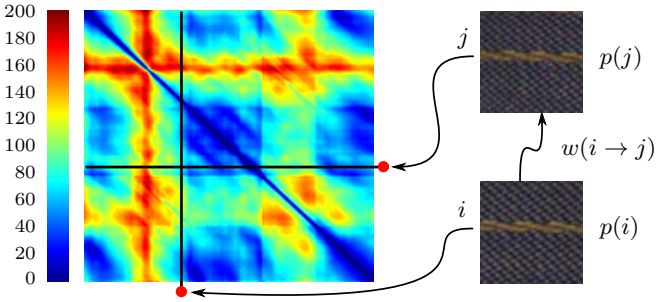

Figure 4: *A feature graph in matrix form, with color coded weights of the similarity between two patches $p(i)$ and $p(j)$. Blue represents a low matching error and hence a high similarity, while red represents a low patch similarity. Note that the matrix is not square, as there can be no edges into node 0, nor is it symmetric, as $w(i,j) = SSD(i,j-1)$ rather than $SSD(i,j)$.*

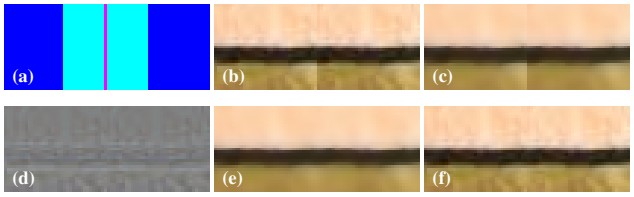

Figure 5: *An illustration of blending on jumps. (a) A colorized indication of the jump strip and blending area. (b) Synthesized feature without blending. (c) Synthesized base layer without blending. (d) Synthesized detail layer. (e) Synthesized base layer with extrapolation-blending. (f) Synthesized base layer with extrapolation blending and added detail.*

constraints. A potential solution could be to employ path optimization techniques that are capable of minimizing discontinuity along the entire path, e.g., by dynamic programming or belief propagation [Sun et al. 2005]. However, in our application such an approach is not suitable for several reasons. First, if the desired length of the walk is long compared to the input path provided by the user, the optimal solution degenerates to simply cycling the cheapest loop, as illustrated in Figure 3. Furthermore, it is not guaranteed that, when the user changes the desired length of the walk, the new optimal solution will have the previous one as a prefix. However, when a user draws a path with the brush tool, this corresponds to a permanent modification of the walk length. Failing to consider this inevitably causes the output stroke to flicker during interactive painting, as it would have to be re-rendered to remain optimal under changing stroke length (see supplementary video). In contrast, a random walk, such as the one employed by Schödl et al. [2000], can generate a randomized solution, but provides no guarantees on the discontinuity cost and assigns a non-zero probability to the highest-discontinuity edges.

So rather than finding a globally optimal walk of a given length or randomly traversing the graph, our synthesis algorithm generates a randomized, low-discontinuity walk of *at least* a given length. Instead of randomly picking the next outgoing edge to traverse, we pick the next goal node to visit. To minimize the discontinuity cost, we do not automatically traverse the edge connecting the two nodes, but instead apply Dijkstra's algorithm [Dijkstra 1959] to rapidly find the optimal path connecting the two nodes and append this path to the current walk. We repeat this process, starting from the previous goal node, until the desired length is achieved.

Although the length of the path, as measured in the number of nodes traversed, is not easily predictable, we may simply continue connecting paths until we obtain a walk of at least the desired length, picking each next goal node randomly. This ensures that any visual element present in the input will be rendered from time to time, without enforcing any particular ordering and keeping the visible discontinuities to a reasonable minimum. One could also conceivably bias the walk to a certain sub-portion of the feature by a more sophisticated selection of goal nodes, although we have not found this necessary for our application.

In order to render the selected feature onto a user-provided path, we sample the output path at equidistant intervals, generate a walk and assign to each of the output samples an input sample represented by the node at the given position in the walk. Having thus established correspondences between output and input samples, we use a simple piecewise-rigid mapping based on the Voronoi diagram of the output samples to determine the output pixel values for pixels within the stroke width of the sketched path. The process is illustrated in Figure 6.

Discontinuities in the synthesized path may occur when an edge with greater cost has to be traversed. To mask these without sacrificing fine details, we employ a decomposition-blending approach inspired by Burt and Adelson [1983]. Whenever consecutive output samples are created by a jump between non-consecutive input samples we perform local blending. To that end, we use a bilateral filter to decompose the source image into a base layer and a detail layer, as proposed by Durand and Dorsey [2002]. We then extrapolate the base layer values for each of the consecutive sub-sequences around the jump point and blend them, re-applying detail immediately thereafter, as illustrated in Figure 5.

3.2 Fill

The second tool, which we provide for efficient filling of image areas between line features, is essentially a paint bucket tool as present in all common image editors. However, analogous to the brush tool, our concept is to provide a fill tool that fills image areas with texture selected by the user from the image serving as the feature palette, maintaining consistency with the existing line features. Selection of area features is more straightforward than for line features as no specific structural properties have to be observed

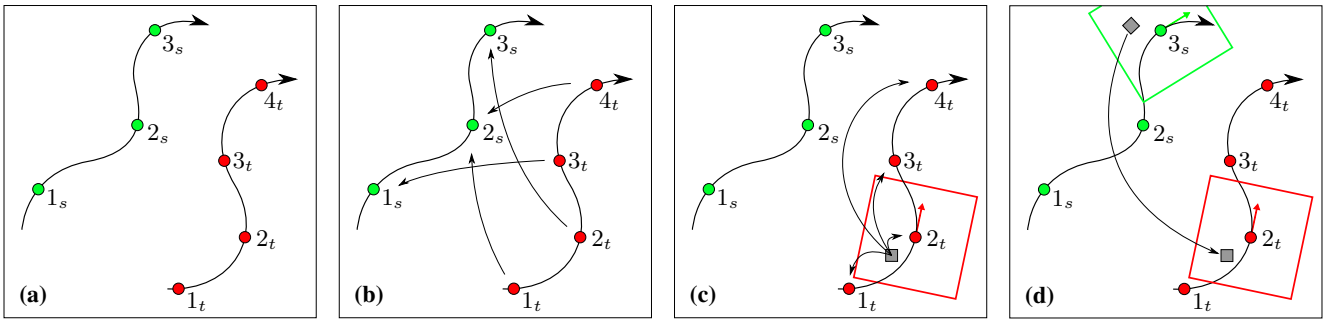

Figure 6: *Line feature mapping process. (a) Both the source path and the target path are sampled (respectively, the green and the red circles) at equal intervals. (b) We map the walk to the target path, determining for each target sample the corresponding source sample. (c) We determine the color of a pixel (gray square) in the target by finding the nearest target sample and (d) taking the value at the same relative position in the corresponding source patch (colored squares, arrows denote patch orientation).*

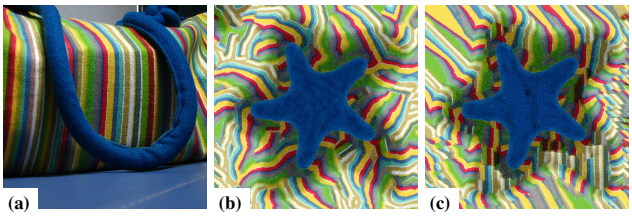

Figure 7: *Given a source image (a), results of the fill tool with (b) and without (c) pre-rotation. Note how the orientation of the stripes on the synthesized linear features guides the orientation of the generated texture in (b). When rotation is not taken into account visible artifacts might appear, as shown in (c). Source credit:* Hrishikesh Premkumar *via flickr*

during the selection. Hence, in our implementation the user can simply specify any arbitrary region in an image and use it as an area feature.

Unlike a simple flood fill tool, we have to consider the boundary conditions of the region being filled to avoid inconsistencies with existing image content like line features. Thus, rather than formulating the task of the fill tool as a simple texture synthesis problem, we treat this step as a content-aware fill which respects boundaries of the filled area and implement the method of Wexler et al. [2007] in combination with PatchMatch [Barnes et al. 2009] for fast nearest-neighbor search.

A multi-scale optimization approach [Wexler et al. 2007] is critical for our purpose, since the areas to be filled span over the majority of the canvas and treating the fill synthesis locally would lead to undesired artifacts and would furthermore be prone to introducing unwanted repetitions in the generated texture. To improve the quality and visual appearance of the result, we also perform the nearest-neighbor search across a limited range of rotations (see Figure 7). However, rather than computing the transformed source patches on the fly, we found that the combination of pre-rotating the source selection and performing the nearest-neighbor search using only translations to be significantly faster, which is crucial for instant results and direct visual feedback to the user.

4 Applications and Results

An overview of our image creation workflow is illustrated in Figure 8. Due to its generality, our approach can be utilized in several applications. One of our primary applications is vector image styl-

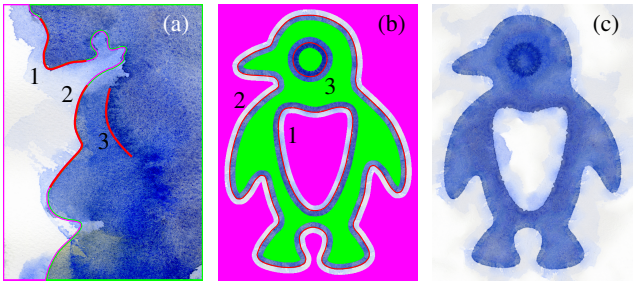

Figure 8: *Image creation workflow overview. (a) Annotated source image: two area features delimited by the pink and green outlines, and three line features indicated by the red curves and the numbers. (b) Line feature synthesis along user-specified paths from the corresponding numbered line features of (a). The pink and green areas represent the parts to be filled in by the corresponding area features of (a). (c) Final result after texture transfer by the fill tool. Source credits:* Alessandro Andreuccetti *via deviantART,* mrjive *via OpenClipArt*

ization: the user selects line and area features in an example image and then assigns them to paths and fill shapes of a vector image, respectively. Figure 1 shows representative results created using our framework. Note that, unlike previous texture-by-numbers approaches, we can handle open paths and strokes. The result images are visually consistent on a local as well as a global scale and represent the visual style of the respective reference image (see comparison with previous texture-by-numbers approaches in Figures 2 and 9). Figure 1a illustrates that even a simple vector image composed of a very limited number of input strokes (three in this example) can lead to richly textured image. In Figure 1b please note the quality of the knitting stitches generated by our line feature synthesis at the boundaries of the different pieces of the penguin. Our approach can also be applied for watercolor painting, as shown in Figure 1c. This challenging task usually requires sophisticated techniques [Curtis et al. 1997; DiVerdi et al. 2013], whereas our approach can solve it without additional specific tools.

An interesting characteristic of our approach is that when paths incident to a region have different "inside colors", the region inpainting algorithm attempts to diffuse the difference between their colors over the intermediate region, producing results similar to Diffusion Curves [Orzan et al. 2008], with no additional creative effort on the artist's part. For example, in Figure 1d, note the diffusion effect along the cat's whiskers and mustache (see also a more complex example of texture transitions in Figure 15). A comparison

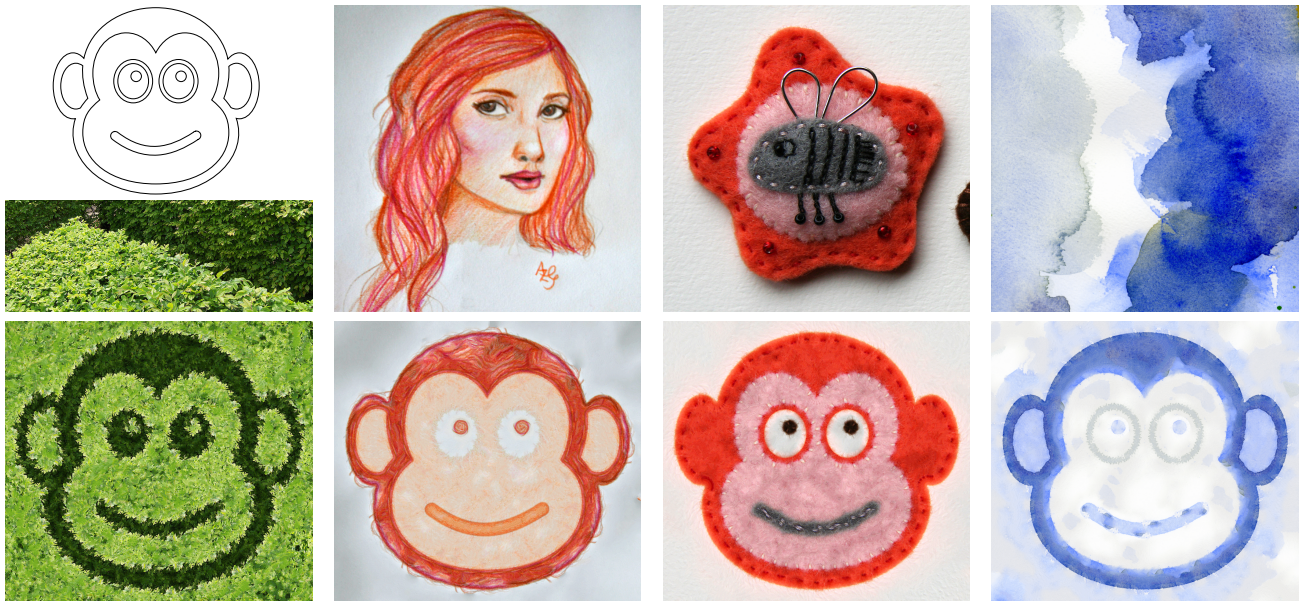

Figure 11: *Example of different stylizations with the same stroke input (top left). Source credits (left–right):* Martouf *via OpenClipArt;* Joe Shlabotnik *via flickr;* Andrea Garcia *via flickr;* Pavla Sýkorová*;* Alessandro Andreuccetti *via deviantART*

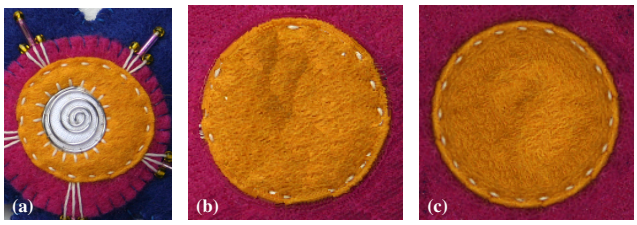

Figure 9: *Comparison with Painting with Texture [Ritter et al. 2006]: (a) source image, (b) result of Painting with Texture, (c) our approach. Note how our method better preserves the important visual charcteristics of the source image thanks to explicit treatment of line and area features. Source credit:* Pavla Sýkorová

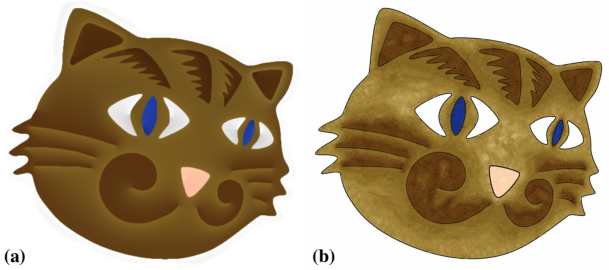

Figure 10: *Comparison of vector stylization techniques. (a) Result of Diffusion Curves [Orzan et al. 2008]. (b) Our result. Source credit:* Pavla Sýkorová

with Diffusion Curves is available in Figure 10. Both approaches took a comparable amount of artistic effort to produce, however, our method enables the transfer of the visual style and richness in terms of texture from a reference image (see Figure 1d). Additional examples of different stylizations given a single user-drawn sketch are shown in Figure 11. In this stylization scenario, the user simply needs to select the line and area features they would like to incorporate in the result image.

Another exciting application is interactive example-based painting. We have developed a painting program which implements just the two tools we introduce in this paper, deliberately leaving out extra functionality of sophisticated image editors, in order to show that our painting-by-feature approach alone enables the creation of appealing results. In our paint program the user may select features from source images and transfer them to manually indicated positions, using the same mode of interaction as with the common brush tool and fill tool known from consumer image editors. A representative interaction with our application is shown in Figure 8 as well as in the supplementary video. It demonstrates that our application is simple to use, and that the user can create and edit paintings interactively with instantaneous feedback. Visually appealing results can be created in a short time, typical editing session for the results shown here were in the order of 1–3 minutes depending on the level of detail the user wishes to incorporate. Additional results are shown in Figure 12. Further potential applications of our method include image editing scenarios such as inpainting. We refer readers to the supplementary material for a representative result.

4.1 Limitations

While our approach has proven suitable for its intended applications and produces high quality results, some limitations do apply.

We do not explicitly handle possible intersections and junctions of line features which may produce visually disturbing transitions in the output image (see Figure 13). These artifacts can partially be alleviated by proper reordering of strokes or using some sort of blending, e.g., min/max-blending (`GL_MIN` or `GL_MAX` blending mode in OpenGL) or decomposition-blending described in Section 3.1. Nevertheless, in future work one may consider to incorporate support for intersections and junctions directly into the synthesis algorithm to automatically produce seamless output.

We also deliberately do not check for consistency of the selected features to give the user full control and artistic freedom. As a consequence the user can select a line feature that is not aligned

Figure 12: *Additional results of interactive example-based painting. Left: source image, Right: result obtained by our approach. Source credits:* Paul Cézanne *(top);* Vincent Van Gogh, Kaldari *via Wikimedia Commons (bottom)*

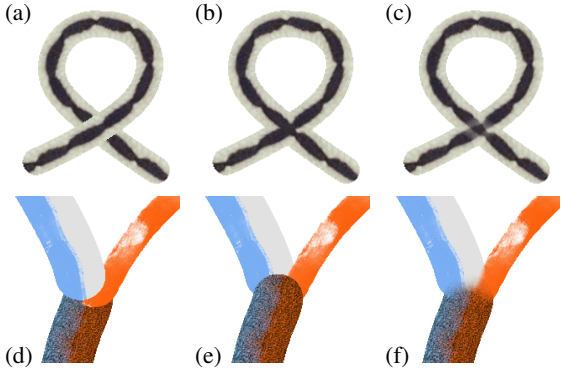

Figure 13: *Self intersections of a brush stroke (a) or junctions of multiple linear features (d) may produce visible discontinuities. These can be alleviated by proper stroke reordering (e), min/max-blending (b) or decomposition-blending (c,f).*

Figure 14: *The user can select linear features which may not be fully in line with our requirements on 1D structure (red and green curves in the left inset), potentially producing unintended results: the green curve generates white vertical sewings (top) and the red curve yileds a completely erronous result (bottom).*

with an actual linear structure in the input image or one that is composed of incompatible structural elements. In these cases our algorithm might produce visually displeasing transitions (see Figure 14). Similarly, selection of an area feature which is incompatible with already drawn line features may also lead to an erroneous result (see Figure 15). An alternative scenario to investigate in future work is that the feature selection process could be assisted by interactive image segmentation tools [Li et al. 2004], or by identification and removal of inconsistent sub-elements, e.g., by texture analysis [Todorovic and Ahuja 2009].

To prevent periodicity, our approach runs a randomized graph walk (see Section 3.1). The disadvantage is that variations might exist between results for a same source image and input sketch. Additional results available in the supplementary video (elephant sequence) show that these variations are very limited, on a local level and visually consistent with the other results on a global level, which is sufficient for our target applications.

5 Conclusion and Future Work

We have presented a feature-based image creation model, useful for vector image stylization as well as manual image creation and image editing. Our flexible example-based stylization approach blurs the traditional border between the vector- and pixel-worlds, allowing us to create and manipulate images while preserving the visual richness of a chosen artistic style. We eagerly anticipate the new possibilities in artwork creation that this approach opens to artists, and are curious about the results which may be achieved by combining this simple, yet powerful basic approach with other existing creation and editing tools.

An interesting direction for future work is the automation of the entire process of vector image stylization. This could be achieved by automatically detecting features in a source image and assigning them to paths and regions of a vector image based, e.g., on similarity of fill and stroke colors to the colors in the feature.

We could also modify our algorithm to automatically synthesize the fill for areas between user-defined curves while they are being drawn, producing an example-based variant of the Diffusion Curves by [Orzan et al. 2008]. However, even though line feature synthesis is fast enough for interactive editing, this would require a real-time fill synthesis algorithm (even with PatchMatch, [Wexler et al. 2007] is too slow to permit this) and a similarly rapid image analysis tool, which would determine source areas for output regions based on the input strokes and other features present in the input in order to keep the output visually consistent.

Similarly, used in conjunction with an automated image decomposition algorithm such as [Guo et al. 2007], one could reduce an input image into a sketch representation and a representative subset of features in order to re-synthesize the original image at a later time. Thus one could facilitate image compression with a configurable loss of information (see supplementary material for an example of image decomposition based on our method).

For the brush tool, it might be possible to investigate whether the input line feature contains any underlying dimensionality (such as texture orientation), and modify our formulation so that the output is constrained by this underlying parameter, determined, e.g., by pen pressure. Similarly, the introduction of control maps for area features could play a role for synthesis.

Acknowledgements

We would like to thank Gioacchino Noris and all anonymous reviewers for their constructive comments. This research has been supported by the Technology Agency of the Czech Republic under the research program TE01020415 (V3C – Visual Computing Competence Center) and partially by the Grant Agency of the Czech Technical University in Prague, grant No. SGS13/214/OHK3/3T/13 (Research of Progressive Computer Graphics Methods).

Figure 15: *An example output of our fill tool when synthesized strokes contain incompatible structures (a). When the whole source image (b) is taken as an example (red and blue rectangles) the fill tool produces pleasing transitions (c). However, when an incompatible portion (red and blue areas) of the source image is selected (d), the algorithm can produce erroneous results (e). Source credit:* Carl Wycoff *via flickr*

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Proceedings of Symposium on Interactive 3D graphics*, 217–226.

ASSA, J., AND COHEN-OR, D. 2012. More of the same: Synthesizing a variety by structural layering. *Computers & Graphics 36*, 4, 250–256.

BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. PatchMatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics 28*, 3, 24:1–24:11.

BURT, J. R., AND ADELSON, E. H. 1983. A multiresolution spline with application to image mosaics. *ACM Transactions on Graphics 2*, 4, 217–236.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-generated watercolor. In *Proceedings of SIGGRAPH 97*, 421–430.

DARABI, S., SHECHTMAN, E., BARNES, C., GOLDMAN, D. B., AND SEN, P. 2012. Image melding: Combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics 31*, 4, 82:1–82:10.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3, 848–855.

DIJKSTRA, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 5, 269–271.

DIVERDI, S., KRISHNASWAMY, A., MECH, R., AND ITO, D. 2013. Painting with polygons: A procedural watercolor engine. *IEEE Transactions on Visualization and Computer Graphics 19*, 5, 723–735.

DURAND, F., AND DORSEY, J. 2002. Fast bilateral filtering for the display of high-dynamic-range images. *ACM Transactions on Graphics 21*, 3, 257–266.

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

FANG, H., AND HART, J. C. 2007. Detail preserving shape deformation in image editing. *ACM Transactions on Graphics 26*, 3, 12:1–12:5.

GUO, C.-E., ZHU, S.-C., AND WU, Y. N. 2007. Primal sketch: Integrating structure and texture. *Computer Vision and Image Understanding 106*, 1, 5–19.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of SIGGRAPH 2001*, 327–340.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1988. Snakes: Active contour models. *International Journal of Computer Vision 1*, 4, 321–331.

KWATRA, V., SCHÖDL, A., ESSA, I. A., TURK, G., AND BOBICK, A. F. 2003. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics 22*, 3, 277–286.

KWATRA, V., ESSA, I. A., BOBICK, A. F., AND KWATRA, N. 2005. Texture optimization for example-based synthesis. *ACM Transactions on Graphics 24*, 3, 795–802.

LEFEBVRE, S., AND HOPPE, H. 2005. Parallel controllable texture synthesis. *ACM Transactions on Graphics 24*, 3, 777–786.

LI, Y., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2004. Lazy snapping. *ACM Transactions on Graphics 23*, 3, 303–308.

MCCANN, J., AND POLLARD, N. S. 2008. Real-time gradient-domain painting. *ACM Transactions on Graphics 27*, 3, 93:1–93:7.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. *ACM Transactions on Graphics 27*, 3, 92:1–92:8.

RISSER, E., HAN, C., DAHYOT, R., AND GRINSPUN, E. 2010. Synthesizing structured image hybrids. *ACM Transactions on Graphics 29*, 4, 85:1–85:6.

RITTER, L., LI, W., CURLESS, B., AGRAWALA, M., AND SALESIN, D. 2006. Painting with texture. In *Proceedings of Eurographics Symposium on Rendering*, 371–376.

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

SHECHTMAN, E., RAV-ACHA, A., IRANI, M., AND SEITZ, S. M. 2010. Regenerative morphing. In *IEEE Conference on Computer Vision and Pattern Recognition*, 615–622.

SUN, J., YUAN, L., JIA, J., AND SHUM, H.-Y. 2005. Image completion with structure propagation. *ACM Transactions on Graphics 24*, 3, 861–868.

TODOROVIC, S., AND AHUJA, N. 2009. Texel-based texture segmentation. In *IEEE International Conference on Computer Vision*, 841–848.

WEXLER, Y., SHECHTMAN, E., AND IRANI, M. 2007. Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 3, 463–476.

YÜCER, K., JACOBSON, A., HORNUNG, A., AND SORKINE, O. 2012. Transfusive image manipulation. *ACM Transactions on Graphics 31*, 6, 176:1–176:9.