Example-based Rendering of Hand-drawn Content

Ing. Jakub Fišer

A dissertation submitted to
the Faculty of Electrical Engineering, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy.

Ph.D. programme: Electrical Engineering and Information Technology
Branch of study: Information Science and Computer Engineering

Supervisor: doc. Ing. Daniel Sýkora, Ph.D.

February 2017
The research presented in this thesis was conducted in collaboration with, and supported by, Adobe Research and has further been supported by the Technology Agency of the Czech Republic under research program TE01020415 (V3C – Visual Computing Competence Center), by Czech Science Foundation under research program P202/12/2413 (OPALIS), by the Grant Agency of the Czech Technical University in Prague, grants No. SGS13/214/OHK3/3T/13 (Research of Progressive Computer Graphics Methods) and No. SGS16/237/OHK3/3T/13 (Research of Modern Computer Graphics Methods), and by SoMoPro II grant (financial contributions from the EU 7 FP People Programme (Marie Curie Actions), REA 291782, and from the South Moravian Region).
Abstract
In this dissertation, we focus on the capabilities of example-based synthesis with a particular interest in hand-drawn appearance. We propose several approaches for creating new digital imagery from examples without making any domain-specific assumptions and being media-agnostic. New methods for content-creation and non-photorealistic stylization are presented together with examples of their practical usage in contemporary digital production pipelines.

Keywords
Computer graphics, example-based, hand-drawn, non-photorealistic rendering, digital art, stylization, texture synthesis

Acknowledgements
I would like to express my deepest thanks to my supervisor Daniel Sýkora, who has been the best advisor and friend to guide me through the pitfalls and delights of doctoral studies. To my colleague, Ondřej Jámiška, for being an inspiration to me in many ways. And to all my co-authors and collaborators that helped me in my research.

Most of all, however, I want to thank my wife Zdeňka and my parents, for their endless support and love.
Abstrakt

Tato práce se zabývá problematikou tzv. example-based syntézy obrazu (tj. neparametrické syntézy založené na příkladu), zejména pak syntézy imitující vzhled ruční kresby. Práce diskutuje metody pro tvorbu digitálního obsahu dle příkladů bez nutnosti omezení na konkrétní aplikační doménu nebo výtvarné médium. Jsou prezentovány a diskutovány nové metody nefotorealistické stylizace společně s ukázkami jejich praktického použití v rámci stávajících procesů tvorby digitálního obsahu.

Klíčová slova

Počítačová grafika, example-based, založeno na příkladu, ručně kresleno, nefotorealistické vykreslování, digitální umění, stylizace, syntéza textur
## Contents

1 Introduction 1
   1.1 Motivation 1
   1.2 Goal 2
   1.3 Contribution 2
   1.4 Structure of the Work 5

2 Previous Work 7
   2.1 Non-photorealistic Rendering 7
   2.2 General Example-based Image and Video Synthesis 12

3 Painting by Feature 17

4 Color Me Noisy 31

5 LazyFluids 47

6 Brushables 67

7 ShipShape 85

8 StyLit 107

9 Conclusion 127
   9.1 Summary 127
   9.2 Concurrent and Future Work 128

References 131

A Author’s Publications 143

B Authorship Contribution Statement 145
1 Introduction

1.1 Motivation

Figure 1.1: Non-photorealistic rendering. The well-known “Utah teapot” model (a) is rendered in a style of wax painting (b). The aim of the NPR area as a whole is to make the result indistinguishable from a real hand-painted image.

Computer graphics has undergone tremendous development in the last two decades. Both emergence of new algorithms and the unprecedented performance boost of computer hardware gave the contemporary content creators capabilities to produce physically-based photorealistic imagery or stylized content in real-time or at least at interactive frame rates. This has posed the assets creation as the major bottleneck in the creative pipeline. This is even more true in the case of the traditional hand-created animation and graphics, where the input must be crafted almost entirely by the artist alone.

The necessity to create or assemble every single frame by hand has pushed the traditional animation techniques, such as stop-motion or classical hand-drawn animation, to the sideline mostly due to their inherent time demands. This does not mean that today’s computer graphics capabilities are not used in these fields. However, one can usually easily tell the computer-assisted content. For instance, an artist could create a fully-polished 2D figure in a reference pose and use the as-rigid-as-possible tools to create a longer animation sequence in a very short amount of time, but the result would lack the richness and variety present in the traditional “hand-polished” artwork.

The non-photorealistic rendering (NPR) methods, a relatively new field of computer graphics that we describe more closely in Section 2.1, let the artists dispatch a great amount of their repetitive work tasks to computers but for the price of losing the absolute control over the whole creative process. The portion of the automated assistance provided by NPR algorithm varies, depending on the particular method and the occurrence of the task within the content creation pipeline, with more precise control usually required in early stages. Despite the significant amount of interesting NPR methods and algorithms, there are still topics related to hand-created content pipelines that are neglected or not addressed and thus pose the opportunities for deeper investigation.
CHAPTER 1. INTRODUCTION

1.2 Goal

The general, long-term ambition of all NPR methods is to make the output indistinguishable from the real hand-created imagery and is also the ultimate goal of this thesis. The problem with the various state-of-the-art stylization techniques presented in Chapter 2 is that they typically try to simulate or approximate a particular type of media and their use is thus limited. The example-based methods, described thoroughly in Section 2.1.4, do not have this limitation but often lack in the visual quality of the output – they are not capable of preserving the appearance of the provided examples fully.

Therefore, the state-of-the-art has to be extended and modified to obtain the desired believable results. This thesis presents the reader with the series of methods that gradually build on each other’s findings to finally propose the technique that delivers plausible output, which is almost indistinguishable from the actual artists’ production.

1.3 Contribution

This section summarizes our contribution to example-based synthesis with focus on hand-drawn appearance and related areas. Every paragraph briefly outlines the context of each of our contributions that are covered in greater detail in the following chapters, discusses the main issues with the state-of-the-art approaches and proposes a solution.

Painting by Feature  In a simple painting process, the artist typically draws outlines and then continuously adds details into both the interior and exterior parts either by placing other strokes or by filling them at once. When the drawing is colored using only flat colors, the transfer and emulation of this workflow in a computer program is usually straightforward and without drawbacks of removing a part of the overall control from the user. However, as the textures get richer and more detailed, simple drawing tools, either real-world or virtual, are less effective and more time-demanding. The state-of-the-art offers two different ways to tackle this problem. One lets the user paint every single stroke, similarly to the real brush and paint scenario (e.g., approach of Lu et al. [2013],
CHAPTER 1. INTRODUCTION

Figure 1.3: Stylizing Animation By Example [Bénard et al., 2013]. Top row: stylized animation; bottom row: input shaded images. The artist paints two extreme poses and the inbetweens are synthesized automatically maintaining temporally-coherent appearance between two consecutive frames.

see Figure 2.5), the other one, generally known as Texture by Numbers [Hertzmann et al., 2001] (see the overview in Figure 2.3), is based on additional guidance maps that map parts of the exemplar texture into the output. While both approaches produce great results, they also have some limitation. The former one lacks the ability to paint larger textural areas without the necessity to draw every brush stroke individually, while the latter one provides very coarse control over the used texture and provides a limited way to handle rich textural boundaries. We argue that a combination of these approaches, namely keeping both the “brush” and “paint” paradigms, would provide new ways to the artistic content creation and we propose such system in Chapter 3. See Figure 1.2 that illustrates the importance of careful boundary handling.

Color Me Noisy The substantial time requirements of hand-made imagery creation process become still more noticeable in the case of animation when, depending on the used technique, even few seconds of animation might take several days or even weeks to produce. Many methods could assist the artists in lifting off the most time-consuming part, i.e., the necessity to (re)create everything within a given frame (including the static parts) from scratch over and over again, with consequent benefits and disadvantages. Often the fastest way is to create proper reusable 3D geometry and then imitate the hand-drawn look with various shaders. Researches has been particularly interested in the problem of temporal coherence of such stylization. Without any control over it, the animation would be very hard to watch as the amount of flickering could easily distract the observers and cause visual fatigue. Therefore perfect temporal coherence has been aimed for with some very impressive outcomes, such as the method of Bénard et al. [2013] (see Figure 1.3) for example input and output) that enables the artist to stylize generic 3D animations with given texture examples and also paint specified keyframes. However, while we agree that excessive amount of temporal incoherence may cause visual discomfort, we also argue that the temporal noise in classic hand-drawn animation is not a negative phenomenon but rather an additional tool or channel of artistic expression to convey certain emotions or tension, not to mention the perceived “hand-drawn look” of the resulting animation. We deduce that by giving the user an instrument to control the
temporal noise with better granularity, these properties of real hand-drawn sequences can be simulated in computer-generated animations. In Chapter 4, we describe such a novel approach to the problem of controllable temporal noise.

![Figure 1.4](image1.png)

**Figure 1.4:** As showed by Newson et al. [2014], in patch-based texture synthesis, textured patches are more likely to be matched with smooth ones during the correspondence search. In this example, the task is to generate more of the source texture (a). Without addressing this issue, the patches of the soil with less high-frequency content end up being used extensively (b). When patches are being selected more uniformly, the output looks as expected (c). Images taken from [Kaspar et al., 2015].

**LazyFluids** The practical usability of any example-based texture synthesis method lies, among other aspects, in the quality and faithfulness of its output as compared to the textural qualities of the input exemplar. The goal is to produce texture qualitatively indistinguishable from the user-provided one. For many media and texture types, the state-of-the-art methods produce compelling results. However, for some types of content, these approaches suffer from the so-called “wash-out” effect which results in losing details present in the original exemplar and can therefore significantly degrade the quality of the output texture. This issue is even more prominent in the case of animation, where the negative effect gradually accumulates (see Figure 5.2 for illustration). This issue was first noted by Bargteil et al. [2006] and studied in greater detail by Newson et al. [2014]. The typical handling of this problem is to try to equalize the histogram of exemplar-patch distribution. This solution greatly improves the output quality, indeed, but it is difficult to control and does not guarantee actual histogram uniformity, it rather increases it. Chapter 5 presents our proposal to get rid of the wash-out effect completely, even in animation synthesis. We also demonstrate how to enforce the true patch-usage uniformity to improve the appearance of resulting textures. Moreover, these findings do not only apply to the textured fluid synthesis (that Chapter 5 deals mainly with) but to texture synthesis in general, and we show how it can be used to improve the fidelity of NPR example-based stylization in Chapter 8.

**Brushables** Besides containing complex boundaries, non-trivial texture examples often show one or few dominant global directions together with smaller, less prominent local variations to these directions. For example, blades of grass all grow approximately
in the same direction but exhibit some minor variation. The direct control over the orientation of the texture features is, therefore, a goal that many state-of-the-art approaches try to achieve. Typically, these methods compensate for transformations [Lefebvre and Hoppe, 2006], allow specification of directions in texture [Zhang et al., 2003] or try to estimate the direction field in the image [Kang et al., 2009]. A key limitation of these existing techniques is the decoupling of direction and edge awareness which makes it hard to produce images where the prescribed directionality of the shapes’ interiors interacts with the appearance of its boundaries. See sources Figure 6.1 where the appearance of boundaries depends on a specific context that is given by the directionality of the interior. The interactive method we propose in Chapter 6 combines both edge and direction awareness and enables new way of shape-aware effects authoring as well as example-based stroke synthesis that outperforms specialized state-of-the-art algorithms [Lu et al., 2013; Zhou et al., 2013].

**ShipShape** When working with freehand input, namely when sketching, the typical creative process often involves an initial rough draft version of the intended final art, which is then iteratively improved to achieve the desired appearance. This process involves time and knowledge of the particular drawing system and thus results in a barrier or compromises when the user lacks either of these qualities. In some cases, the final sketch serves only as an input to further stylization techniques, such as the one described in Chapter 3 or [Zhou et al., 2013], and it is, therefore, crucial to improve - beautify - the rough input with minimum additional effort on the side of the user. To this end, we present an interactive tool, described in Chapter 7, that allows for the creation of compelling sketches from raw user input by taking into account implicit geometric relations (see Figure 7.1).

**StyLit** As mentioned earlier, the synthetic nature of example-based NPR images is often revealed by the quality of the output texture. While many types of the content can be successfully recreated using state-of-the-art methods, real-world artistic media, such as watercolor, chalk or pencil, still present a big challenge. If the high-frequency details are not preserved in the synthesis output, artifacts will emerge. In Chapter 8, built on our previous contributions, we propose a method for the example-based stylization of 3D renderings that preserves the rich expressiveness of hand-created artwork on the level unmatched by the state-of-the-art approaches (see Figure 8.1 for example results). We also present a novel type of synthesis guidance that takes the information about light propagation in the scene into account. The guidance can distinguish among context-dependent illumination effects, for which artists typically use different stylization techniques, and delivers a look closer to the realistic artwork.

### 1.4 Structure of the Work

The rest of this thesis is structured as follows. In Chapter 2, we summarize the progress and state-of-the-art in non-photorealistic rendering, focusing in greater detail on example-based techniques that served as a background for our research.

Our contribution is presented in Chapters 3, 4, 5, 6, 7 and 8. Together, they represent a compendium of publications co-authored by the author of this thesis, published in
high-impact, peer-reviewed journals and contributed to the state-of-the-art in this area.

Chapter 9 summarizes briefly the contribution presented in previous chapters and discusses them within the context of state-of-the-art developed in parallel to this thesis.

Appendix A contains the complete list of the authors publications together with their citations in impactful journals. Appendix B states a specific contribution of the author to each of the presented papers.
2 Previous Work

In Section 1.1, we briefly introduced non-photorealistic rendering together with possible promising research areas under this broad topic. In this chapter, NPR methods are first formally classified, mentioning notable ones. Greater length is dedicated to the introduction of example-based methods that constitute the core of our contribution presented in this report. Relevant approaches and works are discussed together with existing applications (often spanning outside NPR) with emphasis on hand-created-look stylization techniques.

2.1 Non-photorealistic Rendering

Non-photorealistic rendering is a field of computer graphics that is concerned with creating digital content that does not focus on photorealism but rather tries to produce output simulating various artistic stylizations, emphasize expressiveness or present information visualization. This is in contrast to the photorealistic graphics where the output is usually created using physically-based algorithms with the intention to simulate the real-world principles as detailed as possible.

A large body of work has addressed the problem of synthesizing the 2D content imitating different artistic styles and mediums, such as cross-hatching, stippling, oil paint, pastel, watercolor and many others. These methods can be loosely categorized into several groups [Kyprianidis et al., 2013].

2.1.1 Medium Emulation

In order to simulate behavior of the target mediums, these methods typically rely on one or more physically based models of either the mediums itself [Small, 1991; Curtis et al., 1997; Van Laerhoven and Van Reeth, 2005; You et al., 2013], the underlying material, e.g., paper or canvas [Van Laerhoven et al., 2004; Chu and Tai, 2005], or the painting instrument, such as the brush [Su and Xu, 2002; Seah et al., 2005; Baxter and Govindaraju, 2010]. These methods achieve compelling results but are often computationally too expensive, need careful parameter tuning or are focused on very specific subset of mediums and cannot be easily used in general case. See Figure 2.1a,b for example images.

2.1.2 Image Filtering and Segmentation

Many image filters originating in the field of image processing have been used to produce stylized output. In particular, variants of edge-detecting [Kang et al., 2007; Winnemöller, 2011; Lu et al., 2012] or edge-preserving filters, e.g., bilateral filter [Kang and Lee, 2008; Gastal and Oliveira, 2011; Kyprianidis, 2011], or thresholding [Xu and Kaplan, 2008] have been used. Often, image segmentation is produced as prior information collapsing the original image into a set of larger regions [DeCarlo and Santella, 2002; Wen et al., 2006]. These methods can often operate at interactive rates. Due to their nature, they typically produce variants of line drawing/sketch stylization or simplifications based on color smoothing. Figure 2.1c,d list examples of these techniques.
CHAPTER 2. PREVIOUS WORK

2.1.3 Stroke-based Techniques

Stroke-based rendering (SBR) algorithms also try to imitate the process of covering the canvas with paint but work on a higher level of abstraction than medium emulations approaches. Here, the rendering primitives are gradually placed on the canvas, often incorporating advanced sampling strategies for positioning of these atomic elements. Depending on a particular visual style, these primitives are referred to as brush strokes [Haeberli, 1990; Hertzmann, 1998], stipples [Ostromoukhov and Hersch, 1999; Secord, 2002; Vanderhaeghe et al., 2007], hatch marks [Kim et al., 2008; Praun et al., 2001; Webb et al., 2002] or tiles [Cohen et al., 2003; Kopf et al., 2006]. The results of these methods are impressive, but are again tailored to specific subset of artistic styles and can only output content with limited variability. See Figure 2.1e,f for example images.

2.1.4 Example-based Methods

In contrast to the methods mentioned above, example-based image creation is not suited for one particular visual style but rather synthesizes the output with respect to provided examples. This gives the user the ability to produce results with a variety of styles and without any expert knowledge of the synthesized mediums. As a downside, the absence
of any additional statistical or parametric model may lead to a deviated synthesis output when the input does not contain a sufficient amount of valid exemplars. However, we think that for its generality and medium-independence, it is the most promising approach among the listed ones.

NPR Example-based Image Rendering

*Image Analogies*, a concept presented by Hertzmann et al. [2001], decompose the synthesis and transfer of the artistic style into two steps. Given an example pair \( A, A' \) (where \( A' \) is an artistically stylized version of \( A \)), the algorithm first learns this mapping \( A \mapsto A' \) and then applies it to the new source \( B \), transforming it into a result image \( B' \) in the same artistic style as \( A' \) (see Figure 2.2). This work also introduced the texture-by-numbers paradigm in which images \( A \) and \( B \) are segmentation maps, possibly roughly created by hand, allowing the user rapidly rearrange the image (see Figure 2.3 for example workflow). Ritter et al. [2006] later extended this concept in *Painting With Texture* where they allow the user to directly paint with textural elements in an almost interactive manner.

Efros and Freeman [2001] proposed *Image Quilting*, a method for texture synthesis and style transfer (see example in Figure 2.4). With appropriate correspondence maps (conceptually resembling the unfiltered images of Hertzmann et al.), their algorithm worked by copying whole patches from the input to the output and stitching them using dynamic programming.

A different approach to the still-image stylization, as well as the stylization of 3D renderings, is presented by the concept of *Painterly Art Maps* proposed by Yan et al. [2008]. These maps are generated semi-automatically from provided input paintings and can then be used to stylize arbitrary 3D models.

*RealBrush* by Lu et al. [2013] presents another useful application based on example-based techniques. The user first has to obtain examples of real mediums (including optional stroke overlaps) from which the new strokes are synthesized online as the drawing progresses (see the input and result samples in Figure 2.5). In a similar, data-driven way, Berger et al. [2013] proposed a portrait sketching algorithm based on an extensive dataset of artistic sketches at different levels of abstractions. Here, after the initial preprocessing of the input portrait photo, the detected edges are vectorized and replaced with suitable hand-drawn strokes from the set.

NPR Example-based Video Rendering

A naïve attempt to extend the image stylization to video footage would be to render each frame separately. However, this approach would result in a significant temporal flickering, and additionally, with no control over it. It must be taken into account, however, that the temporal noise is not always something that the artist would like to omit altogether.

In practise, despite the actual inability of the traditional hand-drawn animation pipeline to create perfectly temporarily coherent sequences, rough manipulation with temporal noise has become a commonly used artistic tool to convey various emotions or moods and to emphasize the expressiveness or rapidness of the animation, which is successfully leveraged by critically acclaimed artists such as Bill Plympton or Frédéric Back. Because the temporal flickering is so tightly coupled with our perception of this
Figure 2.2: Image Analogies [Hertzmann et al., 2001] - from the original Van Gogh’s painting (a) an “unfiltered” source image (b) was generated using Photoshop’s “Smart Blur” filter. The filter learned from this training pair is then applied to photographs (c,e) to obtain their filtered counterpart (d,f).
CHAPTER 2. PREVIOUS WORK

Figure 2.3: Texture By Numbers [Hertzmann et al., 2001] - in the spirit of Image Analogies, for the original image (a), “unfiltered” source segmentation image (b) is painted by hand as well as unfiltered target image (c). The algorithm then transfers the style from (a) to the filtered output (d) conformed to the annotation (c).

Figure 2.4: Image Quilting [Efros and Freeman, 2001] - (a) source texture, (b) target image, (c,d) correspondence maps, (e) texture transfer result.

kind of animation, the noise may even be artificially introduced to otherwise perfectly coherent sequence to induce the impression of being hand-crafted frame by frame, e.g., in Disney’s *Piglet’s Big Movie*.

Nevertheless, in the state-of-the-art NPR techniques, the temporal noise is usually understood as something undesirable that should be avoided rather than invoked. Most methods using the global optimization approach either explicitly incorporate some form of temporal coherence term into the energy function to suppress the occurrence of the temporal noise in the final output, or work directly in the 3D spatio-temporal domain, as the basic optimization scheme can easily be lifted to 3D [Wexler et al., 2004].

Bénard et al. [2013] presented a method to produce temporally coherent animations by extending the method of *Image Analogies* [Hertzmann et al., 2001] to animation, given a 3D CG animation and a static output style sample. Apart from the local similarity and temporal coherence terms, their energy function also accounts for other texture synthesis needs, such as (more) uniform source patch distribution or emphasized matching of line features. The solver then performs several forward-backward optimization sweeps on each level of the coarse-to-fine pyramid using PatchMatch [Barnes et al., 2009] to improve the nearest neighbor field. One of the great advantages is also the possibility for the user to constrain selected keyframes by painting them explicitly, and let the algorithm smoothly synthesize the in-between frames.

As laid out in Chapter 1, very few works addressed the actual control over the temporal noise in the synthesized sequences. Namely, Noris et al. [2011] presented a system enabling the user to reduce the amount of noise in the set of rough digitally drawn sketches.
Figure 2.5: RealBrush [Lu et al., 2013] relies on a set of scanned and preprocessed source medium stroke exemplars, including the smearing and smudging features (a). Once acquired, the user can produce realistic paintings using this set at interactive rates (b,c). While working great for stroke-like mediums, e.g., oil/acrylic paint, it is rather impractical for fill-like mediums, such as watercolor.

to the desired rate by registering individual strokes and then interpolating between the original animation and noise-free in-betweens. However, their approach still requires hand-drawn animation as an input and is limited solely to sketchy animations.

2.2 General Example-based Image and Video Synthesis

In Section 2.1.4, we listed example-based methods as a subset of non-photorealistic rendering. However, NPR is only a part of general image/video example-based synthesis using approaches originally intended for other purposes, such as texture or variety synthesis. In this section, notable works and results are listed.

The example-based rendering works by transferring texture information from the source to the target. This is similar to texture-hole-filling methods, a field initiated by Efros and Freeman [1999], that fill empty or undesired parts in the output target image with samples taken elsewhere, that is from the source image. The patches to complete the missing portion of the information are selected according to their similarity using a distance metric, e.g., a sum of squared or absolute differences.

As the patches are only selected on the local-similarity basis, there is no guarantee of finding the global optimum. To remedy this, Wexler et al. [2004] formulated the texture synthesis as the energy-minimization problem, with the energy being a sum of differences of all patches to their respective nearest neighbors in the source content. Then they utilized the EM-like approach to decrease this total energy iteratively. The new value of each pixel in each iteration would be recalculated as the weighted average of overlapping patches. Wexler et al. also presented the multi-level synthesis, an approach widely used in computer vision or signal processing communities, to help the algorithm avoid undesirable local minima and also to initialize the synthesis on finer scales with the solution obtained in coarser levels.

As a further improvement, Wexler et al. developed the multi-scale optimization scheme, most notably by replacing the weighted average in E-step with the mean shift to prevent the otherwise inevitable blurring of the synthesized content, and by computing the nearest neighbor candidates for all pixels in the synthesized output instead of only
a sparse subset of them. Simakov et al. [2008] then built upon that and suggested that for the output to faithfully represent the maximal amount of information present in the source, the energy function should be bidirectional, as illustrated in Figure 2.6. Ramanaarayanan and Bala [2007] took a different approach to the constrained texture synthesis and employed graph cuts to assign patches from the source image to the output according to a provided pair of labeling maps.

![Figure 2.6: Bidirectional similarity concept (completeness + coherence) [Simakov et al., 2008] for spatial and spatio-temporal patches – two signals are considered visually similar if all patches of one signal are contained in the other signal, and vice versa.](image)

Almost all example-based synthesis methods rely, in their core, on finding the nearest neighbor field (NNF), which can be precomputed to some extent [Lefebvre and Hoppe, 2005, 2006], but typically is evaluated on the fly [Wexler et al., 2004; Simakov et al., 2008] and is usually the most computationally demanding part of the synthesis. Barnes et al. [2009; 2010] addressed this issue by introducing PatchMatch - a randomized matching algorithm that exploits the local coherence property to speed up the matching process by propagating good correspondences along the NNF.

To introduce randomness in the synthesis process, Lefebvre and Hoppe [2005] proposed a jitter-correct concept, in which the NNF is first perturbed by a variable jitter factor, controlled by the user or derived automatically by the algorithm, on each level of the coarse-to-fine image pyramid. Depending on the size of the jitter and on the level on which it is applied, a certain amount of variation in the output can be achieved; however, it can be hard or impossible to achieve a smooth transition of variation. Later, Risser et al. [2010] used this approach to synthesize a large amount of variations from sparse input sets of complex images (see Figure 2.7b for sample output of their algorithm). To remove the expensive pre-processing step of Appearance Space Texture Synthesis, Panareda Busto [2010] replaced the k-coherence search step with a parallel random walk. This allowed for larger texture exemplars to be used as well as it maintained the full search space unlike the original scheme of Lefebre and Hoppe [2006].

**Related Applications**

Generally speaking, the aforementioned methods are used to produce a new content with properties of the given source exemplars and potentially also satisfying some additional constraints (e.g., local orientation). In the image/video domain, methods such as Texture Optimization for Example-based Synthesis [Kwatra et al., 2005] (see Figure 2.8), Parallel Controllable Texture Synthesis [Lefebvre and Hoppe, 2005] or Multiscale Texture Synthesis [Han et al., 2008] are typically used to create large textures from considerably smaller source exemplars, e.g., to texture large areas without noticeable repetitions. Another useful application scenario is inpainting [Wexler et al., 2004; Darabi et al., 2012] (see Figure 2.9 and 2.10), in which only the missing or unwanted parts of the source are
CHAPTER 2. PREVIOUS WORK

Figure 2.7: Variety synthesis [Risser et al., 2010] - (a) input exemplars, (b) output hybrids.

Figure 2.8: Texture Optimization for Example-based Synthesis [Kwatra et al., 2005]. Image texture synthesis using multi-scale approach - from small source sample larger texture is synthesized.

When the task is to generate variety from a limited set of input exemplars, the user-assisted exemplar-based methods can be used. This task comes to high importance because the human visual system is very good at observing repetitive patterns and thus suppressing it can highly improve the perceived realism of the generated content (e.g., think of the forest in a computer game, where every leaf on each tree is the same). The user is typically required to provide some additional information such as reference layering in More Of The Same [Assa and Cohen-Or, 2012] or to provide the content already aligned to some degree as in Synthesizing Structured Image Hybrids [2010] (see example in Figure 2.7). A similar approach can be utilized also outside the image/video domain.

Figure 2.9: Space-Time Video Completion [Wezler et al., 2004]. Top row: original video sequence with the occluding person marked in magenta; bottom row: the missing content is inpainted using the rest of the source footage.
to generate 3D geometric objects combinations [Zheng et al., 2013], animations [Zheng, 2013], textures [Bhat et al., 2004] or even urban layouts [Aliaga et al., 2008].

Another important application of example-based algorithms is to emulate or transfer the real drawing mediums such as oil paint or watercolor. The workflow may differ - the user may specify the set of reference/target image pairs, as in Image Analogies [2001] and Image Quilting [2001], or they can directly work with brush strokes [Kim and Shin, 2010; Lu et al., 2013].

Figure 2.10: Image inpainting [Darabi et al., 2012] - (a) original image, (b) marked hole region, (c) inpainted result.
3 Painting by Feature: Texture Boundaries for Example-based Image Creation


http://dl.acm.org/citation.cfm?id=2461956

### 3.1 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 3.1. 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 [DeCarlo et al., 2003], such artifacts become immediately apparent (see comparison in Figure 3.2).

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
Figure 3.1: 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
complex, visually appealing drawings with our system requires similar effort as creating a simple contour sketch in standard drawing systems.

3.2 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 3.2b.

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.

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 3.2d 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 tem-
poral 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.

3.3 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.
CHAPTER 3. PAINTING BY FEATURE

Figure 3.3: 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.

3.3.1 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 3.4). 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 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 3.3. 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 3.5.

![Figure 3.5: 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).](image)

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.

3.3.2 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 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. 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.

3.4 Applications and Results

An overview of our image creation workflow is illustrated in Figure 3.6. Due to its generality, our approach can be utilized in several applications. One of our primary applications is vector image stylization: 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 3.1 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 Figure 3.2). Figure 3.1a 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 3.1b 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 3.1c. 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 3.1d, note the diffusion effect along the cat’s whiskers and mustache. A comparison with Diffusion Curves is available in Figure 3.7. 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 3.1d). Additional examples of different stylizations given a single user-drawn sketch are shown in Figure 3.8. 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 3.6 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

Figure 3.6: 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
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 3.9. Further potential applications of our method include image editing scenarios such as inpainting. We refer readers to the supplementary material for a representative result.

3.4.1 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 3.10). 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 3.3.1. 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 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 3.11) . Similarly, selection of an area feature which is incompatible with already drawn line features may also lead to an erroneous result (see Figure 3.12) . 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 3.3.1). 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 se-
Figures 3.8: 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

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.

3.5 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.
Figure 3.10: 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 3.11: 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 yields a completely erroneous result (bottom).

Figure 3.12: 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
4 Color Me Noisy: Example-based Rendering of Hand-colored Animations with Temporal Noise Control


4.1 Introduction

Hand-colored animation is a type of traditional animation, where each frame is created independently, from scratch. It has a distinct visual style represented by a certain amount of temporal flickering which arises due to misalignment of details in consecutive frames. This characteristic feature lends it a look of liveliness and emotional expressivity, which is being successfully leveraged by critically acclaimed artists such as Bill Plympton and Frédéric Back.

While temporal noise is usually understood as an undesirable artifact in NPR techniques [Bousseau et al., 2007; Bénard et al., 2009; Bénard et al., 2011; O’Donovan and Hertzmann, 2012], used judiciously it may serve as an additional medium of artistic expression, either to evoke a hand-crafted look (such as sketchbook scenes in Disney’s

![Figure 4.1: Examples of hand-colored animations generated using our approach (from top to bottom): walker (watercolor), teddy (oil pastel), and strongman (watercolor). Note how our method creates variety introducing a desired level of temporal noise while preserving the high-frequency details of the drawing medium and the low-frequency content created by an artist.](image)

![Figure 4.2: Motivation—hand-colored animations (a) look temporally coherent when low-pass filtered (b). However, at higher frequencies they contain details that reflect physical properties of the drawing medium and introduce temporal noise (c).](image)
 CHAPTER 4. COLOR ME NOISY

Piglet’s Big Movie), or to set a certain mood (e.g., Shadow World sequences in The Lord of the Rings). In Bill Plympton’s more recent work (e.g., Cheatin’) noisy, hand-drawn sequences are combined with coherent sequences to convey different moods.

However, the nature of the medium makes it difficult to control the amount of noise, and high noise levels can cause visual fatigue in the viewer. This, in conjunction with the amount of labor involved in production, creates a demand for a more automated process that lets artists control the amount of noise without eliminating it entirely.

Noris et al. [2011] recently presented a system which affords control over the amount of temporal flickering in a sequence of digitally drawn sketches. By registering individual strokes in selected keyframes, they reduce temporal jitter using a weighted combination of original noisy motion and smoothed inbetweening. Although this approach produces impressive reduction of temporal noise level for sketchy vector drawings, it still requires a hand-drawn animation as an input.

Our aim is to reach a more practical workflow that takes a temporally coherent animation created using existing CG pipelines and enriches it with temporal noise synthesized de novo from examples of an arbitrary drawing medium. A similar workflow was recently used by Bénard et al. [2013] in their framework, which extends Image Analogies [Hertzmann et al., 2001] to render impressive stylized animations with a specific style or drawing medium given by example. They focus on enforcing temporal coherency using a sophisticated system of correspondence propagation; however, the underlying re-synthesis technique does not permit control over the amount of temporal noise.

In this paper, we propose a novel example-based technique that not only preserve temporal coherency but also introduces a controllable amount of temporal flickering that conveys lively dynamics and visual richness which can be used either to evoke an impression of hand-colored look or provide an additional dimension of expressivity.

4.2 Related Work

Synthesizing various drawing media is one of the key challenges of non-photorealistic rendering. A wide spectrum of techniques spanning from computational approaches [Curtis et al., 1997; Haevre et al., 2007; Lu et al., 2012] to realistic example-based methods [Zeng et al., 2009; Lu et al., 2013; Lukáč et al., 2013] has been developed. A key issue arises when these techniques are applied to animations, where frame-independent synthesis leads to unpleasant temporal noise that affects viewing comfort.

Many techniques have been proposed to alleviate this issue by enforcing temporal coherency [Bousseau et al., 2007; Bénard et al., 2009; Bénard et al., 2011; O’Donovan and Hertzmann, 2012]. Although these methods produce visually pleasing results, their visual structure is inconsistent with the natural look of noise typical for hand-colored animation. A similar limitation holds also for procedural noise generation [Bénard et al., 2010; Kass and Pesare, 2011] which allows for temporally coherent stylization by suppressing temporal components of the generated noise.

Related to synthesis with temporal coherence are methods that try to enforce variety during synthesis [Lefebvre and Hoppe, 2005, 2006; Risser et al., 2010]. They introduce the ability to vary randomness between scales, but due to being formulated in index domain, they cannot de-couple visual information across scales, which would be necessary for temporal noise control. Related multi-scale texturing approaches [Vanhoey et al., 2013]
may use a separate source for each scale, but decomposing an example image in this way is problematic.

A different approach to variety synthesis proposed by Assa and Cohen-Or [2012] does not rely on texture synthesis but instead decomposes the exemplar into layers, which are then recombined and the result is randomly warped. In our scenario we would like to conform to the user-defined shape, and the remaining small number of discrete varying outputs is insufficient to simulate the variety typical for hand-colored animation.

Our approach is inspired by image morphing techniques [Shechtman et al., 2010; Darabi et al., 2012] that extend state-of-the-art image synthesis algorithms [Wexler et al., 2007; Simakov et al., 2008]. Although these methods have the potential to simulate the look-and-feel of hand-colored animation they do not address the control over the amount of temporal noise.

4.3 Our Approach

Noise found in hand-colored animations has a specific nature. Artists tend to preserve coherency at a global level—when the sequence is viewed at a distance (see Figure 4.2a) or when a low-pass filter is applied (Figure 4.2b) the animation is perceived to be temporally coherent. However, at a local level, temporal variance in high-frequency details becomes visible (Figure 4.2c). This creates the impression of visual richness, reflecting the real physical properties of the drawing medium used.

![Figure 4.3: Synthesis (denoted by \(\triangleright\) operator) of a noisy target animation \(T\) (subscripts denote frame numbers) from a static reference animation \(R\) (red input) and a source drawing medium \(S\) (blue input). When \(R\) is gradually blurred \((R', R'')\) using a low-pass filter \(h_f\) with increasing strength \(f\) then changes \((d', d'')\) between corresponding synthesized frames \((T'_0, T'_1\) and \(T''_0, T''_1\)) start to be more apparent and the level of perceived temporal noise increases. Note, however, that individual frames of \(T'\) and \(T''\) appear similar when viewed side-by-side with corresponding reference frames in \(R\).](image-url)
A characteristic feature of hand-colored animations is that physical properties of the drawing medium are hard to control, so maintaining temporal coherency becomes tedious. The difficulty increases with the scale of details an artist wishes to preserve as coherent. Due to this, hand-colored animations contain specific spatial changes between individual frames that are perceived as a high-frequency temporal noise when shown successively. The noise has flat power spectrum (white noise) and gets subjectively stronger when the scale of changing details increases (see Section 4.4.1 for details).

Vision science offers an explanation of this perception with multi-channel models of human vision [Winkler, 2005]. When the human visual system processes the temporal signal, two visual mechanisms, the transient and the sustained channels, come into play [Kulikowski and Tolhurst, 1973; Watson, 1986]. The sustained channel performs a detailed analysis of stationary, or slowly moving, objects (low temporal frequencies) while the transient is involved in signalling the spatial location or change in spatial location (high temporal frequencies). The content of transient channel is therefore perceived as noise, stimulus flickering, or apparent movement [Mäkelä et al., 1994]. We hypothesize and experimentally measure (see Section 4.4.1) that the larger the spatial changes in frames, the higher the power spectrum of temporal frequencies, the higher the energy in transient channel, and accordingly the higher the level of perceived noise in animation.

This mechanism motivated us to design a new algorithm that enables control over the amount of perceived temporal noise (see Figure 4.3). We render a sequence of images that have similar low-frequency content as the reference animation while high-frequency details are reintroduced by example in a random fashion. The user can then change the frequency threshold to increase/decrease spatial extent of synthesized details and thus control the level of perceived temporal noise.

In the rest of this section we formulate the problem more precisely and propose an algorithm to solve it. We also briefly mention simple extensions that can further improve the quality of the resulting image sequences.

### 4.3.1 Problem formulation

The input to our algorithm is a sample of a real drawing medium $S$ and a sequence of $N$ reference images $R$ that represent a coherent, noise-free animation with a similar appearance to $S$ (see Figure 4.3). The task is to synthesise a target animation $T$ that
satisfies the following three criteria (see Figure 4.4):

1. **Fine consistency.** Visual dissimilarity between source $S$ and target $T_i$ should remain small ($i$ is the frame number). This can be accomplished by minimizing established patch-based energy [Wexler et al., 2007]:

$$\sum_{q \in T_i} \min_{p \in S} ||P - Q||_2^2$$

(4.1)

where $Q$ denotes a patch of size $w \times w$ centered at the target pixel $q$, and $P$ is a patch of the same size taken from source pixel $p$, possibly undergoing additional geometric transformations (we consider rotations and reflections).

2. **Coarse consistency.** Low-frequency content of $T_i$ should be close to the low-frequency content of $R_i$. Formally we need to minimize the $L_2$-norm over all pixels of the low-pass filtered signals:

$$||h_f \ast R_i - h_f \ast T_i||_2^2$$

(4.2)

where $h_f$ is the low-pass filter with tunable strength $f$ and $\ast$ is the convolution operator.

3. **Temporal noise.** Suppose $R$ is a sequence showing a static image over several frames and $q_i$ is a 1D function yielding the value of a target pixel $q \in T_i$ at the frame $i$. We would like $q_i$ to contain a signal with white properties, i.e., its power spectrum $Q(\omega) = |F(q)|^2$ should have uniformly distributed energies over all frequency bands. Formally we can express this by minimizing the standard deviation of $Q$:

$$\frac{1}{N} \sum_{\omega=1}^{N} \left( Q(\omega) - \frac{1}{N} \sum_{\omega' = 1}^{N} Q(\omega') \right)^2$$

(4.3)

Such a criterion can also be applied to a more general setting when $R$ contains moving objects. In this case we assume global motion between consecutive frames is compensated before values of $q_i$ are computed.

Note that, surprisingly, in (4.3) there is no explicit control over the amount of perceived temporal noise. The only aim of (4.3) is to enforce randomness in the optimization process. Instead, the control is implicitly encoded in the strength $f$ of the low-pass filter $h_f$ used in (4.2). This follows from our original observation that $f$ can influence the scale of random changes between consecutive frames and thus control the level of perceived temporal noise.

### 4.3.2 Algorithm

In this section we propose an algorithm (see Figure 4.5) that jointly optimizes the proposed criteria (6.5–4.3). It extends the multi-scale EM-like optimization scheme originally proposed by Wexler et al. [2007] to find a good local minimum of (6.5).

**Fine consistency.** The algorithm of Wexler et al. [2007] utilizes image pyramids $\triangle_S$ and $\triangle_T$ to represent the source and target images at multiple scales. It starts with the coarsest level $\ell = 1$ and gradually upsamples the solution until the finest level $\ell = M$ is reached. At each level of the pyramid $\ell$ the following steps are performed iteratively:
• find nearest neighbor patches $P \subset \triangle^\ell_S$ for all target patches $Q \subset \triangle^\ell_T$ so that $||P - Q||_2^2$ is minimal.

• for each pixel $q \in \triangle^\ell_T$ compute the mode of colors at collocated pixels $p \in \triangle^\ell_S$ that belong to retrieved nearest neighbor patches $P$.

**Coarse consistency.** To integrate (4.2) into the joint optimization process we can exploit the fact that the original Wexler algorithm uses a multi-scale approach to optimize (6.5). In our setting the synthesis at lower levels of the target pyramid $\triangle^\ell_T$ is redundant since from a certain level $k$ a good solution $\triangle^k_T$ is already known: $\triangle^k_T = h_f * R_i \downarrow^f$, where $\downarrow^f$ denotes the downsampling operator that sets an appropriate sampling rate according to the strength $f$ of the low-pass filter $h_f$. This leads us to propose the following modified version of the original algorithm.

Given the source drawing medium $S$ and the user-specified strength $f$ of the low-pass filter $h_f$, we initialize source pyramid $\triangle^\ell_S$ by low-pass filtering and subsampling $S$ at multiple levels $\ell = 1 \ldots M$:

$$\triangle^\ell_S = h_{f(\ell)} * S \downarrow^{f(\ell)}$$  \hspace{1cm} (4.4)

where $f(\ell)$ is a function which interpolates strength of the low-pass filter $h$ according to the level $\ell$. For a box filter where $f$ is the width of the box, $f(1) = f$ and $f(M) = 1$. Inbetween values are set so that the sampling rate of two consecutive levels decreases with a subtle ratio of 0.85, as in the work of Simakov et al. and Shechtman et al. [2008; 2010], to reach finer granularity during the synthesis and help avoid visual artifacts.

Once the source pyramid is built we create a target pyramid $\triangle^\ell_T$ with the same resolution as levels of $\triangle^\ell_S$ and enforce (4.2) by feeding downsampled low-frequency content of the reference animation frame $R_i$ into the coarsest level of $\triangle^\ell_T$, i.e., $\triangle^1_T = h_f * R_i \downarrow^f$.

After this initialization the algorithm continues as usual.

Note that successive downsampling of reference animation leads to removal of high-frequency details and introduces fuzziness into the shape of region boundaries. This is a desirable effect which is characteristic for drawing media such as watercolor (see Figure 4.2). Nevertheless, there can be situations when these irregularities are unintended. In such case we provide mechanisms that allows to improve the quality of border synthesis using local noise control and source selection. These extensions are further discussed in Section 4.3.3 and supplementary material.

**Temporal noise.** Suppose we have the same simplified setting as described in the formulation of *temporal noise* criteria, i.e., a reference animation $R$ that consists of a static image played over several frames. The algorithm proposed so far would lead to a sequence of static images $T$, where each pixel $q \in T$ would be constant over time. This is the situation we need to avoid as our aim is to produce a noisy sequence.

Direct minimization of (4.3) would be problematic as it requires computation in the frequency domain, operates over a large amount of data, and for moving objects complex optical flow estimation is necessary to compensate for the global motion. Rather than trying to minimize (4.3) explicitly we instead synthesize each frame independently and introduce randomness into the original deterministic algorithm by randomly voting over possible patch candidates and pre-deforming the source $S$. Later (in Section 4.4.1.1) we demonstrate that such a simplified solution is sufficient to obtain noisy sequences with equally distributed energies over all frequency bands as required by (4.3).
Figure 4.5: Algorithm—the source drawing medium $S$ is randomly pre-deformed and image pyramids of source $\Delta S$ and target frame $\Delta T_i$ are built. The coarsest level of $\Delta T_i$ is initialized by downsampled reference frame $R_i$. The user-specified strength $f$ of the low-pass filter $h_f$ is used for downsampling. The algorithm starts from coarsest level of $\Delta T_i$ and continues towards finer levels. At each level $\ell$ fine consistency between $\Delta S$ and $\Delta T$ is improved. During this process generalized PatchMatch is utilized to find nearest neighbor patches. The seed for the randomized search is always changed to avoid determinism.
Recently, PatchMatch—a fast approximate nearest neighbor search algorithm [Barnes et al., 2009, 2010] has become popular. Besides significant performance gains, it offers a kind of non-determinism that we can exploit in our scenario. The algorithm uses a random number generator to perform sampling over possible candidates in the space of source patches. Changing the seed of this generator causes the optimization to converge to a different local minimum, changing the appearance of the resulting image.

For low values of \( f \) when the synthesis comprises only a few pyramid levels, the likelihood of changes caused by randomized PatchMatch reduces significantly. Accordingly, the temporal variance of the resulting sequence \( T \) is insufficient to evoke perception of noise in the observer. We attribute this to two known perceptual principles: visual grouping [Blake and Lee, 2005] and feature fusion [Scharnowski et al., 2007]. It was hypothesized that if two visual features have a “common fate” (e.g. they move slowly together in the same direction) and/or are “close enough” in the successive frames, the observer is able to align and fuse them. They are thus perceived as a single object in an apparent motion. An effect of synthetic, unpleasant “floating texture” is then perceived instead the desired noise (see supplemental video for visual inspection).

We address this by randomly pre-deforming the source texture for each synthesized frame, constructing a control lattice with the control points randomly moved in a small radius. The result is deformed using an as-rigid-as-possible moving least squares approach [Schaefer et al., 2006]. For our examples we set the grid size to 50 pixels and shift each point 15 to 25 pixels in a random direction. The average offset between synthesized features in two successive frames is above 20 pixels, which corresponds approximately to 20’ (visual arcminutes). This value is much higher than the theoretical minimal offset [Scharnowski et al., 2007] needed for spatial superposition (2’ ≈ 2 pixels). This ensures that generated random features are sufficiently far apart to avoid visual fusion.

Note again that the control over the amount of perceived temporal noise is not addressed in this step since it is already encoded in the previous coarse consistency phase by setting the strength \( f \) of the low-pass filter \( h_f \). The algorithm performs the synthesis starting from the initial coarse solution that corresponds to the low-pass filtered version of \( R_i \) and then optimizes for fine consistency while using randomization to avoid getting stuck in the same solution. As the scale of randomly synthesized details increases with the increasing strength \( f \) the resulting target animation \( T \) appears to be more noisy to the observer (see Section 4.4.1 for evaluation).

### 4.3.3 Extensions

The proposed algorithm can be improved further to gain local control over the amount of temporal noise which can help to preserve salient structures (see Figure 4.6), make the viewer pay attention to certain parts, or introduce additional channel of artistic expression (see supplementary videos). To enable this control, the isotropic \( h_f \) in (4.2) is replaced with a spatially varying low-pass filter where for each pixel different strength \( f_p \) is used. This change is incorporated into our algorithm by setting a different starting level for each pixel, i.e., at pixels with higher \( f_p \) the synthesis starts at the coarsest levels of the image pyramid.

Orientation of the synthesized strokes (see Figure 4.7) can also be controlled locally to emphasize the shape of the animated object or motion orientation. To do that the user can specify two additional orientation fields: \( O_S \) for the source drawing medium and \( O_R \)
CHAPTER 4. COLOR ME NOISY

Figure 4.6: Local noise control—with higher levels of noise the overall shape consistency and presence of small but semantically important features are not guaranteed due to suppression of high-frequency details (left). By specifying the spatially varying strength $f_p$ of the low-pass filter $h_f$, sensitive parts can be synthesized with a lower noise level and thus preserved (right).

for the frames of the reference animation. These can either be obtained automatically, e.g., by computing the per-pixel structure tensor [Bénard et al., 2013], or painted by the user. When fine consistency term (6.5) is evaluated $P$ is always rotated to compensate for orientation mismatch between $P$ and $Q$ and during the correspondence propagation in PatchMatch [Barnes et al., 2009], axes-aligned directions are rotated to respect the actual orientation of $P$.

Figure 4.7: Local orientation control—prescribed orientations enable the algorithm to synthesize output that better follows the shape of the target region (right) in contrast to the uncontrolled synthesis (left).

Besides noise level and orientation, the choice of the source drawing medium can also be controlled locally to improve the quality of the synthesized image. Further details can be found in the supplementary material.

4.4 Results

We implemented our method using C++ except for PatchMatch [Barnes et al., 2009], which was implemented in both C++ and CUDA. By default we use simple box filter for the low-pass filter $h_f$ of which the strength $f$ is expressed by the width of the box in pixels. When the source drawing medium contains sharp details a more accurate Lanczos3 filter [Darabi et al., 2012] can be used to improve visual quality. For the fine consistency term we use patches of size $w = 7$ and perform 4 Wexler et al. [2007] optimization iterations using 8 PatchMatch iterations at each pyramid level. This number was set empirically to make a balance between effect of randomization and the final visual quality. A lower value causes visual artifacts while a higher value can suppress the effect
of randomization as there is higher probability that the algorithm reaches a globally optimal solution.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/figure4.8}
\caption{Results—an additional set of 2D animations: (a) golem [crayon], (b) tree [watercolor], (c) dragon [fire]. See the supplementary video for animations in motion.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/figure4.9}
\caption{Examples from the evaluation dataset consisting of hand-colored snakes painted using different drawing media: (a) crayon, (b) chalk, (c) colored pencil, (d) regular pencil, and (e) watercolor. Top row: hand-made, bottom row: synthesized.}
\end{figure}

We applied our method to a set of four 2D and two 3D animations (see 4.1 and 4.8). For the 2D cases a shape in a rest pose was created and then a static textured image $R_0$ was synthesized using [Lukáč et al., 2013] based on a drawing medium $S$. This image was then deformed using as-rigid-as-possible deformation [Sýkora et al., 2009] to produce the temporally coherent animation $R$. In 3D we mapped textures synthesized from $S$ using [Lukáč et al., 2013] on an animated triangle mesh and rendered the temporally coherent animation $R$. For each $R$ we synthesized $T$ based on $S$ in various noise levels and played them sequentially creating the impression of noise slider (see the supplementary video).

The average computation time for one animation frame of size 1Mpix was approximately 30 seconds using one core of a Xeon 3.5GHz and 5 seconds when a CUDA version
4.4.1 Evaluation

To evaluate the proposed method we created a simple experimental animation. It consists of 12 different poses of a striped snake created from a rest pose using as-rigid-as-possible deformation. We printed these poses on a paper using thin outlines and let an artist paint them manually using 5 different drawing media (see Figure 4.9, top row and supplementary video). Then we scanned them and performed rectification. As the deformation field is known for each animation frame, we can easily compute its motion compensated version. The resulting hand-colored sequences serve as both ground truth for comparison and examples of drawing media for the synthesis of target sequences (see Figure 4.9, bottom row).

While it would be possible to compare the visual plausibility of generated animations against these sequences using a two-alternative forced choice subjective experiment, it should be noted that such a comparison would not in itself be rigorous. This is because the natural animation contains multiple unknown hidden parameters, such as locally varying noise levels or orientation field flickering for anisotropic media, that would have to be matched first. Furthermore, such a comparison would be aesthetic at best, because it is impossible to judge the plausibility of the temporal noise separately from the plausibility of the still image, which is significantly affected by the selected synthesis method.

4.4.1.1 Spectral Analysis

We analysed the spectral properties of the synthesized sequences for increasing strengths \( f \) of the low-pass filter \( h_f \), both in spatial and temporal domains after motion compen-
CHAPTER 4. COLOR ME NOISY

sation. Results for the chalk sequence (see Figure 4.9b) are presented in Figure 4.10 (for other media see supplementary material).

In the spatial domain the power spectra of the frames synthesized using different strengths are similar (Figure 4.10a), i.e., the overall visual characteristic does not change significantly. The method does not introduce notable over-smoothing with increasing \( f \); only a subtle sharpening effect is visible. When compared to the power spectrum of the real frame a more notable difference indicating subtle over-smoothing is apparent, i.e., the synthesized images do not look as sharp as the original painted by the artist. The amount of this smoothing effect varies across drawing media and is typically small enough so that the synthesized images look convincing (see supplementary material).

In the temporal domain the average power spectrum of \( T \) has the energy distributed equally over all frequency bands (Figure 4.10b), which corresponds to our aim to obtain characteristics of white noise. It is also visible that the higher the strength \( f \), the higher the overall energy in the temporal spectrum. This indicates increased perception of temporal noise, which can be further verified by measuring power of sustained and transient channels [Winkler, 2005; Aydin et al., 2010]. Results are illustrated in the supplementary material, and overall energies for the chalk sequence are plotted in Figure 4.10c. These measurements confirm that the energy in the transient channel grows with the increasing strength \( f \) of the low-pass filter \( h_f \) and thus the perception of noise level increases.

### 4.4.1.2 Subjective Experiments

The spectral analysis above shows evidence that the increasing strength \( f \) of the low-pass filter \( h_f \) results in corresponding growth of temporal noise. However, the relation between the strength \( f \) set by the user and the real quantity of perceived temporal noise remains to be investigated. Furthermore, it is also not clear how specific properties of the drawing medium (e.g., crayon, watercolor) affect the visibility of increasing noise level in animations and how this influences eye strain of the observer.

To that end we designed two subjective experiments with 50 and 64 participants, respectively. In the first experiment participants were asked to compare pairs of random sequences generated using our method and for each pair select a sequence that appears more noisy to them. In the second experiment we show just one sequence per question and ask the participants to rate the degree of eye strain they experienced while watching it. There were 4 simulated media: crayon, chalk, colored pencil, and watercolor (we excluded the failure case, regular pencil), and 8 generated levels of noise for each animation, i.e., 32 video stimuli in total.

The overall results of both studies are shown in Figure 4.11. According the ANOVA tests [Mongomery and Runger, 1999] the null hypothesis "there is perceptually no difference between levels of temporal noise in the presented sequences" can clearly be rejected \( (p \ll 0.001) \), meaning that change of low-pass filtering strength \( f \) produces sequences with perceptually different noise level. The same holds also for eye strain. The multiple comparison test (Tukey’s honestly significant differences [Hochberg and Tamhane, 1987]) returns an overall ranking of the individual noise levels and the eye strain with an indication of the significance of the differences. In the first experiment, there is a statistically significant difference between each level of temporal noise produced by each value of \( f \).
1.3 1.6 1.9 2.2 2.6 3.1 3.6 4.3
-4 -3 -2 -1 0 1 2 3
(a) Temporal noise (overall)

1.3 1.6 1.9 2.2 2.6 3.1 3.6 4.3
-4 -3 -2 -1 0 1 2 3
(b) Eye strain (overall)

1.3 1.6 1.9 2.2 2.6 3.1 3.6 4.3
1 2 3 4

Figure 4.11: Overall results of subjective experiments—(a) the two-alternatives-forced-choice study on the perception of temporal noise and (b) the study of eye strain in hand-colored animations synthesized using our method. Error bars show the standard errors.

The second experiment exhibits two statistically significant groupings: chalk, crayon, watercolor (greater visual discomfort) and watercolor, pencil (lesser visual discomfort).

Furthermore, the first experiment did not show any statistically significant effect of the simulated medium on the level of perceived temporal noise. Nevertheless, the second experiment indicated that there may be a small effect of medium type on the eye strain. Results also indicate slight non-linear relationship between the strength of the low-pass filter \( f \) and perceived amount of temporal noise. This motivated possible perceptual linearisation of our method, as shown in the supplementary material.

In summary, both studies confirmed there is a relationship between setting the strength \( f \) of the low-pass filter \( h_f \) and the levels of perceived temporal noise and eye strain. With increasing \( f \) the level of noise and eye strain increases. More details about experiment setup and obtained results can be found in the supplementary material.

4.4.2 Comparison

For comparison purposes, we have attempted to adapt the methods of [Lefebvre and Hoppe, 2005] and [Bénard et al., 2013] to synthesize results close to our hand-colored animation scenario. Clips comparing these algorithms with our approach are included in the supplementary video.

We have extended [Lefebvre and Hoppe, 2005] in order to synthesize an animation with a configurable amount of temporal noise by manipulating the noise/scale settings and using an appropriate source of randomness. While the result was a reasonably consistent noisy sequence, as compared to our approach the algorithm is unable to preserve either high-frequency details of the original drawing medium or the prescribed low-frequency content. It also cannot easily provide local control over the amount of noise.

When attempting to use [Bénard et al., 2013] we encountered the problem that even when setting different weights to the temporal coherence term the synthesis tends to converge to near-identical results on consecutive frames when the shape or color of the object...
of interest do not change significantly. The only way to obtain noisy sequence was to deactivate advection vectors and let the algorithm synthesize each frame independently. However, this solution offers only one noise level which cannot be further controlled and there is no guarantee that the resulting sequence will be temporally coherent (see supplementary material for further details).

4.5 Applications

By combining TexToons [Sýkora et al., 2011] with our approach, one can produce hand-colored animation from a sequence of outline-only hand-drawn sketches. Moreover, our technique can mask shower door artifacts that sometimes appear because of the approximative nature of the original TexToons framework (see Figure 4.12 and the supplementary video).

![Figure 4.12: TexToons—the output from the TexToons algorithm is used as a reference for resynthesis. The shifted texture in the original sequence is denoted by the green curve (upper row). With our approach (bottom row) consecutive frames do not suffer from the “shower door” effect.](image)

Other possible applications of our framework such as stylization and imperfection masking in particle simulations, or painterly rendering of photos and videos can be viewed in supplementary materials.

4.6 Limitations

![Figure 4.13: Limitations—synthesized frames (T_{a,b,c}) may not properly convey the look of the drawing medium (S_{a,b,c}) when it contains different colors (S_a) from the reference (R_a) or subtle high-frequency details (S_b) that cannot be distinguished by intensity level or color, or when the reference frame contains solid colors (R_c).](image)
An implicit assumption of our method is that areas in the reference animation $R$ have counterparts in the source $S$ that are similar in the RGB domain. As our method draws the samples exclusively from $S$, absence of a suitable source will change the color of the output to match the most similar one in $S$ (see Figure 4.13a–c). If this is not acceptable, color matching could be applied or different exemplar images provided.

A similar situation occurs when $S$ contains multiple areas that have similar average intensity and chroma values and are only distinguished by their fine-scale structure. As the filter eliminates this information above a certain width, the distinction between these areas is lost (see Figure 4.13e–f). Synthesis in such situations could be improved if some sort of structural descriptor was taken into account.

When $R$ contains large areas of solid color the algorithm starts to produce artifacts (see Figure 4.13g–i). It will also cause the noise level settings to be ineffectual and hinder synthesized frames from carrying the coherency information between frames. To rectify this, one may add some temporally-coherent texture to the solid areas of $R$ as an overlay, using the workflow described in Section 4.4.

4.7 Conclusion and Future Work

We presented a new framework that allows the transfer of a hand-colored look to 2D and 3D CG animations. Its ability to control the amount of temporal noise provides a new channel of artistic expression, and enables the creation of longer sequences that are less distracting to the observer yet still preserve a lively hand-colored look. We showed that simply varying the strength of the spatial low-pass filter is sufficient to control the amount of perceived temporal noise, and demonstrated that the algorithm can mask visual artifacts in temporally coherent animations. As a future work we plan to extend it to handle more challenging situations (such as automatically distinguishing areas with different fine-scale structure) and extend local noise control to automatically suppress temporal noise in areas with high edge or saliency detector response.
5 LazyFluids: Appearance Transfer for Fluid Animations

http://dl.acm.org/citation.cfm?id=2766983

5.1 Introduction

Figure 5.1: LazyFluids in action—an artist first designs a target fluid animation that consists of a sequence of motion fields (a) and alpha masks (b), and then selects a video exemplar of a fluid element with desired appearance (c) and alpha mask (d). Finally, our algorithm transfers appearance of the exemplar to the target animation while respecting its motion properties and boundary effects (e). The resulting animation can then be used as a part of a more complex composition (f). All alpha masks in the paper are visualised in a way that fully opaque pixels are black and fully transparent are white. Dragon painting © Jakub Javora.

Special effects based on fluid elements are ubiquitous in current digital movie production. To achieve a desired look, an artist typically makes a composition out of pre-captured videos of real fluids with a desired appearance. A key limitation here is that the motion properties of these videos remain fixed. When finer control is needed the artist has to resort to full fluid simulation followed by an advanced rendering algorithm. In this scenario, however, limited resolution and the complexity of material properties, lighting, or other parameters may hinder delivering the desired visual look.

We would like to offer artists a more practical workflow that can narrow the gap between appearance and motion controllability:

1. Quickly design the target animation using 2D CG techniques (e.g., a real-time fluid simulator [Stam, 1999] or particle system [Reeves, 1983]; see Figure 5.1a,b).
2. Pick a photo or a video sequence containing the desired look (the source exemplar; see Figure 5.1c).

3. Add an alpha channel to the source exemplar to distinguish between interior and boundary samples (see Figure 5.1d).

4. Run an example-based synthesis algorithm to transfer appearance from the source exemplar to the target fluid animation (see Figure 5.1e).

Although such example-based workflow can considerably simplify the creation of fluid elements, we found that the current state of the art in flow-guided appearance transfer [Neyret, 2003; Kwatra et al., 2005; Bénard et al., 2013; Browning et al., 2014] does not solve (4), either producing disturbing temporal artifacts or failing to reproduce visual characteristics of real fluid elements.

In this paper we analyze the source of failure in methods originating from Kwatra et al. [2005] and formulate a novel optimization method that addresses it. We extend the formulation to use video exemplars and support rich boundary effects, which are crucial for compelling appearance transfer. Finally, we compare our results with the current state-of-the-art and demonstrate various realistic use cases that confirm the practical utility of our approach.

5.2 Related Work

Texture advection is a common approach to appearance transfer for fluid animations. This technique was pioneered by Max and Becker [1992] and later extended by others [Neyret, 2003; Bousseau et al., 2007; Yu et al., 2011]. Although this simple yet effective solution produces impressive results, its key limitation is that it suffers from notable texture distortion that needs to be alleviated by blending with a new, undistorted texture source. This typically leads to disturbing ghosting artifacts. Another disadvantage is that larger texture exemplars are required to cover longer motions. In situations when the length of motion is not known a priori, procedurally generated textures [Perlin, 1985] or blending multiple textures can alleviate this limitation.

A different approach to appearance transfer, requiring only small exemplars, was presented by Kwatra et al. [2005] and later improved by Lefebvre et al. [2006]. The method has also been extended to work on arbitrary surfaces [Han et al., 2006; Bargteil et al., 2006; Kwatra et al., 2007; Narain et al., 2007]. A great advantage of this technique is that the amount of texture distortion is effectively controlled by a texture synthesis algorithm whose aim is to match the target appearance with the source texture. Although these techniques achieve compelling results on a carefully selected set of sources we found they often fail on exemplars of real fluid elements, generating excessive repetition or flat areas.

Bhat et al. [2004] proposed a different flow-based video synthesis technique that can be classified as a middle ground between texture advection and synthesis. It uses a set of textured particles moving along user-specified flow-lines and leverages video textures [Schödl et al., 2000; Kwatra et al., 2003] to generate infinite or looped sequences from a short video exemplar. The technique can produce compelling results; however, it supports only simple flow fields and requires a suitable video source of the appropriate fluid element.
Figure 5.2: Examples of gradual wash-out. Previous approach to flow-guided synthesis [Kwatra et al., 2005] produce compelling results (b) when applied to exemplars containing repetitive patterns where the size of the repeated element corresponds to the patch size used for the synthesis (a). With a source that contains areas that are comparatively smooth, like blurred areas (c) or low-contrast parts of real fluid elements (e), the results are initially good. But after several frames the resulting animation degrades into a repetitive use of the smoothest patches (d,f) because of an effect described by Newson et al. [2014]. This effect prevails even if more advanced techniques such as (g) discrete solvers [Han et al., 2006], (h) bi-directional similarity [Simakov et al., 2008], or (i) occurrence maps [Kaspar et al., 2015] are used. Our approach (j) resolves this problem thanks to the ability to enforce uniform patch usage. Smoke exemplar © Richard Roscoe.
Our approach bears some resemblance to regenerative morphing techniques \cite{Shechtman2010, Darabi2012} that produce visually interesting transitions between dissimilar images. They were recently applied to stylization of fluid simulations \cite{Browning2014} with compelling results. Nevertheless, a key drawback is that the user must prepare a set of keyframes that roughly match the appearance of the target flow at selected time steps. When synthesizing transitions between these keyframes, these techniques can produce temporal artifacts such as ghosting and pulsation, which break the smoothness of the resulting animation.

Recently, \cite{Benard2013} proposed an extension of image analogies \cite{Hertzmann2001} to synthesize impressive stylized animations guided by a synthetic motion field. Although their approach can be applied in our scenario, we found that it tends to lose high frequency details (both in the interior and at the boundaries) and produces distinct temporal artifacts such as popping and drifting. These are perceived as natural in the context of stylization but feel disturbing when applied to smooth fluid animations.

### 5.3 Problem Formulation

There are two inputs to our method (see Figures 5.10 and 5.11):

- An exemplar of a fluid element $Z$ that can be a single RGBA image or a sequence of $M$ RGBA images $(Z^t)_{t=1}^M$.
- A sequence of $N$ target alpha masks $(X^t_a)_{t=1}^N$ (gray-scale images) with corresponding 2D motion fields $(F^t)_{t=1}^N$.

Source $Z$ represents the desired appearance, $X_a$ captures the shape, and $F$ captures the motion of the target fluid animation. The aim is to transfer the appearance from $Z$ to $X$ in a way that the resulting visual content moves along with $F$, is coherent in time, and respects boundary-specific effects prescribed by $X_a$ and $Z_a$ (the alpha channel of $Z$).

#### 5.3.1 Analysis

\cite{Kwatra2005} addresses a simpler problem with no alpha masks $(Z_a$ and $X^t_a$), treating the source $Z$ as a single RGB image and the target $X$ as a sequence of RGB images. For simplicity in the analysis we do the same and later we extend the formulation to handle alpha masks. In this simplified scenario the problem is formulated as a minimization of the following energy (originally proposed by Kwatra et al. [2005]):

$$E(Z, X^t, \hat{X}^{t-1}) = E_s(Z, X^t) + \lambda E_t(X^t, \hat{X}^{t-1}) \quad (5.1)$$

where $X^t$ is the currently synthesized frame and $\hat{X}^{t-1}$ is the previously synthesized frame, forward-warped using motion field $F^{t-1}$.

The energy (5.1) contains two terms:

- **Source coherence:**

$$E_s(Z, X^t) = \sum_{p \in X^t} \min_{q \in Z} ||x_p^t - z_q||^2 \quad (5.2)$$

This ensures the appearance of the target animation frame $X^t$ is similar to the exemplar $Z$. Here $z_q$ and $x_p^t$ denote source and target patches centered at pixels $p \in X^t$ and $q \in Z$ respectively.
Temporal coherence:

\[ E_t(X^t, \hat{X}^{t-1}) = \sum_{p \in X^t} ||X^t(p) - \hat{X}^{t-1}(p)||^2 \quad (5.3) \]

This ensures the resulting fluid animation changes smoothly in time and moves along with \( F \). Here \( X^t(p) \) denotes color at a pixel \( p \in X^t \).

To minimize (5.1) Kwatra et al. [2005] proposed a method that produces impressive results (see Figure 5.2b) when used with carefully selected sources containing repetitive patterns where the size of the repeated elements is similar to that of the source patch \( z_p \) (Figure 5.2a). However, even a small modification of these sources (Figure 5.2c) can lead to notable degradation (Figure 5.2d). This behavior becomes much more apparent (Figure 5.2f) when \( Z \) is an exemplar with variable content such as realistic smoke or fire elements (Figure 5.2e). These exemplars are not repetitive, and they often contain areas of low contrast. We call this effect gradual wash-out and seek a viable strategy to avoid it.

Bargteil et al. [2006] were the first who noted that the approach of Kwatra et al. [2005] produces gradual wash-out and thought that there was a trade-off between temporal coherence and content preservation. They tried to suppress it by lowering \( \lambda \) in (5.1), i.e., reducing the influence of the temporal coherence term (5.3). This solution slows the degradation down, but it cannot resolve the root of the problem.

In the results of Han et al. [2006], gradual wash-out is also visible although not explicitly addressed. They mentioned a different problem that might seem to be a source of gradual wash-out—that the least-square solver used during the E-step of Kwatra et al.’s algorithm tends to produce blurring artifacts. They tried to alleviate this behavior by using a discrete solver based on k-coherence search. Although their solution can bring some improvement when synthesizing a single image, we found that it makes the gradual wash-out even worse (see Figure 5.2g) and that the source of the problem is actually hidden somewhere else.

An better explanation for this erroneous behavior was recently provided by Newson et al. [2014], who show that during the nearest neighbour retrieval, textured patches are more likely to be matched with smooth ones. We observe that, in conjunction with Kwatra et al.’s algorithm, this effect leads to positive feedback that propagates smoother patches, which come to prevail. If the source \( Z \) does not contain visibly smoother patches, the algorithm tends to pick a patch or a set of patches that are as smooth as possible (e.g., areas with lower contrast) and starts to prefer them. This explains why all the methods originating from Kwatra et al. work only for exemplars consisting of repetitive patterns (see Figure 5.2a) and why they fail for other exemplars (Figure 5.2e).

To avoid a preference for smoother patches Newson et al. proposed using texture features, which resemble feature masks used in [Lefebvre and Hoppe, 2006]. However, a fundamental issue is that it is not clear how to initialize the solution in a way that avoids excessive repetitions of texture features during the synthesis. Newson et al. used inpainting, which is not applicable in our scenario.

Wei et al. [2008] and independently Simakov et al. [2008] proposed a bi-directional similarity (BDS) measure that in addition to source coherence (5.2) uses a new source completeness term:

\[ E_c(Z, X^t) = \sum_{p \in Z} \min_{q \in X^t} ||z_p - x^t_q||^2 \quad (5.4) \]
Its aim is to ensure that all source patches are represented in the synthesized output. Although this extension can bring an improvement, there is a fundamental limitation that the source must be at least as large as the target. If this is not satisfied, an optimal solution would have a few small islands of source patches with the rest filled with repetitions of the smoothest patches. This is the case in our scenario, where appearance exemplars are typically much smaller than the target (see Figure 5.2h).

Kopf et al. [2007] and later Chen and Wang [2010] proposed an approach in which histogram matching of patch colors and offsets is used to bias the optimization towards a solution that penalizes excessive use of a certain subset of source patches. Recently, in concurrent work to ours, Kaspar et al. [2015] extended this technique to explicitly reject patches that have already been used more than twice the uniform usage level. Although these approaches have the potential to suppress the gradual wash-out, they can still lead to a solution with non-uniform patch usage. The key issue is that there is no mechanism to strictly enforce all source patches being used equally in the target, and thus the degradation is still visible (see Figure 5.2i).

5.4 Our Approach

To fully avoid the preference for particular patches that leads to appearance degradation, we need to strictly preserve uniformity of patch usage. We do this by minimizing (5.1) subject to an additional uniformity constraint:

\[ \sum_{p \in Z} \delta(p) = |X| \quad \text{and} \quad \delta(p) - K \in \{0, 1\} \quad (5.5) \]

where \( \delta(p) \) counts the usage of a source patch \( z_p \) centered at a pixel \( p \in Z \) and \( K \) is the floor of the ratio between the number of target \( |X| \) and source \( |Z| \) patches, i.e., \( K = \lfloor |X|/|Z| \rfloor \).

To solve this new constrained optimization we draw inspiration from a concept previously proposed by Rosenberger et al. [2009] that was originally used to optimize BDS in the context of shape synthesis. There are algorithms such as Simakov et al. [2008] that can achieve better BDS. However, in our scenario the aim is not to optimize BDS but to perform synthesis in a way that the uniformity constraint (5.5) is satisfied. For this goal the concept proposed by Rosenberger et al. is more suitable as it provides a mechanism to satisfy uniform patch usage.

In the following sections we first demonstrate how to apply the concept of Rosenberger et al. [2009] in our scenario (Section 5.4.1). Then we extend it to enable rich boundary effects (Section 5.4.2) and temporal coherence (Section 5.4.3). Finally, we propose a joint formulation (Section 5.4.4) which encompasses all mentioned features into one optimization problem and provide two additional improvements (Section 5.4.5).

5.4.1 Nearest-neighbour Field

We made a key modification to Kwatra et al.’s [2005] algorithm to enforce uniform patch usage. We changed how the nearest-neighbour field (NNF) is computed during each iteration of the algorithm. Similarly to Rosenberger et al. [2009] we reverse the direction of NNF retrieval (cf. Figure 5.3), i.e., for each source patch \( z_p, p \in Z \) we find a target
patch $x_q$ that has minimal distance: $D(z_p, x_q) = ||z_p - x_q||^2$. Since we do this search independently, it can happen that two source patches can identify the same target patch as their nearest neighbour. We resolve this collision by keeping the correspondence with the smaller patch distance. Moreover, since the number of patches in $Z$ is usually smaller than the number in $X$ we need to repeat the NNF retrieval until all patches in $X$ have been assigned their counterparts in $Z$ (see Figure 5.3). To make sure every patch from $Z$ is used equally in $X$ we:

1. use a counter $c_p$ that is initially set to 0 and then gradually incremented whenever $z_p$ is assigned to $x_q$ (the white numbers inside circles in Figure 5.3).

2. perform nearest neighbour retrieval only between patches $z_p$ with $c_p < K$ and yet unassigned patches in $X$ (the empty circles in Figure 5.3)

When $|X|$ is not divisible by $|Z|$, i.e., when there is a non-zero remainder $R = |X| \mod |Z|$, the situation becomes more complex. Rosenberger et al. [2009] proposed randomly picking $R$ patches from $Z$ to even up $R$; however, this random pick may bias the solution towards patches that unnecessarily increase the overall energy (5.1). We instead propose a better solution that lets all patches equally participate during the NNF retrieval phase.

During the repeated retrieval we ease the original limitation that only patches with $c_p < K$ can be considered for assignment and also allow patches with $c_p = K$ since some of them need to be used to even up a non-zero $R$. We then sort the list of nearest neighbours candidates $(z_p, x_q)$ in order of increasing $D(z_p, x_q)$ (supposing all colliding...
pairs have been removed) and in this order we perform the following operations for each nearest neighbour candidate \((z_p, x_q)\):

\[
\text{if } c_p < K \text{ then } \quad \text{we assign } z_p \text{ to } x_q \text{ and increment } c_p \\
\text{else if } c_p = K \text{ and } R > 0 \text{ then } \quad \text{we assign } z_p \text{ to } x_q, \text{ increment } c_p \text{ and decrement } R.
\]

This ensures that the uniformity constraint (5.5) is satisfied while letting all source patches participate equally during the NNF retrieval.

### 5.4.2 Boundary Effects

The algorithm described in the previous section assumes that all patches from the source will be used equally in the target. In our scenario, however, we need to make a distinction between the boundaries \((B)\) and interiors \((I)\) of fluid elements (see Figure 5.4). To construct \(I\) and \(B\) we blur the corresponding alpha masks \((Z_a \rightarrow Z_\alpha\) and \(X_a \rightarrow X_\alpha\)) using Gaussian blur with radius \(r\) and apply lower \(l\) and upper \(u\) opacity thresholds. For \(Z\) this yields \(B_Z\): \(Z_\alpha \in (l, u)\) and \(I_Z\): \(Z_\alpha \geq u\) (likewise for \(X\), see Figure 5.4). This segmentation lets us restrict the NNF retrieval so that all patches from \(B_Z\) are matched to those from \(B_X\) and all patches from \(I_Z\) to those from \(I_X\). To enforce uniformity (5.5) in each segment we set \(K = |I_X| / |I_Z|\) and \(R = |I_X| \mod |I_Z|\) for all patches in \(I_Z\) and \(K = |B_X| / |B_Z|\) and \(R = |B_X| \mod |B_Z|\) for all patches in \(B_Z\).

![Figure 5.4: Construction of boundary B and interior I segments in the source Z and the target X. Input alpha masks Z_a and X_a are blurred (Z_\alpha and X_\alpha) and then lower (green curve) and upper (red curve) opacity thresholds are applied to obtain B_Z and B_X, and I_Z and I_X.](image)

To enable the synthesis of a detailed alpha mask for the target we use \(Z_a\) and \(X_a\) as additional pixel channels. During each iteration of the modified Kwatra et al. algorithm [2005] a pixel channel corresponding to the target alpha mask \(X_a\) is modified.
along with the regular color channels $X_r$, $X_g$, and $X_b$. To increase the influence of these additional channels when computing the pixel difference we set their weight to be three times higher than the individual color channels.

For some exemplars a simple boundary/interior distinction might not be sufficient since patches from the outer part of the boundary can still be assigned to the inner part and vice versa. To alleviate this confusion, we add blurred alpha masks $Z_\alpha$ and $X_\alpha$ as additional pixel channels. $X_\alpha$ stays fixed during the synthesis and biases the NNF retrieval so that patches closer to the transition $B_Z \leftrightarrow I_Z$ in the source are more likely to be mapped to the target transition $B_X \leftrightarrow I_X$ and vice versa. We let the user control this bias by adding a special weighting parameter $\eta$. Lower $\eta$ means greater variability in the synthesized boundary effects.

5.4.3 Temporal Coherence

To ensure temporal coherence (5.3) Kwatra et al. [2005] iteratively blend the currently synthesized frame with a forward-warped version of the previously synthesized frame $\hat{X}_t^1$. Unfortunately, this approach works only for static sources since it enforces similarity only to the previously synthesized frame (see Figure 5.5, left). However, we would like to support video exemplars $(Z_t^i)_{i=1}^M$ as well, so we need a different approach.

![Figure 5.5: Temporal coherence in the case of video exemplar—the original approach of Kwatra et al. [2005] (left) enforces temporal coherence by measuring similarity between the synthesized frame $X_t$ and the forward-warped previously synthesized frame $\hat{X}_t^1$. In case of a video exemplar $(Z_t^i)_{i=1}^M$ the synthesis yields incorrect results since the temporal coherence term enforces continuity in spite of changes between frames $Z_t$ and $Z_t^1$. In our solution (right) we search for patches that are independently similar to the source, both in the forward-warped version of the previous frame and the currently synthesized frame.]

Our approach is similar to what we do for boundary effects: we introduce 4 additional RGBA channels into the source $Z_t(p)$ and target $X_t(q)$ pixels that contain values from collocated pixels in previous frames $Z_t^1(p)$ and $\hat{X}_t^1(q)$. The additional RGBA channels influence the overall patch similarity (see Figure 5.5, right) and thus bias the NNF retrieval to prefer source patches whose appearance is close to both the current and the
forward-warped previous frame. For a static exemplar we simply duplicate the content of the regular RGBA channel.

Spatially variable temporal coherence is needed to handle emitters—places where new fluid is spawned. At those places the forward-warping mechanism would incorporate the surrounding empty pixels, but we need to create a new patch of fluid inside the emitter instead. To accomplish this we set \( \lambda_q \) equal to \( \lambda \) for pixels where fluid exists in both the previous and current frames, and set it to zero for pixels where the new fluid appears. These regions can easily be deduced from the current frame mask \( X^t_a \) and the forward-warped mask of the previous frame \( \hat{X}^t_{a} \) (see Figure 5.6) using the following equation:

\[
\lambda_q = (1 - \max(0, X^t_a(q), \hat{X}^t_{a} (q))) \lambda. \tag{5.6}
\]

Figure 5.6: Allowing new fluid to be injected into the domain by locally down-weighting the temporal coherence term. It becomes a spatially varying function \( \lambda_q \) (right), which is zero where the forward-mapped mask of the previous frame \( \hat{X}^t_{a} \) does not overlap the current frame’s mask \( X^t_a \) (left).

5.4.4 Joint Formulation

The extensions proposed in Sections 5.4.1, 5.4.2 and 5.4.3 can now be formally combined into one joint optimization problem. We concatenate all additional channels to form a new enriched source \( \tilde{Z} = (Z^t, Z^t_a, Z^t_{1}) \) and new enriched target \( \tilde{X} = (X^t, X^t_a, \hat{X}^t_{1}) \) (see Figure 5.7). Now the aim becomes minimizing the following energy:

\[
E_s(\tilde{Z}, \tilde{X}, \lambda, \eta) = \sum_{q \in \bar{X}} \min_{p \in \bar{Z}} D(\tilde{z}_p, \tilde{x}_q, \lambda_q, \eta) \tag{5.7}
\]

subject to uniformity constraint (5.5). Here \( \lambda_q \) is the spatially variant weight for temporal coherence, \( \eta \) is the weight for boundary coherence, and \( \tilde{x} \) and \( \tilde{z} \) denote patches with 9 channels per pixel:

\[
\tilde{x} = (x^t_{rgb}, x^t_a, x^t_{\alpha}, \hat{x}^t_{1rgb}, \hat{x}^t_{1a}) \quad \tilde{z} = (z^t_{rgb}, z^t_a, z^t_{\alpha}, z^t_{1rgb}, z^t_{1a}) \tag{5.8}
\]
Figure 5.7: Measuring the distance between a source and target pixel within patches $\tilde{z}$ and $\tilde{x}$. The distance is the sum of 9 terms corresponding to the following pixel channels: 4 for the current frame $([Z, X]_{rgb}^t)$, 4 for the previous frame $([Z, \hat{X}]_{rgb}^t)$, and 1 for the blurred alpha mask of the current frame $[Z, X]_{\alpha t}^t$. Channels $X_{rgb}^t$ (red squares) are modified during the synthesis while the others (green squares) remain fixed. Parameter $\lambda$ is the spatially variant weight that enforces temporal coherence and $\eta$ is the boundary coherence weight.
where $x_{rgb}^t$ denotes the color and $x_a^t$ the alpha mask of the currently synthesized frame, $x_a^t$ is the blurred alpha mask of the current frame, $\hat{x}_{rgb}^t$ is the color and $\hat{x}_a^t$ the alpha mask of the previous frame (likewise for $\hat{z}$). Finally $D$ is the distance measure between patches $\hat{x}$ and $\hat{z}$ (see Figure 5.7):

$$D(\hat{z}, \hat{x}, \lambda, \eta) = ||z_{rgb}^t - x_{rgb}^t||^2$$
$$+ 3||z_a^t - x_a^t||^2$$
$$+ \eta||z_a^t - \hat{x}_a^t||^2$$
$$+ \lambda||z_{rgb}^t - \hat{x}_{rgb}^t||^2$$
$$+ 3\lambda||z_{rgb}^t - \hat{x}_{rgb}^t||^2$$  \hspace{1cm} (5.9)

To minimize (5.7) we use the EM-like multi-scale algorithm described by Kwatra et al. [2005] and Wexler et al. [2007]. The only necessary modifications are that the source and target patches contain pixels with 9 weighted channels (4 are synthesized and 5 are for guidance, c.f. Figure 5.7) and that the NNF retrieval phase is replaced by our method supporting uniform patch usage (see Sections 5.4.1 & 5.4.2). Temporal coherence is implicitly incorporated thanks to additional pixel channels—there is no need to use a special algorithm for controllable texture synthesis as described by Kwatra et al. The first frame is synthesized with $\lambda = 0$ and then we proceed in frame-by-frame order analogous to Kwatra et al. An interesting side-effect of the uniformity constraint is that the algorithm does not require any specific initialization. It is possible to use either the initialization described in Kwatra et al. or any other (e.g., zeroing). The algorithm always comes up with a solution that uses all source patches an equal number of times.

5.4.5 Further Extensions

The method can be extended to handle arbitrarily rotated patches by having the NNF retrieval phase look over the space of rotations in addition to translations. The patch
counting mechanism and NNF construction would remain unchanged. Although such an extension could improve the appearance on some specific flows (e.g., pure rotation in Figure 5.2) it can significantly increase the overall processing time. In practice we use a simpler approximation with notably lower computational overhead, originally proposed in [Lukáč et al., 2013]. We pre-rotate the exemplar by 90°, 180°, and 270° and perform synthesis with this enriched source (see Figure 5.8).

The fidelity of the resulting animation can further be improved using synthetic motion blur. This additional effect can easily be implemented since we know the exact motion field of the target animation; thus we can perform line convection using anisotropic Gaussian filter in a direction and length given by the target motion field (see Figure 5.9).

5.5 Results

We have implemented our technique in C++. To achieve feasible performance we accelerated the NNF retrieval phase described in Section 5.4.1 using PatchMatch with integrated support for masking [Barnes et al., 2009]. To further accelerate this performance bottleneck we used a parallel tiled algorithm [Barnes et al., 2010]. With this optimization it usually takes around 5 minutes to synthesize one 1Mpix frame on a 3GHz CPU with 4 cores.

To prepare data sets we implemented a custom fluid simulator [Stam, 1999] that allows an artist to quickly design desired fluid animations in real-time. Using this tool we created 7 different fluid animations with different motion properties to be used as targets, and selected 10 different fluid elements (7 static images and 3 videos) with variable appearance and complexity to be used as exemplars. To produce the results we used the following parameter setting: patch size $5 \times 5$, temporal coherence $\lambda = 0.2$, richness of boundary effects $\eta = 3$, radius for Gaussian blur: $r = 3$ (this setting might change according to the
Figure 5.10: Results and comparison: various fluid exemplars consisting of color $Z_{rgb}$ and alpha mask $Z_a$ (both static images) were used to synthesize a target fluid animation $X$ specified by a sequence of flow fields $(F^t)_{t=1}^{180}$ and alpha masks $(X^t_a)_{t=1}^{180}$ (showing only a single frame, see supplementary video for the whole sequence). Note how our approach produces convincing results despite varying source complexity. Fire blast exemplar (2nd column) © Corteck via flickr, volcano smoke exemplar (5th column) © Arnar Thórisson.
resolution of the target animation), and thresholds: \( l = 0.1, u = 0.9 \) to extract boundary region from of the blurred target alpha channel.

First we synthesized a single fluid animation using multiple exemplars (see Figure 5.10). Our approach produced convincing results despite notable visual differences between individual sources. For the fire exemplars in Figures 5.1 and 5.11 we compared the static source to the video source. From the comparison (see supplementary video) it is visible that our approach produces convincing results for both cases. The advantage of the video source is that it nicely enhances the motion richness of the target animation.

We also compared our approach with the original Kwatra et al. [2005] method as well as Bénard et al. [2013] and Browning et al. [2014], two current state-of-the-art techniques in flow-guided appearance transfer. For each method we used the same fluid animation as well as fluid exemplars as in Figure 5.10 to enable side-by-side comparison. Since Kwatra et al. does not natively support synthesis with boundary effects we decided to illustrate its behavior by using our own approach, but with the original NNF retrieval phase—not our improved method that enforces uniform patch usage. Setup for Bénard et al. was very close to our method. For style input \( S \) we used the blurred alpha channel of our exemplar \( Z_a \), for style output \( \hat{S} \) we used the RGB channels of our exemplar \( Z_{rgb} \), and finally for input image \( I \) we used our blurred target alpha mask \( X_a \). For Browning et al. we had to prepare a set of keyframes (every 10th frame) that were synthesized using our approach as independent frames (i.e., we set \( \lambda = 0 \)).

We ran these methods on all 180 frames of the test sequence in Figure 5.10. Static frames from this sequence are presented in Figure 5.13 and complete videos are in supplementary materials. It is clear from the results that the original NNF retrieval used by Kwatra et al. produces severe gradual wash-out and thus quickly diverts the appearance...

---

**Figure 5.11:** Synthesis with a video exemplar: input flow fields \( F \) with alpha masks \( X_a \) and fluid exemplar \( Z_a \) and \( Z_{rgb} \) was used to synthesize fluid animation \( X \). The fireball sequence \( (F^t)_{t=1}^{190} \) was used to compare a static source and a video source \( (Z^t)_{t=1}^{120} \) (see supplementary video for the comparison).
of the resulting sequence from the appearance of the exemplar. In the results of Bénard et al. the gradual wash-out is also visible, however, it is not as apparent thanks to the offset histogram matching used in the original method. However, several disturbing temporal artifacts are visible such as high-frequency popping, and the boundary effects are not as detailed as in our results. The results of Browning et al. exhibit the closest appearance to the exemplar. There is no gradual wash-out visible since the method relies on frequent keyframes that were synthesized using our method. However, the results show ghosting and a disturbing pulsation effect that is more visible in the video.

Finally, to further demonstrate practical utility of our approach we produced 4 compositions where different fluid animations were synthesized using realistic fluid exemplars and then combined with an environment to create a desired visual effect (see Figures 5.1 and 5.12 and supplementary video). Such practical results were impossible to achieve using previous techniques as they either suffer from gradual wash-out or produce disturbing temporal artifacts.

5.6 Limitations and Future Work

We found that in practice our method performs very well. It is quite robust, being agnostic to parameter settings, type of input fluid simulation and the choice of exemplar. Nevertheless, there are several limitations that we would like to mention.

In cases where the source exemplar is substantially bigger than the target, and when it contains large areas with noticeably smoother or lower contrast patches, our technique cannot always fully avoid gradual wash-out since the harmful preference for smoother patches [Newson et al., 2014] can still prevail. This is a challenging situation that has great potential for further investigation.

Our technique does not take into account additional flow-field parameters such as curvature, divergence or Jacobian. Those can be computed by the simulation and can be used to guide the appearance of synthesized texture locally (like in [Narain et al., 2007]) to better convey realistic physical properties of the given fluid element. Similar limitations hold for shading, self shadowing, or volumetric effects. Those are baked into the exemplar for some particular lighting conditions, but they might not be realistic in a different environment. Both issues are good candidates for follow-up work.

Since our algorithm synthesizes the i-th target frame using the i-th frame of the video exemplar we need the video exemplar to have at least as many frames as the output sequence. If it is shorter, some additional looping is necessary to avoid hard jumps in appearance. Techniques that produce looped sequences from unlooped footage [Schödl et al., 2000; Liao et al., 2013] can help the user prepare input data in this case. A related issue is motion in the input sequence. When its direction or speed do not match the target animation our method can produce unsatisfactory results. In this case some additional stabilization of the source would be necessary.

5.7 Conclusion

We have presented a novel approach to appearance transfer for fluid animations. Our technique is the first that performs flow-guided texture synthesis that convincingly preserve the appearance of realistic fluid exemplars, while at the same time avoiding dis-
Figure 5.12: A more complex composition created by an artist using our system (top). Three different flow fields $F$ with alpha masks $X_a$ and exemplars $Z_{rgb}$ with alpha channels $Z_a$ (bottom) were used to synthesize the resulting fluid animations. Firepit painting © Jakub Javora.
turbing temporal artifacts. We have demonstrated the practical utility of our technique in realistic scenarios, giving practitioners a new option for creating fluid-based special effects. We also believe that our novel constrained formulation, which ensures uniform patch use, should inspire future research that will improve the quality of patch-based texture synthesis and related image editing techniques.
Figure 5.13: Results and comparison (see text for details): a single frame (No. 135) extracted from a longer sequence synthesized using [Kwatra et al., 2005], [Bénard et al., 2013], [Browning et al., 2014], and our approach on five different exemplars of fluid elements (shown in the bottom right corner in each view). The top left corner shows a detail of the synthesized result. Fire blast exemplar (1st row) © Corteck via flickr, volcano smoke exemplar (4th row) © Arnar Thorisson.
6 Brushables: Example-based Edge-aware Directional Texture Painting


6.1 Introduction

Figure 6.1: Examples of images synthesized using our method (right) generated from various sources (left). Our method simultaneously produces meaningful boundaries and interior structures, with textural features that respect the shape and direction specified by the user. Source credits: denim: inxti @ shutterstock; plank: My Life Graphic @ shutterstock; grass: varuna @ shutterstock; cookie: Alessandro Paiva @ rgbstock

Example-based image synthesis enables transfer of visual characteristics from a given
exemplar to a user-defined target image [Ashikhmin, 2001; Hertzmann et al., 2001]. In this context a texture-by-numbers metaphor is typically used to guide the transfer of textural information between specific locations in the source and target images. Ritter et al. [2006] showed that the quality of the synthesis can be improved when the algorithm takes into account specific effects that occur close to the boundaries of individual segments. This edge-aware approach was recently improved by Lukác et al. [2013] who showed how to synthesize boundary effects in a direction-aware manner independently from the interior—the direction of synthesized boundary features exactly follows the direction of the boundary shape. Direction awareness was also previously used in general texture synthesis [Zhang et al., 2003; Lefebvre and Hoppe, 2006; Eisenacher et al., 2008; Diamanti et al., 2015] to allow control of the orientation of the synthesized texture.

A key limitation of existing techniques is that they treat edge and direction awareness independently, making it hard to produce images where the prescribed directionality of the shape’s interior interacts with the appearance of its boundaries.

In this paper, we propose a novel method for interactive example-based image synthesis that combines edge and direction awareness in a single algorithm. While these features are useful independently to synthesize textural areas and linear edge structures with user-specified orientation, combining them enables complex shape-aware effects that no previous method can handle. See, e.g., examples in Figure 6.1 where the appearance of boundaries (e.g., blades of grass or hair ends) depends on a specific context that is given by the directionality of the interior.

Our method builds upon the popular patch-based optimization scheme originally developed by Wexler at al. [Wexler et al., 2007] and later extended by others [Barnes et al., 2009; Darabi et al., 2012; Kaspar et al., 2015]. A key contribution of our work is that we provide a new extension of the original Wexler et al. formulation that combines both direction and edge awareness into one optimization problem. We further improve the visual quality of the synthesized result using a novel coherence weighting mechanism. We also propose a unified interactive framework that helps the user prepare the necessary input data for the synthesis. We extend previous related techniques for detecting [Kang et al., 2009; Kyprianidis, 2011] and authoring [Zhang et al., 2006; Fisher et al., 2007] direction fields by creating a new signed direction field. The sign was not considered previously, and we show that it helps the user specify semantically meaningful configurations where unsigned orientation fields are insufficient (see, e.g., direction of hair/grass growth in Figure 6.1).

### 6.2 Related Work

One of the first instances of combining texture synthesis with a painting interface was Synthesizing Natural Textures [Ashikhmin, 2001]. The user painted an output suggestion in the color domain, and the synthesis created output that roughly matched the colors. However, color information was not enough to finely specify texture areas.

Image Analogies [Hertzmann et al., 2001] alleviated this limitation with a texture-by-numbers approach. The user pre-segmented the input image and directly painted a segmentation mask. However, the lack of additional information about boundary orientation led to visible inconsistencies.

Painting with Texture [Ritter et al., 2006] represented a further development in this
area. It was the first approach explicitly designed for synthesizing stroke interactions and texture edge effects by introducing a shape mask into the patch distance term. The mask provided rudimentary edge awareness, but its small size could not represent subtle orientation changes and larger sizes would make the synthesis over-constrained, causing visible repetitions and other artifacts.

Painting by Feature [Lukáč et al., 2013] presented an improvement over the previous techniques by treating lines and edges separately from the interior texture. Instead of relying on pre-segmented input images, the user interactively selected a line feature or a texture to be used as an example and then painted them into the output canvas. Despite full creative freedom, this technique could become tedious, requiring painstaking boundary tracing even when edges were obvious. This method also did not provide an explicit control over the directionality of the texture in the interior regions.

RealBrush [Lu et al., 2013] is a canonical example of stroke synthesis, capable of transferring the directionality and edge effects of the input strokes directly to the result using a painting metaphor. However, since it uses a lengthwise cut-and-stitch approach instead of full synthesis, it is strictly limited to 1D curves and cannot synthesize arbitrary area structures. Other stroke synthesis systems [Lu et al., 2014; Zhou et al., 2013] typically suffer from the same limitation.

Accounting for directionality in texture synthesis is a proven idea. It can compensate for transformations [Eisenacher et al., 2008; Lefebvre and Hoppe, 2006] or allow specification of direction in textures [Zhang et al., 2003; Diamanti et al., 2015]. Detecting orientation in images is also crucial for various stylization techniques [Hays and Essa, 2004; Kang et al., 2009; Kyprianidis, 2011]. However, a painting scenario such as ours requires further considerations. The direction fields should be authored seamlessly using the basic brush metaphor, and the detection needs to be configurable to ensure compatibility of input and output direction fields.

Structure tensors [Brox et al., 2006] and edge tangent flow [Kang et al., 2007] are common techniques to detect orientation in textures. Their key limitation is that they cannot provide consistent direction: the orientation sign is either omitted or inconsistent in the final solution. However, this is crucial in our scenario because real textures typically contain asymmetric structures. Although there are techniques that try to find consistent direction [Kang et al., 2009; Xu et al., 2009], they typically fail on larger scales or when singularities are present in the input field.
User-guided authoring of vector fields has been extensively studied in the context of 3D surfaces [Zhang et al., 2006; Fisher et al., 2007; Crane et al., 2010; Maharik et al., 2011]. Although these techniques compute smoothly varying vector fields from a sparse set of user-provided constraints, their main drawback is that every new constraint has a global impact on the resulting field. In our scenario we would like to modify the existing field on-the-fly by adding new directional strokes whose local impact is controlled by the user.

Optimization-based texture synthesis methods [Kwatra et al., 2005; Wexler et al., 2007] are the current state of art for synthesis applications [Darabi et al., 2012; Fišer et al., 2014; Kaspar et al., 2015]. They accurately reproduce exemplar structures at interactive rates, thanks to fast approximate nearest-neighbor search [Barnes et al., 2009]. We take advantage of the flexibility of this framework to introduce edge and direction awareness, and make adaptations to mitigate artifacts introduced by the free-form nature of our scenario.

For edge awareness, we build upon shape descriptors, commonly used in computer vision [Belongie et al., 2002; Berg and Malik, 2001]. They examine large areas of the shape to properly consider context, which is computationally expensive. Because texture synthesis requires numerous evaluations in a short time-frame, this can be a bottleneck.

### 6.3 Our Approach

Figure 6.2 illustrates the workflow of our method. The user starts with a regular RGB image and uses interactive image segmentation and matting to extract the area of interest along with the opacity values. The resulting RGBA image source $S$ then serves as the basis for further processing (see Figure 6.2a).

Initially we take all pixels in $S$ with non-zero alpha to form a binary shape mask $M_s$ and then let the user determine the edge extent, i.e., how wide the boundary effects are. Finally, we employ direction analysis to obtain a source direction field $d_s$ with a desired level of smoothness and consistent sign of the tangent vectors (see Section 6.3.1).

In the following painting phase, the user uses a brush tool to paint a mask that defines the set of pixels to be synthesized—the direction field $d_t$ and its shape mask $M_t$. Then the use can alter or refine the target direction field (the combing process). For this our novel direction diffusion algorithm (Section 6.3.2) gives precise control over the stroke extent, seamlessly combining multiple strokes, and combining new strokes with the pre-existing direction field.

Finally, given the source image $S$, source and target masks ($M_s$ and $M_t$) and direction fields ($d_s$ and $d_t$) we run our direction- and edge-aware texture synthesis (Section 6.3.3). We synthesize the output texture, using a novel Shape Hint to ensure that boundary effects are synthesized appropriately in a context-sensitive way, enforcing the prescribed direction, and using a coherence weighting mechanism to improve the final visual quality of the synthesized image $T$ even under strong non-rigid deformation.

### 6.3.1 Direction Analysis

Before we can paint taking the directionality of the source into account, we need to estimate it. Our first step is to create a direction field $d_s$ that specifies local direction at
all pixels of $S$. To support arbitrary input exemplars and have a self-contained approach, we determine the direction field using only the RGB color information.

For best results, a reasonable direction field $d_s$ should be locally smooth and perpendicular to the gradient field of $S$—a tangent field. Because smoothness and perpendicularity cannot usually be satisfied simultaneously, additional filtering is required. We also take the sign of the tangent vectors into account, since they are often semantically significant.

Estimation of smooth tangent fields has been explored before, predominantly in image stylization techniques [Kang et al., 2007; Kyprianidis, 2011]. However, these approaches typically ignore the sign of the tangent vector, since the filters they ultimately employ are symmetric with respect to the sign. In particular, the multi-lateral filter employed by Kang et al. uses a non-linear term to preserve the sign of the tangent vectors. In such case, flipping the signs of some of the tangents in the initialization phase will not affect the magnitudes or absolute direction of the tangents in the resulting tangent field, merely their signs. This means that we can solve sign harmonization independently as a pre-processing pass and then apply one of the filters to get a coherent, smooth result.

We base our sign harmonization on the edge tangent flow (ETF) filter [Kang et al., 2007], but we improve the initialization. We fix the sign of the pixel with the greatest gradient magnitude and use a breadth-first propagation to harmonize the tangents along with their signs. Figure 6.3 shows how this method eliminates the 180-degree discontinuities present in earlier methods.

### 6.3.2 Direction Diffusion

In previous approaches [Darabi et al., 2012; Lukáč et al., 2013], the texture direction emerges implicitly from the color domain so as to match the boundary conditions. In contrast, we give the user explicit control over texture direction, much like stroke synthesis approaches do [Zhou et al., 2013; Lu et al., 2014]. Given a region painted by the user with a variable-width brush, we determine the direction field $d_t$, assigning a direction to every pixel in the region.

Like stroke synthesis, we are given a user-specified 1D stroke path with an instantaneous direction at every path sample. We propagate the sparse direction samples to the entire stroke area. As an act of painting, the effect of brushing should be local, with its...
influence strictly limited to the area within the brush footprint leaving the rest of the image unaffected. To avoid synthesis artifacts, we also must ensure that we do not create discontinuities in the direction field at the brush boundary and that the target direction field has a similar level of smoothness to the direction field of the input.

Related approaches use various optimization processes to construct a smooth direction field from sparse user-specified constraints [Zhang et al., 2006; Fisher et al., 2007]. However, these techniques are global by nature and do not provide for a localized, controlled way to combine new strokes with an existing direction field, which is needed to permit *combing* and general refinement. We use a kernel-based diffusion scheme to smoothly diffuse and blend the direction of an arbitrary number of strokes of variable radius, while also permitting blending with a pre-existing field.

Given a stroke path $K$ consisting of all points $k \in K$, we calculate the direction $d_k(p)$ diffused from this stroke at a point $p$ as follows:

$$d_k(p) = \frac{1}{w_k(p)} \int_{k \in K} G(||p - k||^2, \sigma_k^2) \cdot d'(k)$$

(6.1)

where $G(x, \sigma^2)$ is a gaussian kernel with the standard deviation set to half the stroke width, $d'(k)$ is the local normalized tangent, and

$$w_k(p) = \int_{k \in K} G(||p - k||^2, \sigma_k^2)$$

(6.2)

This yields a smooth interpolation that can be evaluated analytically if the input stroke is approximated as a polyline, and the generalization to multiple simultaneous strokes is straightforward (see Figure 6.4a). If we need to combine the diffused direction of the current stroke with the aggregated direction field of all the previous strokes (as in Figure 6.4b), we calculate the convex mix of the previous value $d^{n-1}(p)$ with the new one $d_k(p)$ like so:

$$d^n(p) = w_s(p) \cdot d_s(p) + (1 - w_s(p)) \cdot d^{n-1}(p)$$

(6.3)

assuming $w_s(p)$ is clamped to remain in the convex interval $(0, 1)$.

![Figure 6.4: A demonstration of direction field authoring and refinement. (a) a composition of thick strokes made with a 120px wide brush next to its synthesis result; (b) direction field with two 80px refinement strokes and the refined synthesis result.](image)
6.3.3 Example-based Synthesis

Once source and target direction fields $d_s$ and $d_t$ are prepared we proceed to the synthesis phase, generating the output image while respecting the principles of edge and direction awareness we have described earlier.

We build our synthesis algorithm upon established the patch-based optimization framework introduced originally by Wexler et al. [2007]. We chose this framework for its flexibility: we can substantially alter its behavior by substituting our own patch distance measure and patch voting logic, making it fit our own requirements.

We introduce edge-awareness into the synthesis by adding a new shape distance term to the energy function we minimize:

$$E(T, S) = \sum_{q \in T} \min_{p \in S} \left( D_{\text{patch}}(p, q) + \lambda D_{\text{shape}}(p, q) \right)$$ (6.4)

$D_{\text{patch}}(p, q)$ measures the color distance of patches and $D_{\text{shape}}(p, q)$ the distance of local shapes around pixels $p \in S$ and $q \in T$.

Direction awareness is added to these distance measures by taking local direction at both $p$ and $q$ into account. We do this by introducing a rotation operator $\otimes \alpha_{pq}$, which rotates the local frame of reference for the patch or shape descriptor by the difference in local direction at $p$ and $q$, i.e., $\alpha_{pq} = d_t(q) - d_s(p)$.

We can then calculate the color distance as the direction-aware sum of squared differences:

$$D_{\text{patch}}(p, q) = \left| \left| \mathbf{P}^s_p - \mathbf{P}^t_q \otimes \alpha_{pq} \right| \right|^2$$ (6.5)

between the source patch $\mathbf{P}^s_p$ centered on $p \in S$ and the rotated target patch $\mathbf{P}^t_q$ centered on $q \in T$. Similarly, the direction-aware shape distance is evaluated as:

$$D_{\text{shape}}(p, q) = \chi^2 \left( \mathbf{H}^s_p, \mathbf{H}^t_q \otimes \alpha_{pq} \right)$$ (6.6)

i.e., the distance between source and target Shape Hint histograms described below, which introduce shape awareness by considering both the spatial distance from the texture boundary, and its shape relative to the local direction field.

**Shape Hint** To introduce edge-awareness into the synthesis we use Shape Hints—a local shape descriptor derived from the shape context [Belongie et al., 2002], which we have simplified and adapted for interactive use. Shape descriptors like these are a powerful tool commonly used to find similar locations within shapes. Compared to previous context aware solutions based on a distance transform [Lefebvre and Hoppe, 2006; Bénard et al., 2013], a shape descriptor considers a larger context, allowing it to distinguish between locations at edges and corners or around interior holes; this is vital for our concept of edge-awareness, since it lets us pick patches from appropriate regions more contextually (see Figure 6.5). It is also more flexible than comparing mask patches, as in Painting with Texture [Ritter et al., 2006]; the distance measure is continuous rather than discrete and degenerates gracefully, without overconstraining the synthesis at texture edges.

Like the shape context, our descriptor counts the edge pixels that fall into “bins” mapped to image space. These counts are then treated as histograms of edge pixels and can be compared using the $\chi^2$ metric. This creates a descriptor that is capable
of capturing the shape of the object boundary with a configurable level of tolerance to high-frequency variations, based on how large is the spatial support of the bins. To date, performance considerations precluded the use of shape context in texture synthesis, as typically local descriptors need to be evaluated repeatedly at many points of the image, and the computational complexity of evaluating a shape context scales quadratically with its radius.

To overcome this limitation, we propose an adaptation wherein we change the shape of the bins used to count edge pixels (see Figure 6.6). Instead of annular sections, we use circular bins, similar to image descriptors like FREAK [Alahi et al., 2012], but we keep the shape context’s compact representation based on edge pixel counting and its method of calculating similarity.

As circles are rotationally invariant, the shape of the bin becomes constant with respect to both the orientation of the descriptor as a whole and the bin’s position therein. The value of any bin at any point can thus be pre-computed by convolving the edge pixel map with a disc filter of the appropriate radius, and consequentially, we can evaluate the descriptor with a constant number of bitmap queries regardless of its spatial support or the number of edge pixels in the image.

This not only leads to faster evaluation, making use in texture synthesis possible, but also permits free-form continuous rotations of the descriptor at no additional computational cost. We have found it sufficient to only use a single radial layer of bins, although the descriptor naturally generalizes to multiple layers.

In synthesis, we use the source and target masks $M_T$ and $M_S$ to calculate the Shape Hints, with the radius of the descriptor set to the \textit{edge extent} that the user defined earlier in the source analysis phase. This value should roughly correspond to the width of the boundary effects, i.e., how “deep” into the texture they extend. Content within this range is implicitly treated as the boundary, while content deeper inside is considered to be in the interior.
CHAPTER 6. BRUSHABLES

Figure 6.6: Shape and arrangement of bins in an oriented single-layer shape context (left) and in our Shape Hint (right).

Alpha Channel To further improve the quality of the synthesis at boundaries we add an alpha mask as an additional pixel channel. This has two effects. It lets us synthesize opacity, and together with the Shape Hint, guides the synthesis towards a solution where pixels close to boundaries in the source are more likely to be matched with boundary pixels in the target. To give the opacity comparable weight to color we multiply the difference in alpha channel by 3 when computing the sum of squared differences in (6.5).

While the alpha channel gives us the ability to synthesize opacity and “fading out” at the boundaries, it is in itself not sufficient to capture longer-range edge effects, and cannot discriminate boundaries from the interior in textures with partially transparent interiors. Therefore, a combination of alpha channel synthesis and shape matching is optimal for synthesis of edge effects in our scenario.

Optimization To minimize (6.4) we use the Expectation-Maximization optimization outlined by Wexler et al [2007] that consists of alternating search and voting steps on an image pyramid in a coarse-to-fine order. To improve texture coherence and richness in the synthesized image, we propose an improvement to the voting step to take both local nearest-neighbor field coherency and the color histograms of both images into account. When evaluating the final color $C(p)$ of a pixel $p$, we iterate through the overlapping patches mapped to its neighborhood and perform a weighted average of the candidates $c_x$ gathered from them:

$$C(p) = \frac{\sum_{q \in N_p} w_c(q) \cdot w_h(q) \cdot C(q)}{\sum_{q \in N_p} w_c(q) \cdot w_h(q)}$$  

(6.7)

where $w_h$ is the color histogram weight of the candidate pixel, as detailed by Kopf et al [Kopf et al., 2007] and $w_c$ is the coherence weight, which serves to propagate coherent arrangements of patches from the source. As described in the original paper,
the histogram weight promotes pixel candidates with relatively underrepresented colors, improving the diversity of the synthesized image.

The coherence weight is vital in our scenario, since free-form rotations of the texture tend to induce non-rigid mapping in the nearest-neighbor field, which in turn causes blurry and visually displeasing results (see comparison in Figure 6.7). By increasing the weight of coherently-mapped configurations of patches, we encourage forming larger, coherently mapped areas over multiple iterations. This preserves high-frequency detail and causes less blurring.

![Comparison of results with and without coherence](image)

**Figure 6.7**: A comparison of results (a,c) and corresponding nearest-neighbor fields (b,d) synthesised with (left) and without (right) the coherence weight. The details (e,f) show how structural details of individual blades are better preserved with the coherence weight. Note also how the coherence weight leads to larger patches in the nearest-neighbor fields.

To calculate the coherency weight, we examine the coherence of mapped pixel configurations as follows:

In effect, a nearest-neighbor match is a rigid mapping from $T$ to $S$. The matched coordinates and relative rotation at a pixel $q$ thus define a mapping $R_q$, which maps the pixel grid in $T$ to a rotated and offset pixel grid in $S$. Because the optimization is based on the assumption that these mappings are approximately identical for the group of pixels within the area of a patch, we design our coherency measure as a quantification of how this assumption holds. To evaluate this measure, we examine the patch neighborhood of
a pixel $q_0$ and the induced mappings therein (c.f. Figure 6.8):

$$w_c(q_0) = \sum_{q \in N_{q_0}} G(||R_{q_0}(q) - R_q(q)||^2, \sigma_c^2)$$  \hspace{1cm} (6.8)

where $\sigma_c^2$ is the coherency range, which we set to 2 throughout.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6_8.png}
\caption{Calculating the coherence of a patch: we examine all pixels $q$ in a patch around the pixel $q_0$ in the target image $T$. The position of pixel $q$ is projected into the pixels $p$ and $p'$ in the source image $S$ using both the rigid transformation induced by the match at $q_0$: $p = R_{q_0}(q)$ and its own transformation: $p' = R_q(q)$. The more coherent the matching is, the lower the sum of distances $||p - p'||$ (red arrow).}
\end{figure}

Multiplying these weights, along with the guaranteed range on both of them, ensures that the weighting scheme degenerates gracefully in any edge case.

### 6.3.4 Implementation Details

We have implemented the described algorithm in C++11 and run it on a desktop computer. The synthesis takes approximately 5 seconds for a megapixel output image, with 80% of the time spent calculating the nearest-neighbor field. The texture analysis step was more computationally intensive, taking up to 30 seconds for larger settings of the ETF filter; however, this only needs to be done once for each source texture as a preprocess, and the results can be efficiently stored. The rest of the method operates interactively.

As most of our parameters have intuitive semantics, they were set contextually as appropriate. The range of the edge tangent flow filter was usually set to a default value of 10 pixels. This setting was only increased for noisier textures to approximately 35 pixels. $\lambda$ in Equation 6.4 was hardcoded to a value of 25 (equal to the number of pixels in a patch used to measure color distance), and the width of the brush was interactively adjusted as appropriate.
Edge extent was the crucial parameter to achieve edge awareness; setting it too low can cause boundary patches to be randomly used in the interior, while setting it too high can cause the extent of the synthesized texture to visibly deviate inwards from the user-specified mask. Ultimately, the value in pixels should be set approximately equal to the width of the boundary effects the user wishes to capture.

6.4 Results

Figures 6.1 and 6.14 shows the synthesis results of a variety of natural textures. Our approach coherently synthesizes textured outputs with direction configurations not present in the original source (see e.g., the crochet results). Plank example demonstrates how our algorithm picks the semantically correct edge features according to local direction. The grass and the colored pencil examples show that even transparency is synthesized correctly both on the boundary and in the interior. The braided wig example serves to highlight the strength of shape descriptor-based edge awareness; the narrower sections are synthesized from braid patches, while the wider parts are synthesized out of the upper, combed part of the exemplar. Again, current approaches do not have such capabilities. The ornamental leaves are an example of a relatively simpler stroke synthesis application. It shows how our approach organically synthesizes branching by virtue of not relying on stroke semantics. The red wig shows how locally-variant anisotropic textures can be coherently deformed to novel configurations.

Note that after texture analysis, the only user input we require are the brush strokes.
Thus, our tool places no more burden on the end user than a regular brush. The overall interaction takes only a couple of seconds, depending mainly on the ability of the user to draw individual strokes (see supplementary video for an example of interactive sessions). This brings an improvement over Painting by Feature [Lukáč et al., 2013], which requires more elaborate input to achieve similar results (see Figure 6.9).

Our method can also be easily extended to process images with multiple, segmentable textures (see Figures 6.10 and 6.11). In this case, we require that a segmentation map be provided for the input texture, and the direction brush is concurrently used to paint also the output segmentation. The synthesis is then adjusted so that it only maps patches between compatible segments. Segment boundaries are considered in the same way as foreground boundaries for the Shape Hint.

Multiple texture extension allows us to make a comparison with Painting by Feature [Lukáč et al., 2013] (see Figure 6.11 and supplementary material). Our method produces comparable or better visual quality without the necessity to use a custom synthesis algorithm for the boundaries. It also notably improves the look of the interior parts by maintaining the appearance of the original source and creating seamless transitions from the edge that follows prescribed direction field.

Our method can also be used to synthesize example-based brush strokes of comparable quality to those produced by RealBrush framework [Lu et al., 2013] (see Figure 6.12). In addition, the same algorithm can be applied to fill larger areas, which the original RealBrush method cannot do.

In Figure 6.13 we show results where only the edge or direction awareness is taken into account. This example demonstrates limitations of previous approaches (such as [Ritter et al., 2006] or [Lefebvre and Hoppe, 2006]) where a joint edge- and direction-aware formulation was not considered.

### 6.4.1 Limitations

Our algorithm does not automatically take changes in texture scale into account, nor does it natively compensate for perspective. Support for these could be added by preprocessing the input image to compensate for these.

Because our algorithm does not take advantage of any domain knowledge, it cannot replicate stroke-specific effects that require such knowledge. Most significantly, the
Figure 6.11: Brushables can also be used in the RealBrush scenario [Lu et al., 2013]. Our approach can synthesize new strokes like RealBrush can, and also synthesize regions of arbitrary shape.

Figure 6.12: Results from Figure 6.11 with the shape awareness and direction awareness turned off. Those examples demonstrate importance of joint formulation proposed in our framework and illustrate limitation of previous approaches, which take into account only direction [Lefebvre and Hoppe, 2006] or edge [Ritter et al., 2006] awareness.
Figure 6.13: Comparison of our approach with Painting by Feature (PBF) [Lukáč et al., 2013] with respect to ability to handle edges with highly varying width. Note how our unified approach integrates interiors with edges smoothly, whereas in the PBF output there are discontinuities between the areas synthesized as edges and those synthesized as interiors (see red arrows). In the bottom example, PBF is able to more closely match the user-specified shape, but does so at the cost of faithfulness to the example and visual richness. Source credit: monkey: © ACM; hedge source: Joe Shlabotnik @ flickr
smudging and smearing effects supported by RealBrush [Lu et al., 2013] cannot be replicated. Instead, overlapping strokes merge into a single larger area and are synthesized as such (see Figure 6.11 right).

Furthermore, because we rely on an area representation rather than an outline-based one, our approach does not natively handle interior lines like Painting by Feature [Lukáč et al., 2013] does. This effect could be emulated by selecting the line in the example as a separate texture, painting that and combining the results. Still, the nature of our brush-based interaction model makes this less convenient than similar operations are for vector-based tools.

In some textures, there may be hidden variables not related to direction that affect incidence of features both on the edges and on the interior; these might include e.g. the holes in the cracker, or the precise position of the hairband in the braid. In such cases, our approach is unable to distinguish the underlying semantics and will distribute these features randomly. The ability to specify manual constrains, such as the ones used in appearance-space texture synthesis [Lefebvre and Hoppe, 2006], could allow the user to resolve these cases.

Our algorithm also exhibits some of the artifacts of the original synthesis method of Wexler et al. [Wexler et al., 2007], namely the repetition of textural features. Extensions to this optimization scheme that eliminate these have been proposed [Kaspar et al., 2015; Jamriška et al., 2015]; we consider these to be orthogonal to, and compatible with, our work.

6.5 Conclusion and Future Work

As discussed above, our approach handles complicated natural textures using a simple mode of interaction demonstrated earlier. Adding direction awareness to the synthesis process lets us handle textures with locally-variant anisotropic properties without requiring large exemplars or losing information. Our direction detection and authoring framework give users control over the output direction field that is semantically significant for many textures.

Adding the shape hint to texture synthesis enables robust handling of edge effects. Combined with alpha-channel synthesis, our approach can reproduce edge effects present in partially transparent textures. As a result, edges need not be explicitly drawn by the artists any more.

When combined, these two features become even more powerful, allowing semantically significant edge areas to be used for synthesis in different places. This allows artists to use previously unaccessible textures for true interactive texture painting.

For future work, we would like to better handle the cases where the direction configurations in the source do not match the target direction field. One possible solution is to automatically adapt the target direction field in a constrained and meaningful way. Furthermore, we would like to experiment with our shape hint in the domain of shape synthesis. It might be able to give rough user sketches the same type of high-level detail that a source shape does. Another possible avenue is to synthesize the mixing of textures using a blending approach like Image Melding [Darabi et al., 2012].

Our approach integrates naturally into digital painting pipelines, thanks to its intuitive mode of interaction. Its ability to handle painting media exemplars lends itself to the
Figure 6.14: Various sources (top): cracker, crochet, denim, sample of color pencil, bread, red wig, braided wig, ornamental leaves, plank, and grass were used to synthesize target images (below). Note how our approach handles both linear structures and regions with boundaries and how user-specified directions are gracefully preserved in the result. Source credits: cookie: Alessandro Paiva @ rgbstock; crochet: anneheathen @ flickr; denim: inxti @ shutterstock; bread: Giles Hodges @ DeviantArt; red wig: Lenor Ko @ shutterstock; braided wig: Karina Bakalyan @ shutterstock; ivy leaves: Michael & Christa Richert @ rgbstock; plank: My Life Graphic @ shutterstock; grass: varuna @ shutterstock
creation of digital art. The ability to synthesize complex natural textures with edge
effects make it useful for photo editing or matte painting applications.
7 Advanced Drawing Beautification with ShipShape


7.1 Introduction

![Examples of drawings created using ShipShape. The final drawings (black) were created from the imprecise user input (gray) by beautifying one stroke at a time, using geometric properties such as symmetry and path identity. See Figure 7.17 for more results.](image)

Sketching with a mouse, tablet, or touch screen is an easy and understandable way to create digital content, as it closely mimics its real-world counterpart, pen and paper. Its low demands make it widely accessible to novices and inexperienced users. However, its imprecision means that it is usually only used as a preliminary draft or a concept sketch. Making a more polished drawing requires significantly more time and experience with the drawing application being used. Furthermore, when working with drawing or
CHAPTER 7. SHIPSHAPE

Figure 7.2: Incremental beautification workflow. Every newly drawn stroke (blue) is beautified using previously created data (gray). The first stroke is left unchanged. As the drawing continues, more suitable geometric constraints emerge and are applied, such as path identity (2,6,7), reflection (2,6) or arc fitting (3,4). For comparison with the final beautified output (8), I shows the original input strokes.

Figure 7.3: Supported geometric rules and transformations in our framework. The blue paths represent the data being beautified, while gray paths are data already processed. For more detailed description of the criteria used to evaluate these constraints, see Section 7.3.1.

sketching software, users are often forced to switch between different drawing modes or tools or to memorize cumbersome shortcut combinations.

While we do not question the necessity or usefulness of complex tools to achieve non-trivial results, we argue that for certain scenarios, such as geometric diagram design or logo study creation, the interactive beautification [Igarashi et al., 1997] approach is more beneficial. Such workflows retain the intuitiveness of freehand input while benefiting from an underlying algorithm that automatically rectifies strokes based upon their geometric relations, giving them more formal appearance. With the quickly growing popularity of touch-enabled devices, the applicability of this approach expands greatly. However, whatever the potential of automatic beautification in a more general sketching context, most of the existing applications focus on highly structured drawings like technical sketches.

One of the biggest challenges in drawing beautification is resolving ambiguity of the user input, since the intention and its execution are often considerably dissimilar. Additionally, this issue becomes progressively more complex as the number of primitives present in the drawing increases.

In this paper, we present a system for beautifying freehand sketches that provides mul-
tiple suggestions in spirit of Igarashi et al. [1997]. Strokes are processed incrementally (see Figure 7.2) to prevent the combinatorial explosion of possible outputs. Unlike previous work, our approach supports polycurves composed of general cubic Bézier curves in addition to simple line segments and arcs. The system is scale-independent, and can easily be extended by new operations and inferred geometric constraints that are quickly evaluated and applied. The algorithm was integrated into Adobe Illustrator, including undo/redo capability. We present various examples to demonstrate its practical usability.

7.2 Related Work

The need to create diagrams and technical drawings that satisfy various geometric constraints led to the development of complex design tools such as CAD systems. However, these systems’ complexity often limits their intuitiveness. Pavlidis and Van Wyk [1985] were one of the first to try to alleviate this conflict by proposing a method for basic rectification of simple rectangular diagrams and flowcharts. However, their process became ambiguous and prone to errors when more complex drawings were considered, since the method needed to drop many constraints to keep the solution tractable.

To alleviate this limitation, Igarashi et al. [1997] proposed an interactive beautification system in which the user added strokes one by one and the system improved the solution incrementally while keeping the previously processed drawing unchanged. This solution kept the problem tractable even for very complex drawings. Moreover, the system also presented several beautified suggestions and let the user pick the final one. This brought more user control to the whole beautification process. Following a similar principle, other researchers developed systems for more specific scenarios such as the interactive creation of 3D drawings [Igarashi and Hughes, 2001], block diagrams [Plimmer and Grundy, 2005; Wang et al., 2005], forms [Zeleznik et al., 2008], and mathematical equations [LaViola and Zeleznik, 2004].

However, a common limitation of the approaches mentioned above is that they treat the image as a set of line segments. To alleviate this drawback Paulson and Hammond [2008] proposed a system called PaleoSketch that fit the user input to one of eight predefined geometric shapes, such as line, spiral or helix. In a similar vein, Murugappan et al. [2009] and Cheema et al. [2012] allowed line segments, circles and arcs.

Related to drawing beautification, there are also approaches to beautify curves independently, without considering more complex geometric relationships. Those approaches are orthogonal to our pipeline. They use either geometric curve fitting [Baran et al., 2010; Orbay and Kara, 2011] or some example-based strategy [Lee et al., 2011; Zitnick, 2013]. Additionally, advanced methods for vectorizing and refining raster inputs have been proposed [Noris et al., 2013; Su et al., 2014], which enable users to convert bitmap images into high quality vector output. However these do not exploit inter-stroke relationships. In our case we assume that the built-in curve beautification mechanism of Adobe Illustrator preprocesses the user’s rough input strokes into smooth, fair paths.

This paper extends our previous work [Fišer et al., 2015]. In Section 7.3.1 we discuss improvements to the arc and circle center rules, and introduce a generalized transformation adjustment framework. Section 7.3.4 describes a new method for curve alignment, and Section 7.3.5 describes the transformation adjustment mechanism in detail. Finally,
Section 7.4 describes a new framework for handling curves with corners.

7.3 Our Approach

A key motivation for our system is wanting to work with arbitrarily curved paths. This capability was not available in previous beautification systems. Although some can recognize a variety of curves including spirals and general 5th degree polynomials (Paleo-Sketch [Paulson and Hammond, 2008]), they recognize them only in isolation and do not allow to take other existing paths into consideration, which is important for interactive design.

Systems like that of Igarashi et al. [1997] generate a set of potential constraints and then produce suggestions by satisfying subsets of these. A key challenge that prohibits simply generalizing these systems to support general curved paths is the number of degrees of freedom, which boosts the number of potential constraints that need to be evaluated. Moreover, unlike line or arc segments, many of a general path’s properties, for example the exact coordinates of a point joining two smooth curves, do not have any meaning to the user. It would not be helpful to add constraints for this point. Finally, satisfying constraints on a subset of the defining properties might distort the path into something that barely resembles the original. Supporting generalized paths requires a different approach.

Our system is based on an extensible set of self-contained geometric rules, each built as a black box and independent of other rules. Every rule represents a single geometric property, such as having an endpoint snapped or being a reflected version of an existing path. The input to each rule is an input path consisting of an end-to-end connected series of Bézier curves, and the set of existing, resolved paths. The black box evaluates the likelihood that the path conforms to the geometric property, considering the resolved paths, and outputs zero or more modified versions of the path. Each modified version gets a score, representing the likelihood that the modification is correct.

For example, the same-line-length rule would, for input that is a line segment, create output versions that are the same lengths as existing line segments, along with scores that indicate how close the segment’s initial length was to the modified length. Each rule also has some threshold that determines that the score for a modification is too low, and in that case it does not output the path.

The rules also mark properties of the path that have become fixed and therefore can no longer be modified by future rules. For example, the endpoint-snapping rule marks one or both endpoint coordinates of a path as fixed. The same-line-length and parallel-line rules do not attempt to modify a segment with two fixed endpoints.

Since the rules do not depend on each other, it is easy to add new rules to support additional geometric traits. Figure 7.3 shows an illustrated list of rules supported in our system.

Chaining the rules can lead to complex modifications of the input stroke and is at the core of our framework. We treat the rule application as branching in a directed rooted tree of paths, where the root node corresponds to the unmodified input path. Each branch of the tree corresponds to a unique application of one rule and the branch is given a weight corresponding to the rule’s score.

To find suitable transformations for the user input, we traverse down to the leaf nodes
Formally, given a node $n^i$ with Bézier path $p^i$, the set of resolved paths $S$, and the set of all rules $r_j \in R$, we compute an output set $P^i = \{r_j(p^i,S)\}$. We then create a child node $n^i_j$ for each $p^i_j \in P^i$. If $P^i$ is empty, $n^i$ is a leaf node.

Since we need to compare scores among different rules, likelihoods are always normalized into the interval $[0, 1]$. If a rule generates any modified paths, it also generates a copy of the unmodified path, indicating the suggestion that the rule did not apply. The likelihood for the unmodified path is 1 minus the maximum likelihood of any modified path.

We can then use all scores from the nodes we visited while descending into a particular

![Diagram](image_url)

**Figure 7.4:** Successive rule evaluation and application. In this example, the evaluation engine consists of three geometric rules—endpoint snapping, perpendicularity, and length equality. The old data (gray path) is fixed in the canvas. When a new path (blue) is added, it becomes the root node of the evaluation graph and the expansion begins by testing all rules on it. A likelihood score is calculated for each rule application and the tree is expanded using a best-first search scheme, until leaf nodes are reached. Due to the significant redundancy in the search space, many leaf nodes will contain duplicate suggestions. Therefore, we prune the graph during the expansion step using the information from already reached leaf nodes (see Section 7.3 and Figure 7.5 for more information).
leaf node \( n \) to calculate the overall likelihood score for the chained transformation as

\[
\bar{L}_i = 1 - \frac{d-1}{\prod_{k=1}^{d-1} (1 - L(r_j(a^k, S)))}
\]  

(7.1)

where \( d \) is the depth of \( n \) in the tree, \( a^k \) is the \( k \)th ancestor of \( n \), and \( L(r_j(a^k, S)) \) denotes the likelihood score from applying rule \( r_j \) to node \( a^k \).

We expand the search tree in a best-first search manner, where the order of visiting the child nodes is determined by the overall score \( \bar{L} \) of the node’s path. While traversing the tree, we construct a suggestion set \( Q \) of leaf nodes, which is initially empty and gets filled as the leaf nodes are encountered in the traversal. Once not empty, \( Q \) helps prune the search. Before we expand a particular subtree, we compare the geometric properties of its root with properties of each path \( q \in Q \). If all tested properties are found in some path \( q \), the whole subtree can be omitted from further processing (see Figure 7.5).

**Figure 7.5:** Search graph pruning. The rules are represented by colored boxes with hue being distinct rules and lightness their unique applications (e.g., if red color represents endpoint snapping, then different shades of red correspond to snapping to different positions). An inner node \( n \) has been expanded into three branches (a,b,c). Before further traversal, all subtrees stemming from the child nodes of \( n \) are tested against suggestions \( q \in Q \). Here, branches (a) and (c) are fully contained in \( q_0 \) and \( q_2 \) respectively and thus only branch (b) is evaluated further.

Furthermore, to keep the user from having to go through too many suggestions, we limit the size of \( Q \). Since we traverse the graph in a best-first manner, we stop the search after finding some number of unique leaf nodes (10 in our implementation).

### 7.3.1 Supported Rules and Operations

Geometric transformations in our framework are evaluated by testing various properties of the new path and the set of previously drawn and processed paths. While tests of some properties are simple, others, such as path matching, require more complex processing. We first summarize rules supported by our system (illustrated in Figure 7.3), and then we present some additional implementation issues including a more detailed description for non-trivial rules.
**Line Detection** We estimate a path’s deviation from straightness by measuring the ratio between its length and the distance between its endpoints, as in QuickDraw [Cheema et al., 2012].

**Arc Detection** We sample the input path and perform a least-squares circle fit on the samples to obtain center and radius parameter values. To determine the angular span value, we project the samples onto the circle fit. The arc is then sampled again and we evaluate the discrete Fréchet distance [Eiter and Mannila, 1994] between the arc samples and the samples of the input path. When the span is close to $2\pi$ or the path is closed, we replace it with a full circle.

**Endpoint Snapping** We look at the distance between each of the path endpoints and resolved endpoints. Additionally, we also try snapping to inner parts of the resolved paths. Specialized tests based on the properties of line segments and circular arcs lower the computational complexity of this operation. Note that we do not join the two end-to-end-snapped paths. This can cause unpleasant artifacts where they meet, but the effect of a join can be mimicked by using round end caps on the strokes.

**End Tangent Alignment** If the path endpoint is snapped, we measure the angle between its tangent and the tangent of the point it is attached to.

**Line Parallelism and Perpendicularity** We compare the angle between two line segment paths with the angle needed to satisfy the parallelism or perpendicularity constraint. Additionally, we also take the distance between the line segments into account to slightly increase the priority of nearby paths. To evaluate these properties on the input non-rectified paths, we use their line segments approximations, i.e., line segments connecting their two endpoints.

**Line Length Equality** We evaluate the ratio of length of both tested line segments. As in previous case, we incorporate their mutual distance in the final likelihood computation.

**Arc and Circle Center Snapping** Similar to endpoint snapping, we evaluate the distance between the current arc center and potential ones, in this case endpoints of other paths, other centers, centers of rotations, and centers of regular polygons composed from series of line segments. However, as arcs with small angular span are noticeably harder to draw without a guide (see Figure 7.6a), the center of the initial arc fit might be located too far apart from the desired center point (Figure 7.6b) and therefore using fixed distance, when looking for potential center-snapping points, might not be sufficient. To address this issue, we adaptively change this distance to $\max(D, 2r(1 - \theta/2\pi))$, where $\theta$ is the span of the tested arc, $r$ is its radius and $D$ is the standard search distance radius ($D = 30$ view-space pixels in our implementation).

**Path Identity** To detect that two paths have similar shapes, we align them and compute their discrete Fréchet distance. More details are given in Section 7.3.4.

**Transformation Adjustment** For a tested path $x$ and resolved reference path $y$ of the “same shape” (determined by successful application of the path-identity rule) we perform a variety of modifications to the transformation to create symmetries, align paths, and equalize spacing. More details are given in Section 7.3.5.
Figure 7.6: Adaptive arc/circle center-point-snap search distance refinement. Arc segments with small angular span are often drawn very imprecisely (a). When the engine fits an exact arc into such data, its center is often too far from the desired center point, as the distance $d$ between them is bigger than the limit $D$ under which the prospective center point positions are looked for (b). Adaptive expansion of the search radius $D'$ increases the likelihood that even the imprecise input will give the user the expected (precise) output.

Path OffsetOffset paths generalize line parallelism. To detect them, we go along the tested path and measure its distance to the reference path. More details are given in Section 7.3.6.

7.3.2 View-Space Distances

Testing paths for different geometric properties ultimately requires measuring lengths and distances. While many path attributes can be compared using relative values, absolute values are still necessary, e.g., for snapping endpoints. Using absolute values, however, leads to unexpected behavior when the canvas is zoomed in and out. To eliminate this problem, we compute all distances in view-space pixels, making all distance tests magnification-independent.

7.3.3 Path Sampling

Working with cubic Bézier curves analytically is inconvenient and difficult. Many practical tasks, such as finding a path’s length or the minimal distance between two paths, can only be solved using numerical approaches. Therefore, we perform all operations on sampled paths. Since the resolved paths do not change, we can precompute and store the samples for resolved paths, and sample only new paths. Furthermore, to reduce the memory requirement and computational complexity of different path comparisons, we simplify the sampling using the Ramer–Douglas–Peucker algorithm [1972; 1973]. For a polyline $p$, this finds a reduced version $pt$ with fewer points within given tolerance $\epsilon$, i.e., all points of $pt$ lie within the distance $\epsilon$ of the original path (see Figure 7.7). Our implementation uses $\epsilon = 4$ view-space pixels at the time the path was drawn.
Figure 7.7: Path sample simplification. The original Bézier path (a) is equidistantly sampled, giving a polyline (b). The Ramer-Douglas-Peucker algorithm then recursively simplifies the polyline by omitting points closer than $\epsilon$ (c) to the current approximation, finally constructing simplified polyline (d).

### 7.3.4 Path Matching

A key part of our contribution involves resolving higher-level geometric relations like path rotational and reflection symmetry. To identify these relations, we must first classify paths that are the “same shape”—paths that are different instances of the same “template”.

To evaluate the similarity between two sampled paths $p_a$ and $p_b$, we employ a discrete variant of Fréchet distance [Eiter and Mannila, 1994], a well-established similarity measure. Formally, it is defined as follows: Let $(M,d)$ be a metric space and let the path be defined as a continuous mapping $f : [a,b] \rightarrow M$, where $a, b \in \mathbb{R}$, $a \leq b$. Given two paths $f : [a,b] \rightarrow M$ and $g : [a',b'] \rightarrow M$, their Fréchet distance $\delta_F$ is defined as

$$\delta_F(f,g) = \inf_{\alpha,\beta} \max_{t \in [0,1]} d(f(\alpha(t)), g(\beta(t))),$$

where $\alpha$ (resp. $\beta$) is an arbitrary continuous non-decreasing function from $[0,1]$ onto $[a,b]$ (resp. $[a',b']$). Intuitively, it is usually described using a leash metaphor: a man walks from the beginning to the end of one path while his dog on a leash walks from the beginning to the end of the other. They can vary their speeds but they cannot walk backwards. The Fréchet distance is the length of the shortest leash that can allow them to successfully traverse the paths.

As outlined by Eiter and Mannila, this can be computed for two point sets using a dynamic programming approach. The extension to point and line-segment sets (Figure 7.8b) is then straightforward. However, the measure takes into account the absolute positions of the sample points, while we are interested in relative difference. Therefore, we have to adjust the alignment of the two tested paths. We then compute the discrete Fréchet distance between the aligned paths, divided by the length of the new path to obtain the relative similarity measure.
Figure 7.8: Discrete Fréchet distance. The minimum length of the line connecting ordered sets of point samples (a). Since we store the resolved paths in the simplified form, we compute the Fréchet distance between an ordered set of points and an ordered set of line segments (b) rather than between two point sets.

An affine similarity transform is a composition of a rotation, a uniform scale, and a translation. To align the paths, we find the affine similarity matrix that transforms the reference path to match the new path as closely as possible.

Assume the rotation angle is $\theta$, the scale is $s$, and the translation is $(tx, ty)$. Define $scos = s \cdot \cos \theta$ and $ssin = s \cdot \sin \theta$. The matrix is then

$$
\begin{bmatrix}
s cos & -ssin & 0 \\
ssin & scos & 0 \\
tx & ty & 1
\end{bmatrix}
$$

We compute the affine similarity transformation matrix $M$ as follows. We first create two equal-length lists of points, each consisting of $N$ equally-spaced samples from the reference and new paths. If $\{P_i\}$ are the points from the reference path and $\{Q_i\}$ the points from the new path, we find the $M$ that minimizes the sum of the squared distances

$$
E = \sum_{i=1}^{N} ||P_i \cdot M - Q_i||^2
$$

This is a quadratic function of $scos$, $ssin$, $tx$, and $ty$ and can be solved as a least-squares problem over these four variables.

Before computing the Fréchet distance, we multiply the reference path samples by $M$. If the Fréchet distance indicates that the paths are sufficiently similar, we create a suggestion consisting of the reference path transformed by this same $M$.

A path that is a transformed copy of another path is permanently annotated as such, thereby allowing us to optimize path matching by only testing against a single instance of the path. For later processing, we also annotate the path with the transformation matrix.

If the drawing already contains multiple instances of a path, we consider it more likely that the user intended a new path to match. We therefore boost its score $s$ by replacing it with $1 - (1 - s)^{ln i}$ where $i$ is the number of existing instances.

Because the new path might be a reflected and/or reversed version of the reference path, we perform four tests between them to determine the correct match.
Figure 7.9: Transformation adjustment and transformation step snapping. The reference path \( R \) already has a copy \( C \) in the drawing, with \( M_{RC} \) being the transformation from \( R \) to \( C \). \( D \) is the test path with \( M_{RD} \) being the transformation from \( R \) to \( D \). Transformation adjustment considers both \( M_{RD} \) and the derived relative matrix \( M_{CD} \) that transforms \( C \) to \( D \) (a). The step transform for \( D \) is then \( M_{CD} \), the relative transform from \( C \) (b). The relative transform for \( T \) relative to \( D \) is similar to the step transform for \( D \) (c). Applying \( M_{CD} \) to \( D \) generates a well-spaced suggestion (d).

7.3.5 Transformation Adjustment

If the test path is a transformed version of a reference path, there are various tests we perform to adjust the transformation matrix to make the result more pleasing. We first begin by separating the matrix in Equation 7.3 into separate rotation, scale, and translation components as follows:

\[
\begin{align*}
\text{rotation} &= \arctan2(ssin,кос) \\
\text{scale} &= \sqrt{кос^2 + ssin^2} \\
\text{translation} &= (tx,tx)
\end{align*}
\] (7.5)

The transformation can be adjusted in various ways, often generating multiple suggestions. Although we optimized path matching to only compare against one instance of a path that has multiple copies in the drawing, we test the transformation relative to each copy; see Figure 7.9a.

Rotation Snapping If the rotation component is close to an angle that is an integral divisor of \( 2\pi \), it is snapped to being that angle (e.g., to 45 degrees; see Figure 7.10b4).

Scale Snapping If the scale component is close to an integer or to 0.5, it is snapped to being that exact scale.

Translation Snapping Translation snapping takes several forms:

- If the transformation contains a rotation component, we find the rotation center and compare it to existing points in the drawing. If it is sufficiently close we adjust the translation to place the center of rotation at that point.
• If the test path is a reflected version of the resolved path, we first compute the axis of reflection and reflect the resolved path across this axis. If the test path is sufficiently close to this reflected path, we adjust the translation to move it to that position.

• In other cases, we snap the $x$ and $y$ components of the translation to zero.

**Step Transform Snapping** Step transform snapping allows the user to create multiple, equally transformed copies of a path (see Figure 7.10b3). When we snap a path to an instance of a path, we store the relative transformation to that instance as the **step transform**. The step transform is the relative transform of the most highly-scoring suggestion. In Figure 7.9b, the existing drawing contains three resolved paths that are all the same shape. $R$ was drawn first, and is the reference path. $C$ is the first copy, and its step transform is the transformation from $R$ to $C$. $D$ is the second copy, and it was horizontally snapped to $C$. Because the transformation from $C$ scored more highly (containing a snap) than the transformation from $R$, the step transform for $D$ is the relative transform from $C$ to $D$.

Step transform snapping compares the transformation from a path instance to the step transform for that instance. If the two transformations are similar, then a step-snapping suggestion is generated. In Figure 7.9c, the newly drawn path $T$ is compared to all three existing instances $R$, $C$, and $D$. The transformation $M_{DT}$ from $D$ to $T$ is similar to the step transform of $D$. This generates a step-snapping suggestion to place $T$ in the position that exactly matches the step transform; see Figure 7.9d.

Although this example only includes translation in the step transform, they are fully general, and can include rotation, scale, and reflection (see Figure 7.10b2).

**Reflection Axis Snapping** Users often want to reflect multiple paths against the same axis of reflection (for example, see the bear in Figure 7.1), or want to reflect a path across an existing line segment. To accommodate this, we collect all existing axes of reflection and line segments. If the new path is reflected, we compare its axis of reflection to these potential axes, and if it is close, we generate a suggestion to reflect across this axis (see Figure 7.10b1). Further, we strengthen the likelihood for an axis that has already been used multiple times by replacing the score $s$ with $1 - (1 - s)^{\ln i}$ where $i$ is the number of times that axis has been used.

### 7.3.6 Offset Path Detection

Offset paths extend the concept of parallelism from line segments to paths. To detect them, we construct a normal line from each sample of the new path. If the line hits an existing reference path, we measure the distance between the sample point and the closest point on the reference. Note that we do not use the distance between the sample point and the line-path intersection, since this would require the user to draw the approximate offset path very precisely. We store the measured distance along with its sign, i.e., on which side of the new path the hit occurred. We then sort all the hit information according to the distance, creating a cumulative distribution function, and pick two values corresponding to $(50 \pm n)$-th percentiles ($n$ being 25 in our implementation). By
Figure 7.10: Practical application of transformation adjustment of the imprecise input (b) to obtain highly symmetrical output (a). We apply reflection axis (1), step transform (2,3) and rotation (2,4) snapping. Also note that the whole drawing is composed of strokes of the same shape.

Comparing the sign and distance values of these samples, we calculate the likelihood of the new path being an offset path of the reference path (see Figure 7.11). If the likelihood is high, we replace the new path with an offset version of the reference.

7.4 Multi-Segment Stroke Processing

The single stroke processing approach gives the user the opportunity to immediately see the results of the input being beautified. However, in certain cases, like drawing simple triangles or squares, this workflow can be tedious and decrease the overall fluency of the beautification pipeline. To this end, we introduce an additional step into our scheme that lets the evaluation engine process strokes with multiple segments. These segments are defined as parts of the unprocessed user input, split by corner features. Once divided, the evaluation engine can process the simple segments using the geometric rules introduced in Section 7.3.1.
CHAPTER 7. SHIPSHAPE

Figure 7.11: Offset path detection. A line is constructed from each point on the sampled path (blue circles) in the normal direction. If an existing reference path is hit (red rays), the minimal distance from the sample to the reference path is calculated (dashed lines) and used in offset-path-likelihood computation (see 7.3.6).

Figure 7.12: Multi-segment stroke processing pipeline. When a complex stroke is drawn (a), it is tested for the presence of corner points. If no corner points are found, the processing continues as described in Section 7.3. If one or more corner points are detected (see Section 7.4.1 for more details), the original stroke is split and broken into segments (b). The segments are then processed sequentially. After each individual segment is added (c, from top to bottom), suggestions are generated (d) using previous segments as well as old strokes. In particular, beginning with the second segment, the beginning endpoint is constrained to match the final endpoint of the previous segment (c, red circles, see Section 7.4.3). After generating suggestions for a segment (d, from top to bottom), an optional set reduction can be done (e) to keep the evaluation sufficiently fast (see Section 7.4.2).

7.4.1 Corner Detection

When the raw freehand input stroke is drawn by the user, it is converted to a sequence of cubic Bézier curves and passed to the beautification pipeline. The first step is to test it for the presence of corner points. Because the initial curve fitting is done by the host application (e.g., Adobe Illustrator), we cannot simply rely on the assumption that corners can only occur at the junction of two Bézier curves. For example, in Figure 7.12a,
the apparent corner in the lower right is actually a small-radius curve. We initially sample
the curves with a small step size (2 view-space pixels) and calculate the tangent vector at
each sample point. Using a sliding window of three successive samples, we calculate the
angular turn value at every sample position except the first and last. Local maxima in
this turn sequence provide the places to break the original input sequence into segments.
To handle outliers like the unwanted “hooks” at the ends, we discard segments whose
length is small compared to the rest of the segments (less than 15% of the length-wise
closest other segment).

7.4.2 Segment Processing

The segments of the complex user input can then be processed one at the time using the
same approach used for the simple input described in Section 7.3. There are, however,
important issues to address. Most notably, processing multi-segment input involves auto-
matic selection of intermediate outputs, which would otherwise be done by the user. As
the number of potential outputs rises exponentially, we cannot explore the whole search
space. Therefore, we perform two reduction steps to make the evaluation of complex
inputs computationally feasible within real-time-to-interactive response time. First, we
limit the number of unique suggestions for each segment to 3 (whereas the single-segment
input can produce up to 10 suggestions). This might seem to be a very severe restriction,
but the split segments are typically simple paths with very little ambiguity. Second, we
process the individual segments in a breadth-first manner that lets us execute another
reduction once all the parallel states reach the same depth (i.e., they all have the same
number of processed segments; see individual rows in Figure 7.12d). For this step, we
assign each intermediate state a value calculated as the arithmetic mean of the scores
of the processed segments. Then, only $N_{IS}$ intermediate states are kept and evaluated
further while the rest are discarded (Figure 7.12e). The performance of multi-segment
input processing is determined by the number of segments $K$ and the intermediate stack
size $N_{IS}$, with $N_{IS} = 1$ being performance-wise equal to sequential processing of indi-
vidual segments. In our implementation, $N_{IS} = 10$ and strokes constituted of up to 10
segments can be processed without noticeable lagging.

7.4.3 Internal Segment Restrictions

As the individual segments are pieces of one original input curve, we must ensure that
the beautified segments are consecutively joined. Thus, we constrain the position of the
first endpoint of each segment after the first (rows 2,3 in Figure 7.12c). Additionally,
if the input stroke is closed, we also constrain the last segment’s final endpoint (row 3
in Figure 7.12c). As a side effect, this also helps to decrease the ambiguity.

7.4.4 Segment Joining And Further Behavior

Once all the segments have been processed, we create the final output stroke by joining
them together. This way, the combined beautified input stroke can be used by rules such
as curve identity. Internally, the beautification engine keeps also tracks the individual
segments so that they behave as if they were drawn one after each other. This lets the
geometric rules show the expected behavior, e.g., the corners of a complex stroke can be
used as snapping points.
CHAPTER 7. SHIPSHAPE

Figure 7.13: Examples of multi-segment stroke processing. The input strokes (blue) are broken into individual segments that are sequentially processed using the single-segment evaluation engine (Section 7.3) and merged after the processing is finished (see Section 7.4 for details).

Figure 7.14: Visual annotation hints. Overlaid visual annotations show which rules have been applied, e.g., line perpendicularity and endpoint snapping (a), line parallelism and single coordinate snapping (b) or path identity (c).

7.5 Implementation Details

While using an existing API requires us to conform to its design rules, it also eliminates the need to handle many tasks unrelated to the research project, such as tracking the input device, fitting paths to the samples, and managing the undo/redo stack. It also benefits the users, as they are not forced to learn yet another user interface, and can instead take advantage of built-in tools of the existing program. Therefore, we decided to integrate our system into Adobe Illustrator as a plugin using its C++ SDK.

As described previously, our method is based on evaluating different geometric rules on a new path using the previously drawn and resolved paths. Thus, we need to be able to detect when a new path is created or an old one is modified or deleted. To this end, we serialize all the path data and store a copy in the document. Illustrator activates our system whenever the user modifies the document. We deserialize the data and compare the paths to the actual paths in the document to detect changes. If we find a new path, we process the new path and update the serialized data. Similarly, when a path is modified, it is treated as new one and reprocessed. Deleting paths does not affect the remaining ones. To support undo and redo, we store the serialized data into a part of the document that is managed by the undo/redo system.

The presentation of the suggestions is deliberately kept as simple as possible and only one suggestion is shown at the time. The user switches among the suggestions using an additional Illustrator tool panel. The last suggestion in the list is always the original
input path and is thus easily accessible. Currently, the list of inferred constraints is shown in textual form in the order in which they were traversed in the search space tree (see Figure 7.18c). The user selects the current suggestion by drawing a new path or changing the selection. To provide additional assistance for the user, we also present a simple visualization of the applied rules together with rectified path. This visual annotation provides immediate feedback about the imposed constraints and relations of the user input (see Figure 7.14).

To further exploit the built-in tools, we support the “Transform Again” feature for rotational symmetry. If the resolved path is a rotated copy of an existing path, it is noted as such so that a new, properly-rotated copy will be created if the user invokes the “Transform Again” command. The user only needs to draw two rotated instances of a path and then can create additional properly-rotated paths without drawing them (see Figure 7.18d). Recall that the rotation angle is adjusted to the nearest integer quotient of $2\pi$, so additional paths can form full $n$-fold rotational symmetry.

The constraints imposed by ShipShape can easily be avoided for certain paths by placing them in layers that are not being rectified. In our implementation, ShipShape runs only on the default layer.

### 7.6 Results

To evaluate the effectiveness of our method, we conducted a preliminary study. We created a plugin for Adobe Illustrator that was installed on a PC with a 23in LCD monitor and a consumer computer mouse as the input device. Six people participated in this study. All of them worked with Illustrator on a daily to weekly basis, but in all cases, their primary work-related tool was a CAD program. First, the users were given a brief introduction and demonstration of our system’s concept, capabilities and limitations, with a few practical examples. The participants could adjust Illustrator settings and the mouse sensitivity according to their needs, and then spent 1 to 3 minutes in free drawing, to get briefly accustomed to the system and the workflow.

The users were then shown three simple illustrations (see Figure 7.15) and presented with the task of drawing each of them anew, using both native Illustrator tools and our prototype, while we measured their drawing times. First, the participants were asked to recreate the figures using any suitable tools and approaches, i.e., they could use all the available tools and modes, such as copying or reflecting. Rather than creating the exact copies of the reference drawings, we directed them to focus on preserving the geometric relations. Interestingly, despite the users’ relatively equal level of experience, they often took very dissimilar ways to recreate the task’s drawing.

In the second part of the test, the participants were only allowed to use the pencil tool with the ShipShape prototype turned on. The only additional allowed operation was undo. Similarly to the first part of each drawing, the users took a different approaches to complete the goal, however, with a single exception, they were all able to finish all three designated drawings from Figure 7.15. The initial measurements (Figure 7.16) suggest that drawing beautification is more suited for simpler drawings and tasks. For example, copying a large part of the bottom-left drawing in Figure 7.15 was always faster than redrawing it.

The main interest of this study was to identify the weak points and bottlenecks of our
Figure 7.15: Evaluation study drawings. The users were asked to recreate these drawings using our ShipShape prototype: Task drawing (left, black), representative raw input (right, gray).

approach and to test how successful our prototype was in generating correct suggestions. The overall feedback from the participants was positive. They found the tool useful and easy to use. Most of the participants, however, considered the limitation of using a single tool only too restrictive, and suggested incorporating parts of our approach (smart snapping, automatic tangent adjustments, etc.) into the relevant built-in tools. All the participants liked the idea of visual annotations (Figure 7.14) and found it helpful. Several users did not like the way the alternative suggestions were presented and explored (see the small gray box in canvas in Figure 7.18) and preferred to undo and redraw the particular strokes.

Additional results are shown in Figure 7.17. Note that an important part of the
Figure 7.16: Comparison of drawing performance. The participants were asked to recreate the drawings from Figure 7.15 using either native tools of Adobe Illustrator (red) or ShipShape prototype (blue line). For simpler drawing, such as the spiral or the clock, ShipShape typically outperformed Illustrator. However, with more complex drawings (table), the utilization of different tools is faster.

drawing workflow was relying on Illustrator’s built-in support for curve smoothing when creating original paths—those that are not copies of other paths. These are shown in blue in Figure 7.17, and they function as “template” paths for the beautification. Other strokes drawn afterwards can be much more imprecise (see Figure 7.1 and Figure 7.17c–g).

7.7 Limitations and Future Work

A common problem of drawing beautification techniques is the quick growth of the number of possible suggestions as the drawing becomes more complex and many satisfiable geometric constraints emerge. Our approach addresses this by combining best-first search with a limited suggestion set size, but additional heuristic-based pruning of the search space, possibly based on empirical measurements, could improve the suggestion set.

Currently, when the user changes an already-resolved path, it is treated as a new one. In some cases, however, it would be beneficial to not only reprocess the modified path but also all other paths being in relationship with it, for example changing any reflected or rotated versions of the path.

7.8 Conclusion

In this paper, we presented an efficient method for beautification of freehand sketches. Since the user input is often imprecise and thus ambiguous, multiple output suggestions must be generated. To this end, we formulated this problem as search in a rooted tree graph where nodes contain transformed input stroke, edges represent applications of geometric rules and suitable suggestions correspond to different paths from root node to some leaf nodes. To avoid the computational complexity of traversal through the whole graph, we utilized a best-first search approach where the order of visiting tree nodes is directed by the likelihood of application of the particular geometric rules.

On top of this framework, we developed a system of self-contained rules representing different geometric transformations, which can be easily extended. We implemented various rules that can work not only with simple primitives like line segments and circular
Figure 7.17: Various results obtained using our method. The side-by-side pairs show the beautified output (black) and the original input strokes (gray). Note that we do not perform any curve smoothing, beyond what is provided by Illustrator. Therefore, when dealing with general curves, the first “template” strokes (blue) have to be drawn more precisely or be smoothed using built-in Illustrator capabilities.

arcs, but also with general Bézier curves, for which we showed how to detect previously unsupported relations such as curve identity or rotational and reflection symmetry.
We demonstrated the usability and potential of our method on various complex drawings. Thanks to the ability to process general curves, our system extends the range of applicability of freehand sketching, which was limited previously to drawings in specialized, highly-structured applications like forms or technical diagrams. We believe that this advantage will become even more apparent with the widespread adoption of touch-centric devices, which rely much less on classical beautification techniques that are based upon menu commands and multiple tools.

![Illustrator interface](image)

**Figure 7.18:** Exploiting the “Transform Again” feature. Illustrator allows the user to repeat the last transformation. When a new path is added (b) to the canvas (a), it is processed and output suggestions are generated. If the user chooses a suggestion that is a rotation (c) we enable the “Transform Again” feature. The user can then easily complete the 8-fold rotational symmetry drawing (d). See Section 7.5
8 StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings


8.1 Introduction

Figure 8.1: Stylization of a 3D rendering (a) produced by our method using various style exemplars provided by the artist (top inset): (b) tonal drawing, (c) colored pencils, (d) oil pastel, and (e) comic drawing. Note how the specific stylization of individual lighting effects on the exemplar sphere is transferred to a similarly illuminated location in the target 3D rendering. Exemplar images: © Daichi Ito (b), Pavla Sýkorová (c, d), and Lukáš Vlček (e).

Stylizing synthetic renderings of 3D models to give them a hand-crafted artistic appearance has wide applications in design, advertising, games, and movies. Previous example-based approaches [Sloan et al., 2001; Hertzmann et al., 2001; Bénard et al., 2013; Fišer et al., 2014; Barnes et al., 2015] have made significant progress, but the synthesized results still have a distinctively synthetic look when compared to real artwork. In this paper we identify two main factors that cause this problem and propose a solution to alleviate them.

The first limiting factor is that the state-of-the-art techniques rely mainly on color information to determine the stylized appearance. This leaves them unable to distinguish among different regions that have similar colors (see Figure 8.2, top). Actual artists pay as much attention to lighting effects as they do to color when painting a scene. They often use different types of brush strokes and even different colors to depict differently lighted regions, even if they have the same color—for example, a gray diffuse region and a similarly-colored shadow might be painted differently (see Figure 8.3). Artists purposefully deviate from the true colors in the scene to emphasize the lighting effects. To better emulate the look of real paintings, synthesis must take the illumination conditions of the synthetic scene into account. Although normals can partially alleviate this limitation [Sloan et al., 2001], they are useful only for a simple shading scenario where the light source is sufficiently far away that the normals correctly determine the locations of the lighting effects. When this assumption is not satisfied, or when there are other
more advanced illumination effects like shadows or glossy reflections, normals become insufficient (see Figure 8.2, bottom).

To allow illumination-dependent stylization, we propose a novel approach in which we compute light propagation in a simple scene (inset in Figure 8.1a), and let the artist provide a corresponding painting (insets in Figure 8.1b–e) that depicts various global illumination effects in arbitrary styles. (For brevity, we refer to the artist’s creation as a painting, but it can use any technique like pen-and-ink, pastel, or colored pencil.) Then, for a more complex target scene (Figure 8.1a) with a similar lighting environment, we compute the light propagation and use it to guide the synthesis—to find appropriate regions in the exemplar painting to transfer appearance from. The shadow, highlight and diffuse regions in the synthesis result exhibit similar visual style to the corresponding regions in the exemplar.

The second limiting factor of previous example-based stylization techniques is that they tend to distort high-level textural features like individual brush strokes [Sloan et al., 2001; Hertzmann et al., 2001; Bénard et al., 2013] (see Figure 8.11d) or excessively reuse a small subset of source patches [Fiser et al., 2014; Barnes et al., 2015], producing a distinct wash-out effect manifesting as artificial repetitions or homogeneous areas that do not exist in the original style exemplar [Kaspar et al., 2015; Jamriška et al., 2015] (see Figure 8.11i). Both types of artifacts give a distinctively synthetic look that decreases the fidelity of the synthesized image. Our solution encourages uniform usage of source patches while controlling the overall error budget to avoid enforcing the use of patches that can cause disturbing visual artifacts. This improvement allows us to generate compelling results in cases where previous approaches fail.

![Figure 8.2: Colors and normals are insufficient to guide the synthesis of stylized artwork.](image)

(Top) Color-based stylization fails when an artist wishes to apply different styles to parts of the source that have similar colors. (Bottom) Normal-based stylization cannot capture illumination changes that result when the light source is relatively close to the object. It also fails to transfer advanced illumination effects such as shadows.
8.2 Related Work

The goal of computer-assisted stylization [Kyprianidis et al., 2013] as pioneered by Paul Haeberli [1990] is to convert a photo or computer generated image into a digital painting. Numerous techniques were developed to achieve this goal, using physical simulation [Curtis et al., 1997; Haevre et al., 2007], procedural techniques [Bousseau et al., 2006; Bénard et al., 2010], advanced image filtering [Winnemöller et al., 2012; Lu et al., 2012], or an algorithmic composition of exemplar strokes [Salisbury et al., 1997; Zhao and Zhu, 2011]. Zeng et al. [2009] decomposed the stylized image into a set of meaningful parts for which semantic interpretation is available [Tu et al., 2005], and then modified the stroke placement process to better convey the semantics of individual regions. All those techniques can produce impressive stylization results in certain cases, but they are limited to a specific appearance determined by the algorithm or by the library of used strokes.

Sloan et al. [2001] introduced The Lit Sphere—a generic example-based technique that uses a shaded sphere painted by an artist as the style exemplar. Pixels from this spherical exemplar are then transferred to the target 3D model using environment mapping [Blinn and Newell, 1976], i.e., the color for a target pixel is transferred from the location in the source with the same normal. This leads to disturbing stretched-texture artifacts (see Figure 8.11c, in which the orange shading lines are very stretched on the biggest torus). Moreover, since the technique requires one-to-one mapping between normals and lighting effects it can be used only for a simple shading scenario where the target has the same lighting environment as the source, the light is very far away, and there are no advanced illumination effects like shadows or glossy reflections.

Hertzmann et al. [2001] proposed a concept of image analogies where a pair of images (unfiltered and filtered) serves as an exemplar. For each pixel in the target, the algorithm finds the best corresponding location in the unfiltered source and transfers the look from
the filtered counterpart. Due to its greedy nature it tends to resolve the balance between maintaining the texture coherency and following the guidance by introducing visible seams, which degrades the overall fidelity of the synthesized image (see Figure 8.11d). This approach was later extended to handle animations [Hashimoto et al., 2003; Bénard et al., 2013] and to control the spatial location as well as local orientation of the source textural features in the synthesized image [Wang et al., 2004; Lee et al., 2010]. However, the guidance is still mainly based on color, making these approaches unable to handle illumination-dependent stylization.

In recent work [Fišer et al., 2014; Barnes et al., 2015] the original Hertzmann et al. synthesis algorithm has been replaced by a texture optimization technique [Kwatra et al., 2005; Wexler et al., 2007]. However, this approach suffers from a wash-out effect (see Figure 8.11i) caused by excessively reusing patches with low-frequency content [Newson et al., 2014]. Numerous strategies have been developed to mitigate this phenomena. Those include a discrete solver [Han et al., 2006], feature masks [Lefebvre and Hoppe, 2006], color histogram matching [Kopf et al., 2007], and bidirectional similarity [Simakov et al., 2008; Wei et al., 2008]. However, as recently demonstrated by Kaspar at al. [2015] and Jamriška et al. [2015], those techniques only work in some particular cases, for example when the source is mostly stationary and does not contain many nearly-homogeneous patches. Those conditions are usually violated for realistic style exemplars. Instead, Kaspar at al. and Jamriška et al. proposed more viable content-independent solutions that encourage uniform patch usage. A similar technique was previously used in [Chen and Wang, 2010] and [Bénard et al., 2013]. Unfortunately, this uniform usage constraint does not apply in our scenario since some patches need to be used more often than others—for example when patches from one source highlight need to be reused for multiple highlights in the target.

Recently, an alternative approach to computer assisted stylization was proposed by Gatys et al. [2015]. It uses a deep neural network trained on object recognition [Simonyan and Zisserman, 2014] to establish a mapping between features in the style and target image. Although this technique produces impressive results, it learns common visual features from a generic set of natural images and thus does not fit the task of style transfer for rendered scenes. More importantly, the transfer is based purely on statistics of color patterns and provides no intuitive way to control the transfer process, so the result of the style transfer is mostly unpredictable.

In a related domain, Diamanti et al. [2015] showed how complex material appearance can be synthesized from limited examples using additional annotations, including normals and a simple shading descriptor. Similar to our method, they use rendered 3D models as targets. However, they focus on realistic material synthesis, and their annotations cannot capture the complex lighting phenomena expressed in our exemplars. In addition, for the synthesis, they use image melding [Darabi et al., 2012], which is unfortunately also prone to the wash-out problem described above.

8.3 Our Approach

We created a generic example-based stylization algorithm that supports arbitrary natural media and styles. The input is an example painting that is coarsely aligned to a simple 3D scene (Section 8.3.1). Our framework can then synthesize renderings of complex new
scenes that imitate the visual style of the exemplar painting.

Previous stylization approaches based solely on color or normals cannot reproduce the richness of hand-painted images—higher level information is needed. One source of information can come from semantically parsing the objects [Zeng et al., 2009], but we believe that it is more important to analyze the light propagation in the scene. Because artists typically paint according to illumination effects, style-specific variations are driven more by illumination changes rather than by object identities (see Figure 8.3). Since the 3D geometry is known in a computer-generated scene, the light propagation can be computed using established rendering algorithms [Kajiya, 1986]. This allows us to distinguish differently illuminated regions (Section 8.3.2). We use the illumination information to guide the synthesis algorithm to achieve context-dependent stylization (Section 8.3.3).

8.3.1 Workflow Overview

In our workflow, an artist first prepares a stylized exemplar. We begin by creating a simple 3D scene that contains all important illumination effects that may subsequently occur in target scenes. A typical example of this scene is “a sphere on a table” (see inset in Figure 8.1a), which bears resemblance to the Lit Sphere [Sloan et al., 2001]. The key difference is that placing the sphere on the table allows us to extend the highlights and shading captured by Lit Sphere to additional illumination effects like soft shadows, color bleeding, and glossy reflections. We render the scene using a global illumination algorithm [Kajiya, 1986] and print it on a paper in low contrast with special alignment marks. The artist then paints on this paper using any preferred media so that the final painting roughly aligns with the dimmed scene preview. We align a photo or scan of the painting with the original rendered image using the alignment marks.

Once the exemplar is ready, a more complex target scene can be created and rendered (see Figure 8.1a). Our algorithm then transfers the hand-painted look from the exemplar to the target scene, preserving the stylization of individual illumination effects.

Implementing this workflow requires two components: (1) a mechanism to calculate light propagation in the scene and (2) an example-based algorithm that uses this light propagation information as guidance to synthesize a target image, preserving the stylization of individual illumination effects in the source exemplar. We describe these components in the following sections.

8.3.2 Light Path Expressions

In light transport, light-surface interactions can generally be classified as either diffuse or specular. Examining the interactions that happen on a path from a light source to the sensor lets us distinguish most important global illumination effects. This technique, known as Light Path Expressions (LPEs), has long been used in rendering [Heckbert, 1990]. One of its uses is to separate the various illumination effects into distinct buffers for the purposes of filtering or in order to use a different rendering algorithm for each.

In our method, we use the same technique to gain insight into the light-scene interactions as seen in image space. By isolating the prominent global illumination effects, we gain additional information about how a real-life scene would appear to an artist, and we can take that information into account in the matching phase of the synthesis. This helps
us make sure that effects like specular highlights, shadows, or diffuse interreflections—all of which are important shape and spatial relationship cues—are stylized consistently with the artist’s intent.

We take advantage of existing renderers’ ability to render these LPE buffers with little computational overhead (we use NVIDIA Iray SDK). We then create multi-channel images of the exemplar rendering and the target rendering, with channels containing the traditional RGB representation and the LPE buffers. All matching operations are performed on these multi-channel pixels.

![Figure 8.4](image)

Figure 8.4: An example of a style exemplar with Light Path Expression images: (a) full global illumination render, (b) direct diffuse (LDE), (c) direct specular (LSE), (d) first two diffuse bounces (LD\{1,2\}E), (e) diffuse interreflection (L.*DDE), (f) hand-drawn style image. Exemplar image © Daichi Ito.

The light paths used in our examples (see Figure 8.4) include direct diffuse and specular components (LDE and LSE), as well as the diffuse interreflection component (L.*DDE) and the first two diffuse bounces (LD\{1,2\}E). Additional channels can be added as necessitated by the given scene (e.g., the caustics channel LS*DE); our synthesis algorithm does not expect any particular number or set of channels and does not require that the path sets captured in different buffers be disjoint.

8.3.3 Synthesis Algorithm

Similarly to image analogies [Hertzmann et al., 2001] our task (see Figure 8.5) is to take three multi-channel images $A$ (exemplar scene rendered with LPE channels), $A'$ (stylized exemplar aligned to the exemplar scene), and $B$ (target scene rendered with LPE channels), and synthesize a new target image $B'$ such that $A : A' :: B : B'$, i.e., that style from $A'$ is transferred to $B'$ according to the similarity between $A$ and $B$. 
To solve this problem Hertzmann et al. originally proposed a simple multi-scale algorithm. For each resolution level and each pixel \( q \in B' \) in scan-line order, a best matching pixel \( p \) is found in the source \( A' \) such that

\[
E(A, B, p, q, \mu) = ||A'(p) - B'(q)||^2 + \mu||A(p) - B(q)||^2 \tag{8.1}
\]

is minimized. Here \( A = \{A, A'\} \), \( B = \{B, B'\} \), and \( \mu \) is a weight that controls the influence of guidance. For any image \( I \), we use \( I(p) \) to denote a feature vector at a pixel \( p \). The vector \( I(p) \) is a concatenation of all pixels in a small square patch of width \( w \) centered at the pixel \( p \), where each pixel can have multiple feature channels. For features, Hertzmann et al. use the intensity value and an output from a steerable filter, whereas Bénard et al. [2013] augment the RGB colors with several additional guidance channels, including temporal coherence and a distance transform. In our case the feature vector contains colors of the full rendered image (RGB) and four LPE channels (each stored as a RGB image):

\[
\{A, B\} = \{\text{FULL}, \text{LDE}, \text{LSE}, \text{L.*DDE}, \text{LD}\{1,2\}E\}. \tag{8.2}
\]

More LPE channels could be added to increase the discriminative power of the feature vector.

Although the original Hertzmann et al. algorithm produces impressive results, it suffers from its greedy nature and can fail to preserve high-level structures (see Figure 8.11f). Subsequent work [Fišer et al., 2014; Barnes et al., 2015] showed that better results can be obtained using the optimization scheme described by Kwatra et al. [2005] and Wexler et al. [2007], which minimizes the following energy:

\[
\sum_{q \in B} \min_{p \in A} E(A, B, p, q, \mu) \tag{8.3}
\]

using EM-like iteration executed multiple times from coarse to fine resolution:

Here \( B'_k \) denotes the current pixel values stored in \( B' \) and \( B'_{k+1} \) the updated values. \( NNF \) is the nearest neighbour field that assigns source patch to each target patch and function \( \text{Average} \) computes the average color of colocated pixels in neighbour patches.

This approach produces notably better results, but as described in Section 8.2, it frequently leads to the wash-out effect (see Figure 8.11i). Kaspar et al. [2015] and Jamriška et al. [2015] mitigate this problem by encouraging uniform source patch usage. However, this restriction is suitable only when each randomly picked sub-region of the source texture is perceived similarly. Enforcing uniform patch usage in our scenario would create artifacts; see Figure 8.7 for a simplified illustration of this problem. For
Algorithm 1: EM-like iteration used to minimize energy (8.3).

**Example**: If the target has comparatively more highlight region than the source, uniform patch usage would force the highlight in the target to be filled with non-highlight source patches (see, e.g., Figure 8.11m, n). In our scenario we need a different scheme that avoids the excessive use of certain patches while still permitting non-uniform utilization. A possible solution would be for each source patch to estimate its optimal usage and then augment the original uniformity-preserving approaches to handle this non-uniform case. However, it is difficult to estimate optimal patch utilization in advance unless we run the actual synthesis. To overcome this chicken-and-egg problem we propose a different approach that inspects the actual matching errors during the synthesis and detects cases when the algorithm starts to force patches into inappropriate locations.

Our solution is based on the idea of reversed NNF retrieval [Rosenberger et al., 2009; Jamriška et al., 2015], in which a best matching target patch is retrieved for each source patch. The advantage of this approach is that it can enforce uniform usage of source patches. However, we must avoid forcing source patches to inappropriate locations in the target. We observe that, in practice, when we sort all matching error values and plot them with normalized axes, the resulting graph has a distinct, hyperbolic shape (see Figure 8.6). It starts with small error values, corresponding to feasible assignments ($A^*$ in Figure 8.6). There is a knee point $k$ where the error starts to increase rapidly. We estimate $k$ by fitting a hyperbolic function $f(x) = (a - bx)^{-1}$ to the data and retrieving the point where $f'(x) = 1$, i.e., $k = \sqrt{1/b + a/b}$. Patches with indices above $k$ are probably erroneous assignments ($A^*$ in Figure 8.6) that need to be avoided. We thus set a feasible error budget $T$ that is an integral of all patch errors with indices below $k$ and modify the original source-to-target assignment in a way that maximizes the number of used source patches $|A^*|$ while satisfying an additional feasibility constraint:

$$\sum_{p \in A^*} \min_{q \in B} E(A^*, B, p, q, \mu) < T \quad (8.4)$$

Such a constrained assignment leads to the desired situation with some target patches remaining unassigned because assigning them would introduce artifacts (c.f. Figure 8.7c). We can repeat the retrieval (c.f. Figure 8.7d) and reuse good source patches to cover remaining positions in the target.

This iterative scheme stops when all target patches are covered. In practice it is feasible to stop even earlier (e.g., when 95% are covered) and use standard target-to-source nearest-neighbour retrieval to speed up the process. The number of iterations depends on the structure of the target scene and the complexity of the exemplar. Typically after 10 iterations more than 80% of target patches have been covered. In the general case, the
number of iterations is roughly proportional to the ratio of the areas of corresponding illumination effects in the source and target images. For example, if the source contains one highlight and the target has four of similar size, then at least four iterations will be necessary to cover them. In practice, the number of iterations is typically slightly higher due to different structures of individual effects.

To complete the algorithm we plug our modified patch assignment process into the original EM iteration (Algorithm 1) by replacing the step where the nearest neighbour field \( \text{NNF} \) is created. We then run the standard coarse-to-fine synthesis.

### 8.3.4 Implementation details

We implemented our technique in C++ and CUDA. To accelerate the retrieval of nearest neighbours, we use PatchMatch with integrated support for masking [Barnes et al., 2009]. To further accelerate the processing we exploit multicore processing using parallel tiling.

**Figure 8.6:** Estimation of the error budget \( T \): the sorted matching errors of all potential source-to-target patch assignments can be approximated by a hyperbolic curve \( f(x) \) on which a knee point \( k \) is detected and used to distinguish between feasible \( A^* \) and erroneous \( A^x \) assignments. The integral of the errors in \( A^* \) gives an estimate of the error budget \( T \).
Figure 8.7: Why enforcing uniform source patch usage is inappropriate in our scenario. (a) We often have the case when different types of patches have different distributions in the source and the target; here the source has much more blue than yellow, but the target requires much more yellow than blue. (b) The uniformity-preserving algorithm initially transfers source patches (marked with gray color) to proper locations in the target. (c) Eventually all suitable target locations can become occupied, leading the algorithm to force remaining source patches (not gray in b) into target positions with high matching error. (d) Our approach detects this erroneous case and restarts the retrieval so that appropriate source patches can be reused to fill remaining suitable positions in the target.

We use fixed patch size $w = 5$ and guidance influence $\mu = 2$. Our pyramid uses 2 for the downsampling ratio, and for a one-megapixel image, we run the synthesis on 6 levels with 6 optimization iterations on each level. The NNF retrieval is accelerated with 6 PatchMatch iterations.

Synthesizing a one-megapixel image takes about 15 minutes on a 3GHz CPU with 4 cores or 3 minutes on the GPU (GeForce GTX Titan Black). In addition to high quality synthesis, we also implemented a preview mode of the algorithm that uses half resolution, compresses LPE channels using PCA, and stores all textures as integers instead of the floating-point numbers used by the high-quality version. This achieves interactive response within 3–6 seconds on the GPU, enabling the applications discussed in Section 8.4.2.

8.4 Results

To validate our method we created a simple “sphere on the table” exemplar scene (see Figure 8.4) and had five trained artists paint it in various styles using different kinds of media, including colored pencils, pastels, acrylic paint, watercolor, pen-and-ink, and markers (see Figure 8.10, top). The artists had the option of including or not including certain effects; for example, they were not required to paint shadow or background if they did not want us to synthesize them.

In practice, a different exemplar scene could have been created to contain specific
Figure 8.8: A collection of 20 different models used to produce the results in this paper and supplementary material. The supplementary material shows the computed LPE channels. Source meshes via TurboSquid: Veleran (a), Gerzi 3D ART (b), shoiko (c), cvbtruong (f), cartomotion (h), luxxeon (i, r), Fernando Luceri (k), sylky (m), Giumann (n), oliverlaric (o), Nagashi (p), WindTrees (q), shiyamon (s).

illumination effects (see, e.g., Figures 8.13 and 8.15), but we found that in most cases very good results could be obtained using a generic “sphere on the table” scene. This proves that our technique generalizes well despite the complexity of the target scene. Once the exemplar was painted we selected various meshes and rendered them under similar lighting conditions (see Figure 8.8), i.e., light positions and materials were similar to the exemplar scene (Figures 8.16 & 8.17 and supplementary videos demonstrate how changing lighting conditions affect the final stylization).

We used different colors to distinguish the object from the background. This helped to avoid patch transfer between different materials and also allowed discrimination among different light interaction effects when one object reflected other, differently colored objects on its surface. The resulting color mixture guided the algorithm to use samples that correspond to a similar interaction area in the source exemplar. In our results only two colors (white and red) were used, however, this color-coded guidance could be generalized to handle more objects with different materials.

The resulting synthesized images are presented in Figures 8.1 and 8.10 with additional results in the supplementary material. Note how accurately they follow the exemplar style. Strokes that correspond to highlights and shadows in the source are consistently transferred to proper locations in the target. Although there is no special treatment for object boundaries, the algorithm synthesizes them convincingly even in the presence of overdrawn strokes (see Figure 8.1c, d). This happens because the discontinuities in LPE channels guide the synthesis to place boundary patches from the source at the
Figure 8.9: The effect of adding individual LPE channels: diffuse (LDE) emphasizes contrast between lighted areas and areas in shadow, specular component (LSE) provides proper stylization of highlights, first and second diffuse bounce (LD\{1,2\}E) emphasizes details in shadows, and diffuse interreflections (L.*DDE) transfer the stylization of indirect illumination. Exemplar image © Daichi Ito.

boundaries in the target. Furthermore, the mechanism for eliminating excessive use of smooth patches encourages the algorithm to use high-frequency patches that typically contain overdrawn strokes.

The feedback from the trained artists who created the exemplars was very positive, with some commenting that many results looked exactly as if they had painted them by hand.

8.4.1 Comparison

We compared our technique with previous example-based stylization algorithms. For this we prepared a source exemplar and a rendering of a target scene where all previously mentioned issues occur; see Figure 8.11. The exemplar (Figure 8.11a) contains distinct stylization of each individual illumination effect and has rich texture details. In the target (Figure 8.11b) the colors of the background and highlights match. The area light is close to the object, so that the assumption of the light being far away is violated, and the distribution of areas representing different lighting conditions is notably different from the source.

The Lit Sphere [Sloan et al., 2001] (Figure 8.11c) uses normals to guide the rendering, causing flaws in the stylization and position of the highlights in the rendered scene.
Because it just uses texture mapping to transfer the appearance in a pixel-wise fashion, there is no patch-wise consistency of the rendered output. The result is that the texture details of the exemplar are not reproduced correctly.

Image analogies [Hertzmann et al., 2001] (Figure 8.11d) and extension [Bénard et al., 2013] (Figure 8.11e) use color for guidance. They fail to retrieve patches from appropriate locations in the exemplar, most visibly by mistakenly using patches from the background for the highlights. The greedy nature of the algorithm leads to texture details being corrupted by artificial seams that break the fidelity of the synthesized image. We extended both approaches using our additional LPE channels (Figure 8.11f, g), but the appearance of texture in the output is still far from the exemplar. Even with the LPE channels, Bénard et al. fails to reproduce highlights because of an additional histogram term [Chen and Wang, 2010] that tries to encourage uniform patch usage.

The deep neural network approach of Gatys et al. [2015] strongly depends on visual patterns used during the training phase. In our scenario this approach completely fails (Figure 8.11h); without proper guidance it is impossible to recover a meaningful assignment between the source and target features.

Figure 8.11i shows how image analogies guided with LPE channels would look when computed using the texture optimization scheme [Wexler et al., 2007] with the original EM-like iteration (Algorithm 1). The wash-out effect is clearly visible because patches with low-frequency content [Newson et al., 2014] have been heavily overused. The variation by Kopf et al. [2007] (Figure 8.11j) fails to take into consideration that the color histogram of the target must be different from the source, so its matching cannot improve the result in our scenario. Using bidirectional similarity [Simakov et al., 2008; Wei et al., 2008] (Figure 8.11k) improves the results only a bit.

The same issue occurs also in the Image Melding method [Darabi et al., 2012] where patches are allowed to change color, rotate and scale and where image gradient channels are used in addition to our LPEs to guide the synthesis (Figure 8.11l).

Enforcing uniform patch usage according to [Jamriška et al., 2015] gives good texture quality, but completely breaks the overall structure (Figure 8.11m). Kaspar et al. [2015] give more flexibility by providing a parameter $\lambda$ that controls the strength of the uniformity enforcement. However, we have found that $\lambda$ must be manually tuned per scene to produce compelling results. Setting $\lambda$ too low does not eliminate the wash-out effect, and setting it too high breaks the structure by enforcing overly uniform patch usage. To get the best result for the target in Figure 8.11b we experimentally set $\lambda = 0.1$. This setting reduces the wash-out effect at the expense of distorting some highlights (Figure 8.11n). Other settings of $\lambda$ produce worse results—see supplementary material for comparison.

Finally, our approach preserves both the textural richness and the overall structure of the target scene without tedious parameter tuning (Figure 8.11o).

In Figure 8.9 we show the influence of individual LPE channels on the resulting synthesis. Despite some changes being rather subtle, they significantly improve the fidelity of the resulting image and makes the style transfer more visually compelling. We also show in Figure 8.12 how our improved synthesis algorithm behaves when used with only colors and normals as a guide. Just using colors (Figure 8.12c) cannot distinguish between the highlights and the background. Just using normals (Figure 8.12d) fails to place the highlights in the correct places, because the relatively close light source places highlights in areas with different surface orientations from the source. Using colors and normals together (Figure 8.12e) works better, but still fails to place all highlights correctly and
Figure 8.10: Results—style exemplars created by two trained artists in different kinds of media (top, denoted by numbers) were applied to models presented in Figure 8.8 (denoted by letters) using our algorithm. Note how the resulting synthesized images (bottom) convey the stylization of lighting effects and preserve the textural richness of the source exemplar. Exemplar images © Daichi Ito (1, 2, 3, 7), Karel Seidl (4), Lukáš Vlček (5), Lucie Svobodová (9), and Pavla Sýkorová (6, 8, 10).
Figure 8.11: To show the need for guidance based on LPE channels, we show the result of our improved synthesis method when based upon different sets of input channels: (c) colors only (d) normals only (e) colors with normals (f) colors with LPE. Only colors with LPE correctly capture the highlights, shadows, and shading effects. Exemplar image © Daichi Ito.
Figure 8.12: Comparison with previous work—an expressive style exemplar (a) has been applied to a scene with five toruses (b) using previous approaches to example-based stylization (c, Sloan et al. [2001]), (d, Hertzmann et al. [2001]), (e, Bénard et al. [2013]), (h, Gatys et al. [2015]), (i, Wexler et al. [2007]), (j, Kopf et al. [2007]) and techniques for general patch-based texture synthesis (f, Hertzmann et al. [2001]), (g, Bénard et al. [2013]), (k, Simakov et al. [2008]), (l, Darabi et al. [2012]), (m, Jamriška et al. [2015]), (n, Kaspar et al. [2015]). Different approaches use different guidance channels: normals (N), colors (RGB), LPEs (LPE), and no guidance (N/A). Note how our approach (o) better preserves the visual structure of the target and the textural details of the source (see insets below). Exemplar image © Pavla Sýkorová.
fails to reproduce some of the shading effects. Only using colors combined with LPE (Figure 8.12f) correctly reproduces all the lighting effects of the target.

### 8.4.2 Applications

Our approach has numerous potential applications. It can be used to preview a target rendering style on various different geometries (see Figure 8.10) or to test out multiple rendering styles on the same geometry (see Figure 8.1). Alternatively, exemplars made by experienced artists can be used by others to produce stylized digital content.

It can also be used in animation (see supplementary video) or for autocompletion [Xing et al., 2014, 2015]. The user can stylize a single frame or only a portion of the model (see Figure 8.13) and then use our technique to transfer the hand-drawn look to the rest of the model or sequence.

![Figure 8.13: Autocomplete shading—an artist draws shading for one finger (b) and our method synthesizes (c) the rest of the hand (a). Exemplar image © Daichi Ito.](image)

![Figure 8.14: On-the-fly shading study—the artist paints a simplified “sphere on the table” scene (top row) and watches as the stylization is transferred to the target model (bottom row). Such a continuous feedback helps to gradually improve the final stylization. Exemplar image © Karel Seidl.](image)
Artists and designers whose skills are insufficient to create a convincing painting of a detailed scene can instead create a “shading study” (as in Sloan et al. [2001]), and then let our method transfer the appearance to the target model automatically. Thanks to our GPU implementation this process can be performed on the fly (see Figure 8.14) using a camera mounted above the canvas. The artist can improve the study based on immediate visual feedback. In Figure 8.14, after seeing the rendering preview of the target model (bottom row), our artist decided to increase the color contrast and make the shadows darker. This approach can also be useful when an artist needs to paint an entire model by hand but would appreciate some reference visualizing how the actual stylization would look like on a more complex surface. Our technique can also be used to transfer style from an existing painting when a reference scene does not exist. We exploit existing techniques for 3D reconstruction from a single image [Zeng et al., 2015] and manually set up lights to mimic the illumination conditions of the original painting.
Then we render corresponding LPE channels and feed them into our stylization pipeline to obtain a roughly corresponding stylized target scene (see Figure 8.15).

8.5 Limitations and Future Work

Although our approach works quite well for many hand-painted exemplars and many target 3D scenes, there are some limitations that must be taken into account.

In Figure 8.10 we demonstrate that our technique does a good job in generalizing illumination effects from a simple exemplar to relatively complex scenes. However, roughness in the style exemplar can sometimes suppress fine geometric details (see Figure 8.16b, k, g). This effect can be alleviated with an additional edge map channel that emphasizes visually important features (Figure 8.16c, d, m).

Our method produces best results when the lighting environment is similar to that in the exemplar scene. While some illumination changes can be handled (see adding a light source in Figure 8.16f, h) we assume that all important illumination effects present in the target are present in the exemplar scene. For example, if the target scene has a dark shadow that is not present in the exemplar, our method cannot find proper patches and fills that area with inappropriate content, producing artifacts (see Figure 8.17). We can sometimes alleviate this by matching the appearance of the exemplar’s rendering with the rendering of the target scene (Figure 8.16i, j, q).

Figure 8.17: Limitation—when the lighting environment is similar to that in the original exemplar scene (a) our approach produces best results (b). It can fail (c) when some illumination effects are missing in the source exemplar—here, highly shaded areas with little indirect illumination. Exemplar image © Karel Seidl.

When the rendering of the target scene is even more complicated, like having multiple objects with interreflections or challenging lighting effects like caustic or subsurface scattering, a different exemplar must be prepared and new LPE channels must be added. Our method is also not suitable for highly exaggerated stylization that does not closely
fit the exemplar 3D scene—for example, when a stylized highlight is drawn where there is no rendered highlight.

In the future we would like to extend our technique to better handle animations. We plan to incorporate temporal coherence between stylized frames and give artistic control over the perceived temporal noise in the spirit of Fišer et al. [2014]. We would also like to explore how our synthesis algorithm with the error budget can address more general texture synthesis problems.

8.6 Conclusion

We have presented an approach to example-based stylization of 3D renderings that takes into account illumination effects. It includes an extended synthesis algorithm that better preserves the visual richness of hand-created style exemplars. Both the general approach and the synthesis algorithm dramatically improve the fidelity of the stylized images. We have demonstrated that our technique can handle a great variety of different stylizations. Our approach confirms the great potential of example-based techniques, and we hope it will inspire others to explore further their applicability in the field of non-photorealistic rendering.
CHAPTER 9. CONCLUSION

9 Conclusion

Six novel methods have been presented that allow for the example-based rendering of content in hand-drawn style and for preprocessing raw user inputs. This chapter summarizes the contribution and novelties of our work, discusses the concurrent development and proposes possible future topics.

9.1 Summary

In Chapter 3, Painting by Feature was presented, a method to “paint with images”. Most texture examples can be separated into the boundary and the interior part and each of them is treated in a different way. We propose a method for faithful synthesis of 1D boundary features in the two-dimensional image space that previous methods could not handle. Combined with a texture synthesis tool, Painting by Feature provides a comprehensive set of instruments to create unique artistic content.

Chapter 4, Color Me Noisy, presented an approach to control the temporal noise in a video sequence and to create its stylized version. This goes against most of the concurrent methods that try to suppress the temporal noise entirely. However, the temporal noise is inevitable in actual hand-painted animation and it serves as another mean of expression. We proposed a way to efficiently control this phenomenon within an existing well-established texture-synthesis framework.

LazyFluids, described in Chapter 5, is a novel method for appearance transfer for fluid animations. A traditional way to create various fluid-based effects in contemporary production pipelines often involves working with a large set of existing pre-captured footage, or running and tuning a computational-heavy 3D fluid simulation and using advanced rendering algorithms. In contrast, our approach uses efficient real-time 2D fluid simulation and enables the artist to stylize it with either a single image or a video sequence of the particular media. The key contribution we presented is the method to enforce uniform usage of source exemplar patches, thus retaining its richness and textural qualities and preventing the so-called “wash-out” effect.

In Chapter 6, we presented Brushables, a method that extends painting with textures to examples with complex boundary effects, that Painting by Feature was unable to handle. We proposed a novel approach that integrates the shape and direction of the texture into the similarity metric and showed the tool that manipulates both to achieve compelling results.

Chapter 7 introduced ShipShape, the technique to beautify raw hand-drawn sketches by applying general geometric and aesthetic rules such as symmetry or parallelism. This allows the users to maintain the simplicity and speed of freehand sketching while taking advantage of the implicit geometric constraints emerging in the forming drawing.

In Chapter 8, we proposed StyLit, an example-based stylization method for 3D renderings. We built upon our synthesis scheme presented in LazyFluids and developed a way to automatically determine the optimal portion of the source example that should be used to stylize the target rendering. In addition, we employed light path expressions to guide the synthesis. This novel type of guidance can better distinguish among context-dependent illumination effects, for which artists typically use different stylization techniques, and delivers a look closer to the realistic artwork.
9.2 Concurrent and Future Work

Several approaches to synthesize curvilinear features were presented, such as the *DeCoBrush* by Lu et al. [2014] or method of Zhou et al. [2013]. These methods produce comparable results to *Painting by Feature* but are specialized to a particular type of input or lack the ability to synthesize textured areas, which limits their scope of use.

In parallel to *LazyFluids*, Kaspar et al. [2015] noticed the importance of uniform patch usage in the patch-based texture synthesis. To this end, they proposed a new similarity scheme that takes into account the per-patch assignment count and penalizes heavily overused patches to encourage other patches to be used. However, unlike the algorithm we presented in *LazyFluids*, this approach does not guarantee true patch usage uniformity, tends to fail in the case of inhomogeneous textures and involves careful tuning of the sensitive penalty parameter.

New interesting methods were proposed to mimic or simulate particular types of artistic media or styles, such as oil paint [Chen et al., 2015; Semmo et al., 2016], watercolor [Montesdeoca et al., 2016], line-drawn sketches [Ben-Zvi et al., 2016], or simplified cartoon-like stylizations [Rosin and Lai, 2015]. These approaches often allow for much better control of the characteristics of the given media, e.g., drying speed of the oil paint, but are only bound to their specific styles.

**Figure 9.1**: A Neural Algorithm of Artistic Style [Gatys et al., 2015]. The system uses convolutional neural network representations to separate and recombine content and style of arbitrary images, providing an algorithm capable of production of artistic imagery.

Gatys et al. [2015] opened new ways to general image and video stylization by using deep neural networks to separate and recombine the content and style from different inputs. This approach spawned many follow-up works, such as, e.g., the *Neural Doodle* [Champandard, 2016], another variation of the texture-by-numbers paradigm. The advantage of these methods is their versatility with respect to the style being used. However, they often fail to fully capture the texture of the example at the fine scale. As another demonstration of the usability of convolutional neural networks, Simo-Serra [2016]...
proposed a method to simplify rough sketches by training the network on sample pairs provided by trained artists.

In conclusion, the non-photorealistic rendering and stylization has gotten increased attention in last few years. The methods we presented in this thesis provided new means to solve some of the example-based texture synthesis’ problems that were preventing from its wide use. We believe that a combination of these methods together with the emerging deep-neural-network-based approaches could lead to a whole new family of powerful stylization tools that would enable, for the first time, to overcome the uncanny valley between actual hand-crafted artistic content and computer generated imagery.
REFERENCES

References


REFERENCES


REFERENCES


REFERENCES


A Author’s Publications

The following publications were co-authored by the author of this thesis and published in high-impact journals, as indexed by ISI.


Cited in:


Cited in:


**Cited in:**


Jakub Fišer, Paul Asente, Stephen Schiller, and Daniel Sýkora. Advanced drawing beautification with ShipShape. *Computers & Graphics*, 56:46–58, 2016a (IF = 1.120)

**Cited in:**


**Cited in:**

B Authorship Contribution Statement

This statement describes the specific contributions of the author of this thesis to the publications presented therein.

1 Painting by Feature: Texture Boundaries for Example-based Image Creation (20%) In Painting by Feature, I came up with the way to combine the novel boundary synthesis with the interior-filling texture synthesis and integrated it into the prototype application. I created most of the results presented in the paper, as well as the comparisons with previous methods, and helped to write the related paper sections – our approach, applications and results.

2 Color Me Noisy: Example-based Rendering of Hand-colored Animations with Temporal Noise Control (30%) In Color Me Noisy, I developed the idea of controllable temporal noise in an animation to mimic hand-created artwork and enabled new ways to control expressiveness of the video. I wrote the method section of the paper, implemented the prototype tools to create animated content with the presented method, and created most of the results.

3 LazyFluids: Appearance Transfer for Fluid Animations (27%) In LazyFluids, I expanded the idea of reversed nearest-neighbor-field search and created a proof-of-concept tool. Also, I heavily experimented with other optimization strategies for the same task. I helped to develop the synthesis pipeline and other tools to create the video results and helped to write the core parts of the paper – problem formulation and description of our method.

4 Brushables: Example-based Edge-aware Directional Texture Painting (15%) In Brushables, I participated in formulating the paper concept, assisted with the comparisons with concurrent methods and created many of the results in the paper.

5 Advanced Drawing Beautification with ShipShape (60%) In ShipShape, I developed the method to interactively beautify the hand-drawn sketches. I created the prototype application, wrote the vast majority of the paper, created most of the presented results and conducted the user study to evaluate the usability of our approach.

6 StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings (40%) In StyLit, I extended the idea behind the LazyFluids into the stylization of 3D renderings guided by the light-path expressions and came up with the idea of using light path expressions as a novel way of synthesis guidance. I wrote the outline of the paper as well as the main method-explaining part. I also created the synthesis pipeline and developed a method to determine the threshold between “good” and “bad” patch-to-patch correspondences.