Data-driven
Digital Drawing and Painting

Jingwan Lu

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Adam Finkelstein

June 2014

“If you hear a voice within you say you cannot paint, then by all means paint and that voice will be silenced.”

—Vincent van Gogh

“Everyone who has any talent at all in sketching, painting, sculpturing or carving, should have the opportunity to use that talent. The expression is important for the person, and can tremendously enrich the lives of other people.”

—Edith Schaeffer

“All you need to paint is a few tools, a little instruction, and a vision in your mind.”

—Bob Ross

Abstract

Digital artists create evocative drawings and paintings using a tablet and stylus coupled with digital painting software. Research systems have shown promising improvements in various aspects of the art creation process by targeting specific drawing styles and natural media, for example oil paint or watercolor. They combine carefully hand-crafted procedural rules and computationally expensive, style-specific physical simulations. Nevertheless, untrained users often find it hard to achieve their target style in these systems due to the challenge of controlling and predicting the outcome of their collective drawing strokes. Moreover even trained digital artists are often restricted by the inherent stylistic limitations of these systems.

In this thesis, we propose a data-driven painting paradigm that allows novices and experts to more easily create visually compelling artworks using exemplars. To make data-driven painting feasible and efficient, we factorize the painting process into a set of orthogonal components: 1) stroke paths; 2) hand gestures; 3) stroke textures; 4) inter-stroke interactions; 5) pigment colors. We present four prototype systems, HelpingHand, RealBrush, DecoBrush and RealPigment, to demonstrate that each component can be synthesized efficiently and independently based on small sets of decoupled exemplars. We propose efficient algorithms to acquire and process visual exemplars and a general framework for data-driven stroke synthesis based on feature matching and optimization. With the convenience of data sharing on the Internet, this data-driven paradigm opens up new opportunities for artists and amateurs to create original stylistic artwork and to abstract, share and reproduce their styles more easily and faithfully.

Acknowledgements

First of all, I would like to thank my advisor, Adam Finkelstein, for his support throughout my PhD process. I was extremely fortunate to have the opportunity to work with him. Adam has always been extremely nice, patient and encouraging. He has provided tremendous amount of inspirations and advice for my academic research as well as my personal development.

A big thank goes to my husband, Qi Fei, for his love and encouragement. We have been separated for five years during my pursue of PhD. It was hard for both of us to maintain a long distance marriage and to pursuit our career development on two different continents. People have asked me "How did you guys get through these years? What keep you together?" I say he loves me for whom I am and supports me wherever I am and whatever I do, and I do the same too. Throughout these years, he has always been a good listener, supportive for every decision I have made. He never complained about my lack of time with him and always put up with my emotions, complaints and stubbornness. Facing all the deadlines, stress and hardship, he is always there encouraging and comforting me and most of all he deeply believes in me. I am really looking forward to getting together with him in the near future.

I would also like to thank my parents, without whom my life would not be possible. They both are diligent engineers and have been my role models as I grew up. I picked up my interest in computer science from my father, an excellent civil and software engineer, who pioneered a series of bridge design CAD software in China. Personality wise, I learned from them the importance of being an independent, honest and responsible person. I am also deeply influenced by their "work hard and play hard" spirit. I myself am determined to do the same, living my life to the max. Mom and Dad, thank you !! I hope to continue to make you proud.

I also deeply appreciate my collaborators and friends from Google, Adobe, and University of Virginia, Stephen DiVerdi, Paul Asente, Radomír Měch, Holger Winnemöller and Connelly Barnes for their countless valuable meetings and discussions. I am very lucky to be able to work with these amazing people for the past few years and enjoy the pleasant west coast in San Francisco, Seattle and San Jose. Stephen is like my second advisor. I received a lot of help and learnt a lot from him. He is also a talented artist who can make awesome videos !!

I am also very grateful for all my teachers, lab-mates and friends, Szymon Rusinkiewicz, Tom Funkhouser, Jianxiong Xiao, Thiago Pereira, Ohad Fried, Xinyi Fan, Sema Berkiten, Yiming Liu, Fu Yu, Tianqiang Liu, Pingmei Xu, Maciej Halber, Nanxi Kang, Chen Song, Jiasi Chen, Connelly Barnes, Xiaobai Chen, Linjie Luo, Siyu Liu, Vladimir Kim, Aleksey Boyko, etc., for their friendship and quality time with me. I will always remember those wonderful dinners, events, all of your encouragement and jokes, all those weekends we spent together in the lab, more and more. Without all of you, my life at Princeton would not have been so colorful and enjoyable.

Special thanks to talented artists at Adobe, Daichi Ito and Daniela Steinsapir, for their valuable feedback and artworks, which are great help and inspirations for our projects.

This work was supported in part by generous gifts from Adobe, Google and Intel.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Digital art is gaining popularity among artists due to its broad range of aesthetic freedom, capability of programmatic enhancements such as undo, repeat, and edit, and ease of transmission. Non-Photorealistic Rendering (NPR) emerged in the 90s and focuses on enabling a wide variety of expressive styles for digital art, [44] and [129]. One major pursuit in NPR is concerned with automatic rendering techniques that directly stylize visual input such as images, videos and 3D models for chosen artistic styles. However, for many applications, it is more important to bring the artists into the design loop. Therefore, a more recent pursuit in NPR focuses on digital simulation of the traditional painting process, giving artist full control over the placement of strokes while letting computer to figure out the rendering details. Along this line of research, we propose a novel data-driven paradigm for digital painting simulation. We design efficient algorithms and intuitive user interfaces for novices to more easily create digital art with the help of high-quality exemplars.

This chapter introduces the digital drawing and painting problem, reviews the related work, discusses the challenges and the thesis organization, elaborates our data-driven paradigm, and concludes with our contributions in digital drawing and painting.

Figure 1.1: Lascaux cave painting, 15000 BC

1.1 Digital Drawing and Painting

Drawing and painting are universally appreciated forms of art gaining popularity throughout the history of mankind. From the ancient cave and wall paintings (Figure 1.1), to the Renaissance masterpieces (Figure 1.2), to the contemporary abstract visual representations (Figure 1.3), painting has been the most versatile and evocative form of visual art that expresses artist's inner emotions, thoughts and stories and inspires viewer's imaginations and interpretations. The work of great artists manifest centuries of art knowledge and practices and possess enduring artistic values that many of us are still trying to understand and recreate today. Beyond their artistic value, drawings and paintings are often appreciated as effective means of visual communication. Compared to photos, paintings abstract the visual information and direct the viewer's attention to the important content. For example, fashion designers, architects, etc., use paintings and drawings to convey their design concepts (Figure 1.4 and 1.5).

Painting is an extremely captivating and fun activity that both kids and adults enjoy. However, in the words of Baxter, *"painting is something simple enough that a child can do, yet deep enough to require a lifetime to master"* [14]. To effectively maneuver brushes to visualize and express something in the painter's mind can take significant amount of training and practice. Even for experienced artists, painting

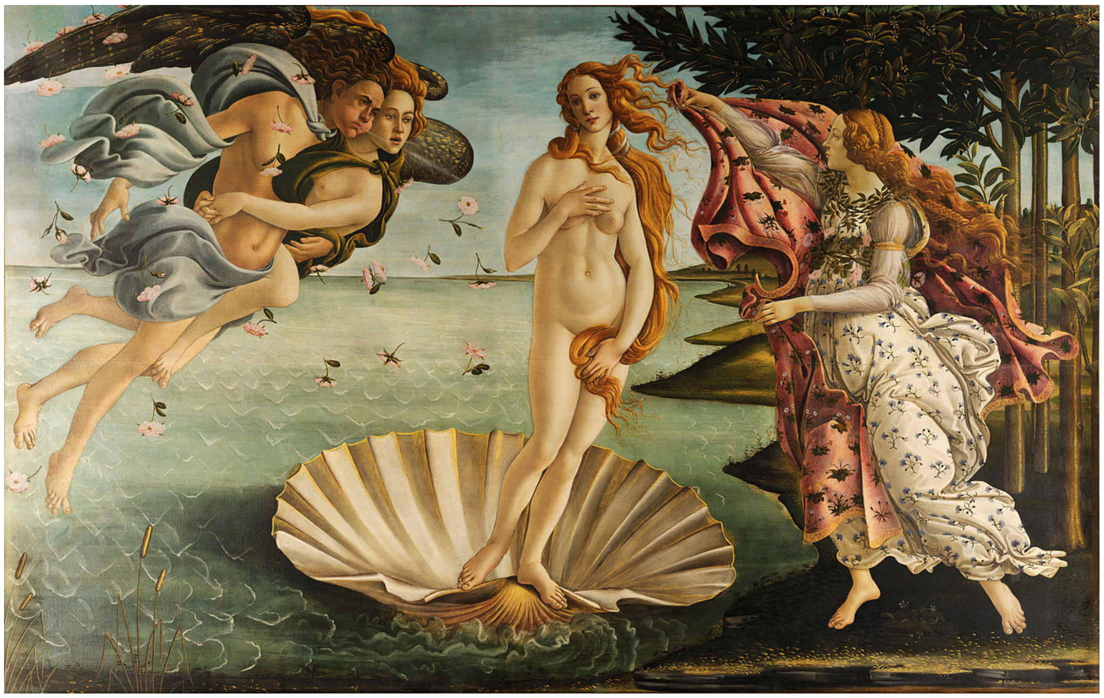


Figure 1.2: Birth of Venus, Sandro Botticelli, 1486

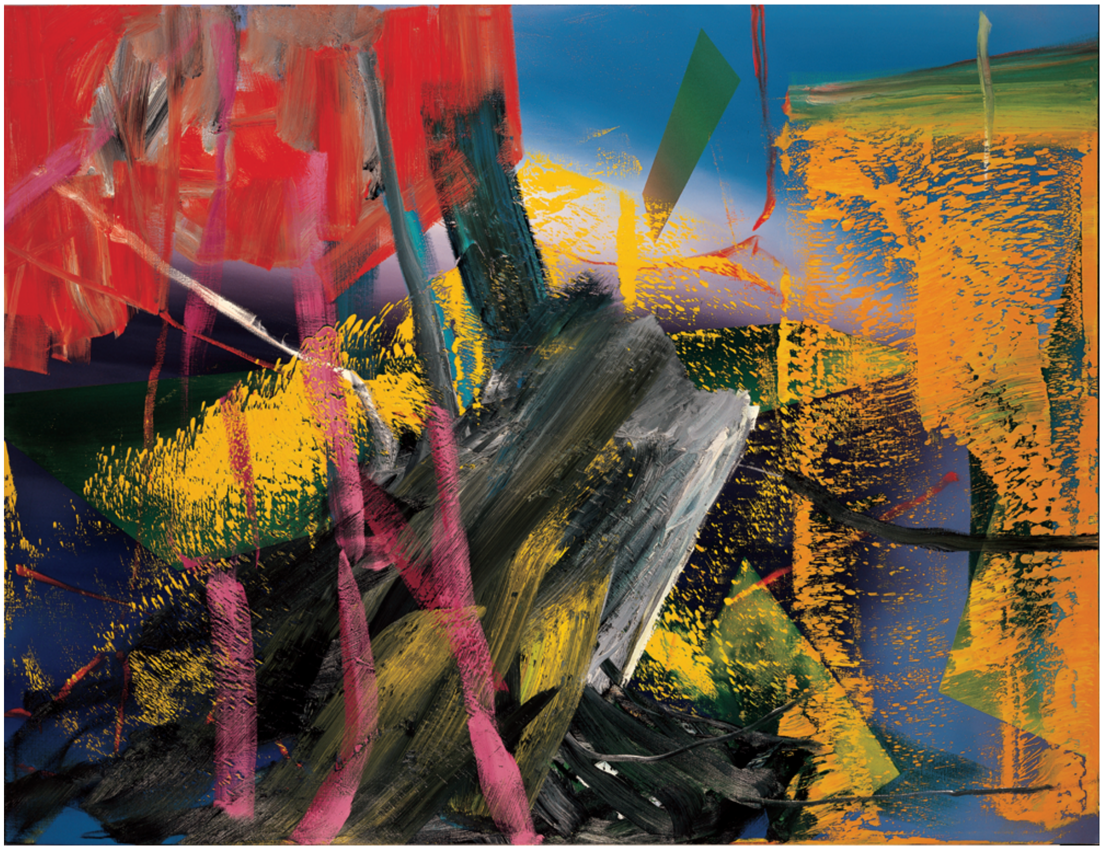


Figure 1.3: Eule, Gerhard Richter, 1982

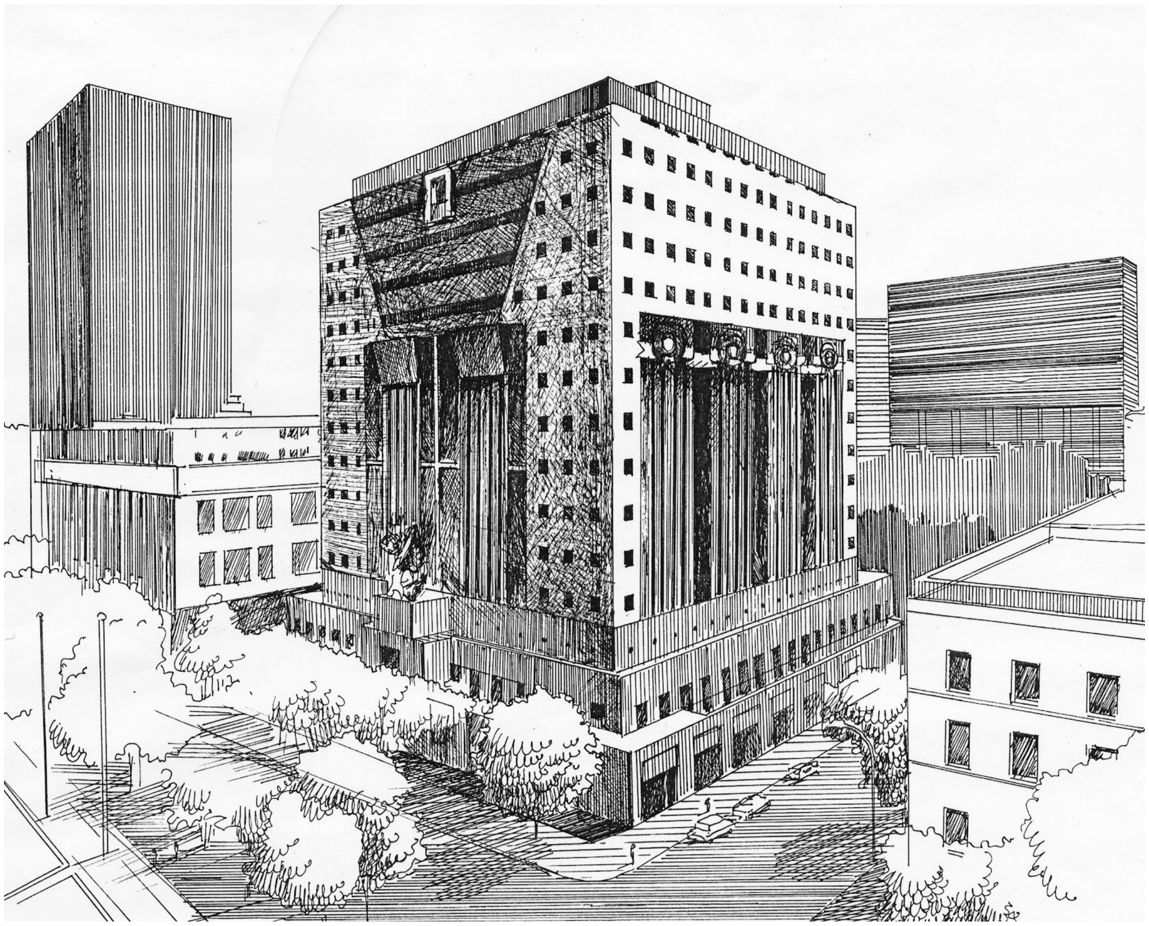


Figure 1.4: Portland building, arch. Michael Graves

can be tedious at times; pieces are started over and over and can take months or years to finish.

In recent decades, digital painting technology has become an important and ubiquitous medium. The general research goal in digital painting is to bring the artistic freedom and expressiveness into the digital world, taking advantage of the computer to make art creation increasingly accessible for casual users and experts alike. Digital painting tools provide a qualitatively similar interactive process, but lower the barrier to entry of art creation. People can now quickly paint something in computer and progressively refine their result using unlimited digital undo, redo and modifications. Tedious physical setup and cleanup, running out of paint, and waiting for paint to dry are no longer concerns in the digital world. Digital art evolves as a brand new art form, creating styles and effects that are not physically achievable and unique in their own ways. New generations of artists are trained to work with the capabilities and limitations of the digital painting medium to produce artistic imagery for various applications, including movies, games, advertisements, designs etc.

Figure 1.5: Marine fashion collection, 2009-2014, Callista1981

1.2 Related Work

Much of the previous work in NPR has focused on designing automatic algorithms to stylize images, videos, or 3D models for selective artistic styles. Survey papers by Bénard [16] and Kyprianidis [79] give a comprehensive discussion of this line of research. With an automatic stylization algorithm, once an input image is selected, users only have limited control over the final rendering by dragging sliders of a set of parameters. The effects of the parameters are often global to the whole image and sometimes not intuitive to understand. Local and precise user control is sometimes more desirable, for the following reasons: automatic algorithms sometimes require precise user intervention to obtain best results, and the artistic goals cannot always be articulated a priori by the user using an input picture and a few parameters. For these reasons, we have witnessed the rise of interactive digital painting technology in the last decade. From commercial software to research systems, digital painting technology has undergone a series of revolutionary changes. This section gives a brief overview of the recent advances in stroke-based digital painting simulation. For a more complete discussion, we refer the readers to survey by DiVerdi [29].

1.2.1 Procedural Approaches

The earliest work in digital painting applied a combination of heuristics, procedural rules and image filters to synthesize strokes in artistic styles. To render a stroke mark, some procedural approaches use simple sweeping and stamping methods. The idea is to continuously sweep 2D bitmap primitives [30, 66, 111], or intersect 3D geometries with the canvas [127, 128, 131], along the input trajectory. Procedural rules have also been used to model the uni-directional ink transfer from the brush to the canvas [98, 99], or the bi-directional pigment transfer where the brush not only deposits but also picks up pigment on the canvas [12, 19]. Previous work in watercolor and calligraphy use a diffusion process [92, 98, 99], or random walk based procedural rules [32, 33, 64], to approximate the dynamic behavior of pigment propagation through canvas media. Procedural approaches generally do not require heavy computation or memory consumption and therefore are widely adopted by commercial painting software, such as Corel Painter [23], Ambient Design Art Rage [28], Autodesk SketchBook Pro [5] and Microsoft FreshPaint [24]. Such applications are continuously advancing, increasingly capable of creating images faithful to hand-drawn art. However, the results are sometimes criticized as "too clean" and not yet realistically reflecting the complexity, richness and spontaneousness of real media.

1.2.2 Physical Simulation Approaches

In recent years, research in digital painting are focused on applying physics knowledge to improve the realism and organic quality of the simulation results. A lot of effort has been made to model a deformable virtual brush and its physical interaction with the canvas. For the best results, the user needs to paint with a tablet and a stylus, which keep track of the user's hand-pose to control the virtual brush simulation engine. The tip of the brush can be physically simulated with a center spine using the spring model [117], or multiple discrete spines [10, 20, 137], or more sophisticatedly individual bristles for most dynamic brush deformation behaviors [31, 92]. To faithfully simulate fluid-like pigment propagation for oil and watercolor, previous work have used 2D flow simulation [21, 25, 136], or 2.5D flow (height field) [13], or even full 3D flow for the dynamic effects of heavy and viscous oil paints [11, 105]. To simulate the realistic appearance of pigments, Kubelka-Munk (KM) model has been introduced to the digital painting community, which physically models the multi-spectrum optical behaviors of paint layers [75, 76]. In order to apply KM or an approximation, pigment coefficients (absorption and scattering coefficients) need to be physically acquired and complex non-linear equations need to be evaluated per pixel at runtime [11, 13, 25, 72, 73, 115, 135, 136]. Physical simulations dramatically improve the synthesis fidelity, producing excellent results for a few natural media such as oil, watercolor, pastel and crayon. However, accurate physical simulations are computational expensive even on high-end hardware. They offer more faithful user experience to the physical painting process, but require special input devices and the skills to control those devices, to some extent similar to that of using a physical brush. Other factors such as robustness and low output resolution also limit the commercial uptake of existing research.

1.2.3 Data-driven Approaches

In the past few years, a few data-driven techniques have been introduced into digital drawing and painting. The work of Xu et al. [151, 153, 154] proposed a data-driven brush model to simulate realistic Chinese calligraphy. They used real calligraphic strokes to train the parameters of a set of writing primitives — solid geometries that represent clusters of brush hair — for the purpose of synthesis. Without explicit brush modeling, statistical models can be trained on real Chinese characters to directly generate strokes of new styles [150]. Other work learned the mapping between input strokes and output diffused shapes to simulate the ink diffusion process [152]. More recent work explored the idea of data-driven whole stroke synthesis, which uses libraries of acquired stroke exemplars to create novel strokes using matching and stitching algorithms [3, 70]. The idea has also been extended to go beyond natural media painting to allow synthesis of real-world textures [93, 113], or even structured patterns [163], using a painting-like interface. This thesis aims at advancing the data-driven idea for more general digital painting applications.

1.2.4 Sketch-based Interface

Sketch-based interface and its applications is another category of related work. In 3D modeling, SKETCH [161] provides a gestural interface for creating rectilinear models such as furniture, based on a few 2D lines drawn by the user. Teddy [60] supports the design of freeform models based on the user's 2D freeform strokes. In character animation, previous work used a series of drawings of a skeleton pose to synthesize an animation sequence [27]. The user can also draw the desired trajectory of a character to guide the transversal of the motion graph [74], or to control the playback of recorded motion segments [55, 133]. Other work in computer graphics have also used sketch-based interfaces for tree and flower modeling [63, 103], and for garment design and manipulations [59, 134]. The common challenge of sketch-based applications is to infer high dimensional output (3D models, animations, etc) from low dimensional input (2D sketches). In digital painting simulation, a stroke-based technique is commonly used to render the artistic intent of the user. It solves a similar low-dimension to high-dimension inference problem, where the input is still the 2D sketch, but the output is the shape and appearance of the stroke. Stroke-based techniques [50] provide a broad range of stylizations, as each stroke can be arbitrarily shaped and textured, by example or by physical simulations.

1.3 Remaining Challenges

Artists have long been trained to maximize the expressive power and artistic potentials of whatever artistic tools at hand. Despite the limitations and constraints of digital painting technologies, great artists can always create amazing art pieces. However, creating visually compelling painting is still hard for novices. And most digital painting results are still criticized as "too clean, too sterile" [14]. Physical simulation systems improved the "organic" quality and "complexity" of digital painting results. However, there are several remaining challenges:

- **Input Hardware requirements.** High-fidelity simulation usually requires high-end input device, such as tablet and stylus. A common obstacle for the adoption of digital painting technologies is the inaccessibility of such devices and the steep learning curve to wield them effectively.

- **Only supporting a few common media.** Physical simulations have focused on common artistic styles such as oil and watercolor painted on canvas with brush. However, progressive artists can freely express their creativity utilizing different instruments to apply various natural media on distinct substrate surfaces, limited only by their imagination. Designing simulation for each of these innovations is a daunting process. Furthermore, accurate high-quality simulations require expensive computation and abundant memory resources. They also expose various parameters that might not be intuitive for novices.

- **Lack of support for structured patterns.** Very few effective simulations or procedural approaches have been proposed for the synthesis of strongly structured vector patterns following sketch-based interface. Most structured designs are still produced from scratch using software like Adobe Illustrator [65].

- **Physical measurement for color prediction.** In order to faithfully reproduce paint-like color effects, most simulation systems require tedious physical measurement of the pigment properties, which is needed for each new type of medium. Therefore, most state-of-art digital painting software still use linear RGBA compositing, which is simple but hardly faithful to the complex optical appearance of real pigments.

1.4 Thesis Overview

In this thesis, we propose a new data-driven painting paradigm that allows novices to more easily create visually compelling artwork with the help of high-quality exemplars. Section 1.5 elaborates the data-driven idea and describes its benefits and challenges. Section 1.5.4 proposes the concept of painting process factorization, which enables fast exemplar acquisition and processing and reduces the solution space for run-time stroke synthesis. We introduce four prototype systems, HelpingHand (Chapter 2), RealBrush (Chapter 3), DecoBrush (Chapter 4) and RealPigment (Chapter 5), that follow the data-driven synthesis framework. As pre-processing, we acquire and process libraries of exemplars created by professional artists. At runtime, given a new input trajectory, we predict the remaining information of the stroke based on the chosen exemplar library. Section 1.5.5 provides an overview of the piecewise matching algorithm (shared by HelpingHand, RealBrush and DecoBrush), which addresses the problem of mapping a low-dimensional input trajectory to high-dimensional stroke properties.

In the remaining chapters, we describe each prototype system that addresses one of the digital drawing and painting challenges highlighted in Section 1.3:

- **No need for high-end input hardware.**
  HelpingHand (Chapter 2) learns from hand gesture data to hallucinate the possible hand pose (pressure, tilt and rotation) used during drawing, based only on x, y positions obtained from low-quality input devices. Additionally, the system can also stylize the input trajectory by example. This eliminates the need for a high-end input device wielded by an expert hand, enabling novices using touch screens and mice to draw in the gestural style of experts.

- **Painting with new natural media.**
  RealBrush (Chapter 3) uses photographed natural media stroke marks to stylize

the user's query path, completely obviating the need for physical simulation. It also stylizes the stroke interactions such as paint smearing and smudging based on captured physical examples. It provides a general synthesis framework for natural media beyond the commonly seen oil, watercolor and pastel. We demonstrate results synthesized with various exotic media that have never been physically simulated before, including plasticine, lipsticks, heavy gel mixed with paints, glitter, etc.

- **Painting with structured patterns.**
  DecoBrush (Chapter 4) further generalizes the conventional "painting" concept beyond the scope of natural media to allow synthesis of structured vector patterns following a sketch-based interface. It advances the concept of "design by sketching", where the user only needs to supply a 1D spine as input and the system automatically figures out the possible structures to grow along the input spine. This interface makes the vector design problem much more accessible for novices, allowing easy modification of existing expert designs for new applications.

- **A simple approach to realistic color prediction.**
  RealPigment (Chapter 5) proposes a data-driven color model which predicts the result color of compositing a new layer of pigment on top of the background layer, based on a physical color compositing chart. Our data-driven color model eliminates the tedious measurement process needed for Kubelka-Munk theory and improves the color prediction fidelity compared to the commonly used RGBA compositing.

We have previously described the systems presented in Chapters 2-4, [91, 88, 89]. This thesis pulls the work together into a common framework, and also describes work in Chapter 5 that remains in preparation for subsequent publication [90].

1.5 Data-driven Painting Paradigm

The recent "Big Data" trend entails a strategy of using simple algorithms and lots of data to answer questions. It obviates the need to make simplifying assumptions or use computationally-intensive algorithms and models to approximate the physical phenomena. With the abundant information present in the data, one can directly point at the data and say "this is what the world really is". Many previous efforts in image editing and computer vision successfully build on this idea, [34, 48, 147]. However, in the digital drawing and painting domain, the Big Data idea has not been directly applied, due to the limited CPU and memory resources and the stringent requirement for interactive performance. Therefore, this thesis focuses on designing data-driven techniques to work under such resource and performance constraints. This section talks about the general data-driven methodology (Section 1.5.1), its benefits (Section 1.5.2) and challenges (Section 1.5.3) for digital drawing and painting, and our specific strategies to address these challenges (Section 1.5.4 and Section 1.5.5).

1.5.1 Methodology

Powered by the availability of Internet-scale image and video collections coupled with greater processing speeds, the last decade has witnessed the rise of the data-driven approaches in computer graphics. The animation community has long used motion capture data to control virtual avatars [81, 87]. More recently data-driven approaches have also been used for animating deformable objects [107]. Efforts have also been made to synthesize geometry by example [41, 160], or manipulate, enhance or re-target photos and videos by example, [85, 86]. Some other research directions, such as texture synthesis and image-based rendering, are inherently data-driven. Texture synthesis addresses the problem of algorithmically constructing a large image from a small sample image by taking advantage of its structural content. Image-based modeling and rendering algorithms rely on a set of 2D images to reconstruct 3D models and render them from novel viewpoints. The general methodology of the data-driven approaches is: first design the amount and type of data to collect, then acquire the data in a controlled way, perform analysis and/or training on the data, and finally use the data to perform prediction or synthesize new variations.

There are parametric methods and non-parametric methods. Parametric data-driven methods usually assume an underlying parametric model to represent the data, then the acquired data is used to train/optimize the model parameters. The prediction result is highly dependent on the feasibility of the model and the quality and quantity of the data. Special efforts need to be made to address the problem of local minimum for non-linear objective functions and over-fitting when there is not enough data. Along this line, in Chapter 5, we attempt to use color compositing chart to predict the parameters of the Kubelka-Munk color model, which can then be used to predict the compositing behavior given new query colors.

Non-parametric data-driven methods, on the other hand, adopt slightly different methodology. They avoid the training phrase, but directly modify and apply the acquired data for synthesis purposes. In general, there are two types of approaches. One is interpolation and extrapolation. For example, some animation algorithms directly interpolate motion capture data or key frames for the synthesis of intermediate frames. Usually, the motion capture database is large and the key-frames are frequent, therefore the resulting blurriness is not objectionable, or even desirable for smooth motion. Another type of approach, that has wider range of applications, chops data into discrete parts and then smartly stitches them together to create interesting variations [38, 41, 122]. Our work in drawing and painting (Chapter 2, 3 and 4) follow this route. Parametric learning-based methods do not, in general, scale well with the number of categories. For example, to include new media or styles, we need to train new models for the new categories and adjust system parameters accordingly. Non-parametric approaches, on the other hand, simply re-use the same general framework given new categories of exemplars, which is why we focus on these methods in this thesis.

1.5.2 Benefits

This section elaborates the benefits of non-parametric data-driven approaches for digital drawing and painting:

- The data represent reality. Exemplars can directly capture the desirable properties of natural media, including organicness, spontaneity and complexity, which can then be utilized directly for more faithful reproduction of the real world.

- The data encode designs and styles implicitly without specifying explicit technical definitions. It is easier for human to understand concrete examples than to comprehend abstract rules. Therefore, the most natural and easy way for novices to learn and start painting is to borrow from and build on examples made by professionals, instead of creating something entirely new from scratch. The use of exemplars as the painting building blocks lowers the barriers for art newcomers.

- With today's fast Internet, styles can be easily shared and reproduced by capturing exemplars and publishing them on-line. In fact, people are already doing this through on-line image warehouses, such as Flickr, and stock photography sites like istockphoto.com to share Adobe Photoshop brushes, Adobe Illustrator designs, etc. Huge opportunities lie in the effective manipulation of existing data for new applications.

- While physical simulation continues to produce compelling synthesis results, they are demanding for CPU and memory resources. It is hard to trim them down to perform well on low-power tablets and cellphone devices, whereas carefully-designed data-driven algorithms have this potential.

- Compared to simulation and procedural modeling, non-parametric data-driven algorithms tend to be parameter-free or expose small numbers of parameters, and therefore easier to understand and use.

- Data-driven techniques have the potential to combine the behaviors of different media to create appearance that is not physically possible. For example, one can combine pencil-like textural appearance with oil-like smearing behavior and watercolor-like pigment compositing effect, creating a unique digital medium on its own.

1.5.3 Challenges

Despite the these various benefits, non-parametric data-driven approaches are less explored in the digital painting and design realm. Only a handful of previous work has attempted this route with limited success, [3, 70, 93, 113, 123, 163]. This thesis aims to make further progresses towards more complete answers to the following questions:

What data to use?

For non-parametric methods, the data and the synthesis results are of the same form. A top-down approach is usually adopted. We first determine the target range of results, for example we might target thick dry oil paint applied with a fan brush using diverse hand-poses on rough surface. Then, we seek to design the data of the target medium to contain a desirable amount of variation and complexity. This step is crucial, since the quality of exemplars directly limit the range of possible synthesis appearances. Section 2.3, 3.5 and 4.4 describe the types of data that are needed to achieve the respective synthesis goals.

How much data?

Drawing and painting is an interactive process, which involves artists iteratively refining their design and composition given the continuously updated visual feedback. Therefore, interactive performance is important. The increasing popularity of battery-powered tablet devices place even harder constraints on the amount of CPU and memory that can be used for digital painting applications. The research question here is "What is the minimal amount of data sufficient to generate high-quality synthesis results?" It ties closely to the question of "What data to use?" We propose to factorize the painting process into a set of independent components and use a small amount of carefully-designed exemplars to efficiently synthesize each component, giving a modularized and general-purpose rendering framework that applies to a large variety of target media and styles (Section 1.5.4).

How to acquire and annotate the data?

The data can come from several different sources:

- Created by professionals or end users and captured by data acquisition devices, including printer, digital camera, tablet and stylus, etc (Chapter 2 HelpingHand and Chapter 3 RealBrush).
- Existing on-line data (Chapter 4 DecoBrush and Chapter 5 RealPigment).
- Completely synthetic or generated by off-line high-fidelity simulation system, for quick acquisition of large amount of data (synthetic color charts in Chapter 5).

Regardless of the sources, in most cases, the training data must be annotated by humans when it involves features that are difficult to be accurately identified by automated algorithms. We attempt to strike a balance between acquiring all necessary information and requiring manual intervention. Our data acquisition and annotation pipelines do not assume domain knowledge, special hardware or carefully controlled capture setup, for the ease of use and the potential to be adopted by novices. For RealBrush and DecoBrush, we only ask the users to specify the rough spine and end points of the strokes, which is a straightforward task and can potentially be conducted and shared collaboratively on-line by a community of users.

How to establish mapping and use data to create new variations?

The central step common to all non-parametric approaches is searching in the dataset for visually similar matches to a given query. For digital painting specifically, we chop the query into pieces at appropriate places and map the pieces to exemplar segments. Section 1.5.5 introduces our piecewise matching idea. Section 2.5, 3.6.1 and 4.5.1 discuss the adaptations to the matching algorithm for different applications. Another common challenge to address in data-driven stroke synthesis is how to morph the exemplar pieces and stitch them seamlessly to compose the query. Depending on different scenarios, this challenge might be addressed by simple warping and cross-blending (Section 2.6 and 3.6.2) or complex procedures involving graph-cut and texture synthesis (Section 3.6.3 and 4.5.4).

How to evaluate the success when there is no ground-truth?

In Non-Photorealistic Rendering, we rarely have access to agreed upon ground-truth to objectively evaluate the success of algorithms. In the case of non-parametric data-driven algorithms for digital painting where the data and the synthesis results are of the same format, we can directly compare them and ask human for their subjective opinions. We propose a user study framework based on pairwise image comparisons for perceptual evaluations of the results. We ask the viewers "Do the synthesized results look real or synthetic?" (Section 2.7.2 and 3.8) to provide some evidence for the robustness of the algorithm.

1.5.4 Painting Process Factorization

Painting is a complex process. Strokes and designs can take many different forms, shapes and textures. Stroke interactions can happen at different overlap configurations between different colors and media. Everything coupled, it seems we need a huge number of exemplars to represent a medium. And the space of all possible strokes are high dimensional and difficult to organize. We propose a factorized painting model that abstracts the physical painting process into a modularized framework composed of five independent components. Each component can be synthesized in a data-driven fashion with a small number of exemplars decoupled from the rest of the painting process. The factorization idea avoids exemplar explosion and provides a much more tractable solution space, which is crucial for low memory consumption and interactive performance. In particular, the five components are:

1. **Input trajectory.** Users interact with the painting system by providing 2D input trajectories. The quality of the trajectories characterizes an user's expertise in art and proficiency with the tablet device. Amateur users often have a hard time making smooth and confident trajectories, resulting in undesirable synthesis results. In that regard, as the first step of a synthesis pipeline, the input trajectory can be improved or stylized by trajectory exemplars provided by professional artists (Section 2.6.1). Then the synthesized trajectory can be fed into the rest of the pipeline to predict the final stroke appearance.

2. **Input hand-pose.** Casual users often paint with touch screens and mice. Their input hand-pose, such as pressure, tilt angles and rotation angle, are not well captured by those input devices. However, hand-pose are often useful for the painting applications, to drive virtual brush simulations or to procedurally determine the 2D stroke shape in order for data-driven prediction of the appearance. For example, RealBrush and DecoBrush approximate the rough outline of the query stroke taking into account the instantaneous pressure at every query sample. More complex procedures can be designed to utilize all four dimensions of the hand-pose to determine a more accurate stroke shape. Therefore, we separate the hand-pose as another system component, which can be synthesized independently by example (Section 2.6.1) and then utilized by the rest of the synthesis pipeline.

3. **Shape.** Shape refers to the outline and the textural appearance of a 2D stroke, which can also be synthesized by example. As an exemplar library, we collect a set of isolated strokes that are produced by a consistent choice of instrument and natural medium (RealBrush) or artistic style (DecoBrush). For natural media exemplars, shape inherently entails the three-way interactions between the instruments, natural media and the canvas grains. For structured vector exemplars, shape represents the geometric arrangement and composition of the design. The color of the stroke can be separated from the "shape" and can be specified by the end users at runtime. We show that both natural media and structured exemplars can be synthesized using a similar data-driven pipeline (Section 3.6 and 4.5).

4. **Inter-stroke interaction texture.** For natural media, we consider two types of inter-stroke interactions, smearing and smudging. Smearing refers to the change in foreground stroke's texture when applying it on top of background strokes of different colors. The change in texture is caused by the background pigment being picked up by the brush and mixed with the foreground pigment along the course of the new stroke. Smudging refers to the resulting texture of stroking over wet paint with a clean instrument without adding more paints. Though not explored by the thesis, structured strokes can have non-trivial interaction effects as well. For example, when strokes overlap, their structures can become interconnected and grown into an integral piece. We propose to synthesize the textural effects of the inter-stroke interaction separate from the "shape" of the single stroke, based on a set of exemplars that contain pairs of overlapping strokes (Section 3.7).

5. **Inter-stroke interaction color.** The interaction color represents the result of the optical interaction of overlapping layers of pigment applied over time. As a simple approximation, most digital painting systems use linear RGBA compositing, which behaves unlike paint. We show that the interaction color can be abstracted from the rest of the pipeline and synthesized based on physical color compositing charts, for more realistic pigment effects (Chapter 5). These

charts are often used by artists to study the pigment behaviors and therefore widely available on-line.

There are several advantages for the factored formulation: First, the decoupling of the painting process reduces the complexity and dimensionality of the data needed to capture for each stage. For example, the textural appearance of the stroke interaction is decoupled from the "shape", which eliminates the need to capture interaction exemplars for each possible "shape". Instead, we only need to capture a set of generic overlapping strokes and extract the influence of the interactions as differential operators (Section 3.5.2). The separation of the interaction color from the texture also reduces the need to capture overlapping strokes for each possible pair of colors. Suppose we would like to capture n different "shape", m interaction variations and l pairs of colors. The factorization reduces the required number of exemplars from $O(n \cdot m \cdot l)$ to $O(n + m + l)$. Second, such formulation enables modularized system design, where each component can be synthesized independently. The synthesis algorithm and the type and quantity of the exemplars for each component can be modified without influencing the design for the rest of the system. Last but not least, the factorization allows combining different behaviors of the different synthesis components for unique appearance or style that does not exist or is not physically possible. For example, one can combine one artist's hand-pose with another artist's stylish trajectory. One can combine playdoh's "shape" with oil's smearing behavior and watercolor's color compositing effect. One can also apply the data-driven color compositing algorithm to colored vector designs.

Section 3.4 gives a concrete example of the factorization idea. DiVerdi [29] discussed a slightly different factorization for natural media simulation in particular.

1.5.5 Piecewise Matching Idea

Artists interact with the digital painting tool via a sketch-based interface, where they sketch 1D curves on the virtual canvas to indicate the regions for applying paints. We call the 1D curve "trajectory" or "path" throughout the thesis. The sketch-based interface naturally leads to the use of stroke-based rendering techniques. A stroke is a parametric description of a 2D region that can be rendered in the image plane. It is the smallest discrete rendering unit that can take different shapes having wavy paths and varying thickness.

In this thesis, we use "stroke" to generally represent the information accompanying the input trajectory to be used for rendering. In particular, HelpingHand (Chapter 2) synthesizes the missing hand-pose information (pressure, tilt angles and rotation angle), given the runtime query trajectory. "Stroke" here refers to the trajectory and the hand-pose data. In RealBrush (Chapter 3), we directly synthesize the 2D appearance of the query (stroke), based on natural media strokes. In DecoBrush (Chapter 4), we refer to the whole structured design as "stroke" and synthesize it along the direction of the input trajectory. The common problem to be solved for all three systems is the mapping between low-dimensional input (trajectory) to high-dimensional exemplar (hand-pose or 2D stroke appearance).

Notice that, given an arbitrary query trajectory, it is unlikely that a single exemplar will be a good match. The exemplar might need to be stretched or squashed depending on the length of the query. Previous work have chopped the exemplars into small units and match the query trajectory to sequence of those units, [3, 70, 93, 163]. The exemplar units are often too small limiting the synthesis quality and the range of styles they can support. Large features in the exemplar strokes might be broken up and hard to recreate during synthesis.

In contrast, we propose a piecewise matching idea that automatically breaks the query trajectory into long segments each of which is matched to a segment of an exemplar stroke. In particular, we represent the input trajectory by a sequence of discrete samples, where each sample is composed of the x, y coordinates in the canvas plane. To perform matching, we construct a feature vector at each sample that represents the local trajectory variation and the relative positioning within the stroke. For each query sample, we perform approximate nearest neighbor search and obtain a set of candidate exemplar samples. Then we use global optimization, taking into account the entire query stroke, to select one exemplar sample for each query sample, favoring connecting consecutive exemplar samples into long segments. Our design of feature vectors and energy functions for optimization strives to preserve the beginning and end characteristics of the exemplars in the synthesized stroke, to select as long as possible exemplar segments of similar trajectory to the query, and to avoid sharp discontinuity in the synthesized appearance. The piecewise matching idea is utilized by HelpingHand, RealBrush and DecoBrush, each of which adapts the design of feature vector and energy functions to achieve a slightly different goal (Section 2.5, 3.6.1 and 4.5.1). The trajectory-based matching is governed by our observation that similar trajectories leads to similar hand-poses, which in turn produces similar stroke appearance (Section 2.3.1).

1.6 Our Contributions

This thesis makes the following contributions.

- We propose factorizing the painting process into a set of orthogonal components: 1) the input trajectory; 2) the input hand-pose; 3) the stroke shape; 4) the inter-stroke interaction texture; 5) the inter-stroke interaction color. We develop prototype systems to demonstrate that each of the components can be synthesized efficiently and independently based on small sets of decoupled exemplars. The factorization idea makes data acquisition and stroke synthesis tractable and efficient, meeting the need of interactive performance.

- We introduce efficient techniques to acquire and process several types of visual exemplars with the goal in mind of ease of acquisition. We perform analysis on the exemplars to provide useful insights for the artistic rendering community. We also make our exemplar databases publicly available to encourage future research in this direction.

- We describe a novel and versatile piecewise matching algorithm that matches a query path to long segments of exemplar strokes using feature-based nearest neighbor search and solving an optimization efficiently with dynamic programming. We show that different types of exemplars can be synthesized using the same framework.

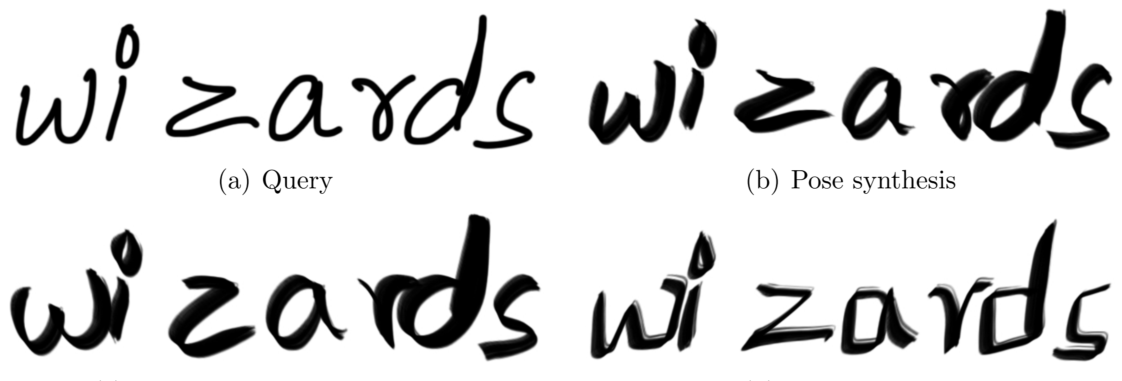
- We design our prototype systems with the ease of use in mind. We provide intuitive user interfaces for novices, where one can simply select a target exemplar library and make simple sketch to create complex-looking strokes and paintings. We enable novices to easily create visually-compelling drawings, paintings and designs in the hands of expert.

- We propose a user study framework that evaluates the perceptual similarity of the synthesis results to the real exemplars. Our results show that casual viewers have a hard time distinguishing our synthesis results from the real exemplars, especially compared to the results of naive synthesis methods.

Chapter 2

HelpingHand: Example-based Stroke Stylization

Digital painters commonly use a tablet and stylus to drive software like Adobe Photoshop. A high quality stylus with 6 degrees of freedom (DOFs: 2D position, pressure, 2D tilt, and 1D rotation) coupled to a virtual brush simulation engine allows skilled users to produce expressive strokes in their own style. However, such devices are difficult for novices to control, and many people draw with less expensive (lower DOF) input devices. This chapter presents a data-driven approach for synthesizing the 6D hand gesture data for users of low-quality input devices. Offline, we collect a library of strokes with 6D data created by trained artists. Online, given a query stroke as a series of 2D positions, we synthesize the 4D hand pose data at each sample based on samples from the library that locally match the query. This framework optionally can also modify the stroke trajectory to match characteristic shapes in the style of the library. Our algorithm outputs a 6D trajectory that can be fed into any virtual brush stroke engine to make expressive strokes for novices or users of limited hardware.



(a) Query

(b) Pose synthesis

(c) Trajectory & pose synthesis

(d) Alternate style

Figure 2.1: Hand-writing synthesis results. (a) Given one or more query strokes, we synthesize (b) hand pose data crucial for bush simulation, based on artist's examples. (c) We can also modify the trajectory to locally match the artist's style. (d) Trajectory and pose synthesis using different artist's style.

2.1 Background

Artists wielding traditional drawing instruments such as a pencil, a stick of charcoal or a paint brush exercise fine control over the interplay between their gestures and the physical medium. The appearance of the strokes is governed not only by the path of the instrument but also the pressure and angle (see inset). Digital artists often work with high quality tablets that track a stylus with 6 DOFs (2D position, pressure, 2D tilt, and rotation), which painting software can use to simulate the interaction of digital brushes in various media with increasing fidelity [10].

However, most people drawing on digital devices today do not use 6-DOF tablets, for two reasons. First, high-quality tablets are relatively expensive as compared with other input devices like mice (2-DOF), multitouch screens ($\geq$2-DOF), or mass-market tablets (3- or 5-DOF). Second, even if everyone owned a 6-DOF tablet, most people would not have the training and experience to be able to wield it effectively. To achieve high quality calligraphy, for example, takes many hours (perhaps years) of practice.

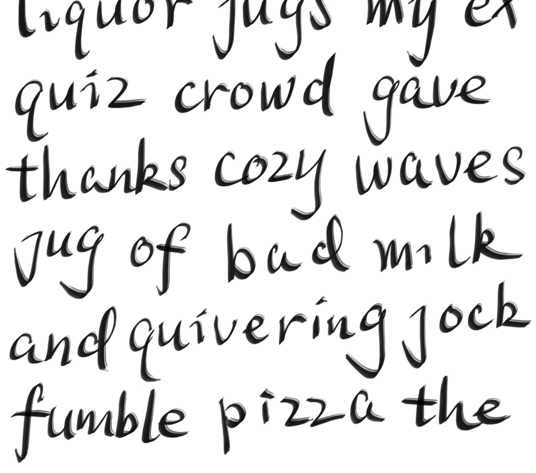
© Sandr Imaging

This chapter describes a method whereby a non-expert draws a 2D query stroke (Figure 2.1a), perhaps using an inexpensive input device, and missing DOFs are synthesized based on a library of examples supplied by an artist (Figure 2.2a). The resulting marks follow the trajectory drawn by the non-expert but convey the gestural style of the artist (Figure 2.1b). We also show how the approach can be extended to *approximate* the input trajectory while adopting some of the character of the strokes in the artist's library (Figure 2.1c). This option is beneficial for users who are not confident in their own style, or when using a coarse input device like a mouse. One can easily change style by choosing a different library (Figure 2.1d).

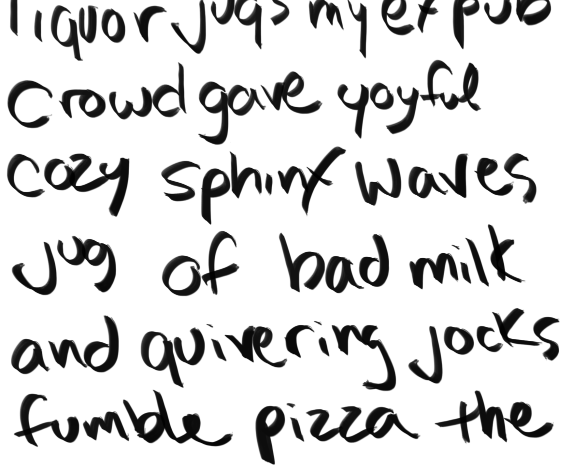
To synthesize hand pose, we address this problem: given a library of 6-DOF strokes (drawn by the artist) and a 2-DOF query stroke (drawn by the user), produce 4-DOFs to accompany the query so that the 6-DOF (2+4) combination *looks like* the strokes in the library. Trajectory stylization optionally replaces the 2-DOF trajectory as well. There is no single correct answer because the same path can be drawn differently, even by a single artist, so our goal is to synthesize *plausible* gestures.

Our method produces gestures that are plausibly in the style of the artist's library. It performs at interactive rates to provide immediate feedback to the user during drawing and to make it easy to choose amongst different styles. Finally, while we have designed the system for interactive use by non-experts or people without high-quality tablets, the same method can apply stylization to any source of line art, including the output of vector illustration software or computer-generated line drawings based on 3D models.

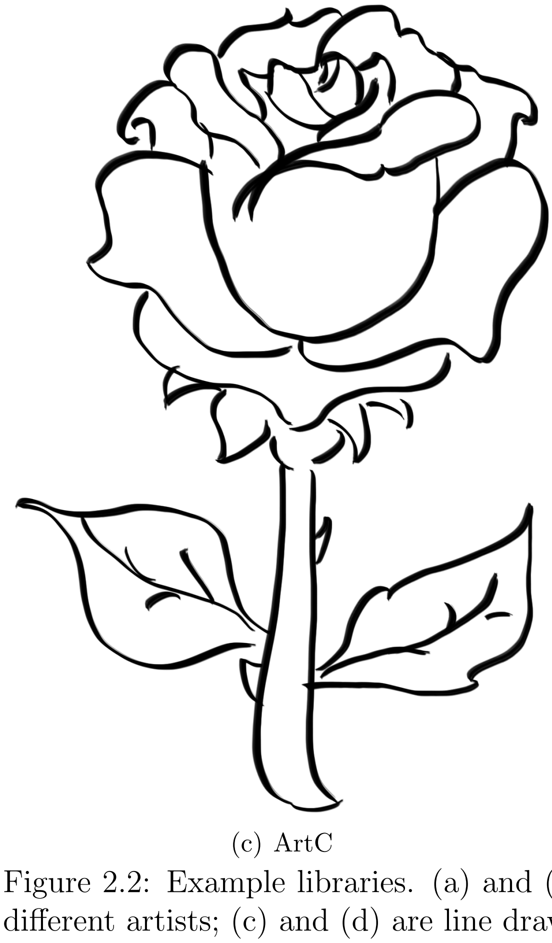
Our main contributions include stroke stylization via example-based synthesis of hand pose—data needed to control a virtual brush—as well as optional trajectory modification. Through data collection and analysis we offer understanding of how hand pose relates to individual writing style. Finally, we explore the space of candidate algorithms and evaluate them quantitatively and qualitatively, concluding with a demonstration of practical results.



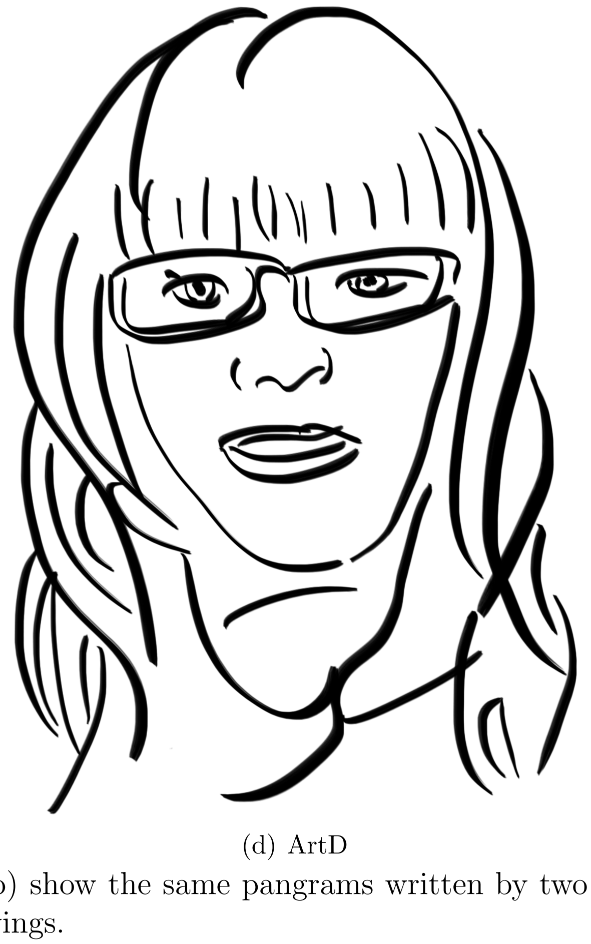
(a) ArtA



(b) ArtB



(c) ArtC



(d) ArtD

Figure 2.2: Example libraries. (a) and (b) show the same pangrams written by two different artists; (c) and (d) are line drawings.

2.2 Related Work

Our fundamental idea builds on recent advances in **non-parametric modeling** that have been shown to be effective for texture synthesis by example in a variety of domains—images, meshes, video, and animation [143]. These techniques work by selectively copying portions of a library to generate new data that maintains a similar statistical distribution and aesthetic property, with careful tailoring to the particular requirements of each domain. Ashikhmin [4] observes that copying swaths of pixels for image texture greatly improves performance over per-pixel methods, but leads to discontinuity artifacts at boundaries. Optimal boundary placement on the 2-D grid is intractable; however, we show that on our 1-D domain it can be solved via dynamic programming. Due to this optimization over the 1-D domain, our method is similar to the video textures of Schödl et al. [122].

We also derive high-level inspiration from the approach for human motion synthesis of Kovar et al. [74] and Hsu et al. [55], both of which synthesize high dimensional output from a database of low dimensional control input. Several aspects of stroke gesture synthesis make our problem unique. For example, artist data collection must be fully automatic, as many strokes are made quickly. Similarly, user input must be processed at interactive rates and without additional user input, to allow for a fluid painting experience. Furthermore, we separate the concerns for pose stylization versus input trajectory stylization, and show how to apply our algorithm to either one individually or both simultaneously.

Stylization. There have been a number of versions of curve and stroke stylization in the graphics community. In the 2D domain, Elasticurves [132] intelligently neaten input strokes based on velocity data, but are only able to produce a single style of smooth result. McCrae and Singh [96] present a different neatening technique that would piecewise fit and blend among french curve segments to sketched input, to create pleasing line shapes. They also demonstrate alternative stylizations by substituting different canonical strokes for french curves. The curve analogies framework of Hertzmann et al. [51] could apply arbitrary stylizations to input curves, by learning variations from pre-existing input-output curve pairs. But its extension to more general brush parameters such as pose is not obvious. Moreover, our approach for trajectory modification relies on a library of output-only exemplars, rather than input-output pairs.

Some previous work address additional brush parameters. House and Singh [54] propose a dynamic control process to generate varying width along smooth curves. Saito et al. [116] describe heuristics to procedurally compute stroke width based on curvature. Both approaches depend on explicitly and narrowly defined relationships between stroke shape and width, and do not support additional pose DOFs as needed for a virtual brush model. The isophote distance of Goodwin et al. [45] formulates heuristics about line thickness, but relies on a 3D scene as input.

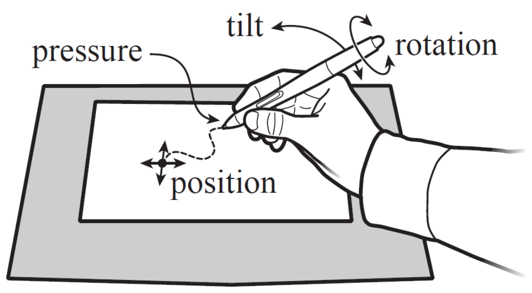
Specifically targeted towards digital painting, Okabe et al. [104] apply realistic brush strokes to curves, by first acquiring videos of the changing shape of real brushes during strokes, and then training an HMM to generate the appropriate footprint based on the user's curves. The IntuPaint system of Vandoren et al. [138] creates artistic paintings using instrumented, physical brushes and workspaces and interactively acquiring the pigment deposition for compositing to a virtual document in realtime. It supports a variety of traditional media tools and techniques, but imposes heavy custom hardware requirements and thus is not suitable towards our goal of more generally applicable brush stroke rendering.

Handwriting. Our approach of learning and then reproducing the brush pose styles of real artists also bears similarities to a branch of biometrics research concerned with handwriting verification and synthesis. Much of the work records samples of individual character glyphs in advance, and then plays back those glyphs based on the desired target text, often with random or physically inspired perturbations [139]. However, these techniques only reproduce strokes they have recorded and labeled, making them inappropriate for freeform sketching.

More general approaches attempt to create dynamical systems that can reproduce 2D handwriting-like motion from training samples [52]. In the extreme, Plamondon [109] developed a kinematic theory of rapid human movements that was used to synthesize stroke trajectories with properties similar to natural motions. Beyond 2D trajectories, Franke et al. [40] create a handwriting robot that holds a pen at a static orientation but can vary pressure throughout a stroke, to create very convincing signature forgeries. Yu et al. [159] consider hardware that acquire 5D input including pressure and 2D tilt, but only for the purposes of writer identification, and not synthesis. We are not aware of previous work that considers the six DOFs of a real brush.

2.3 Data Collection and Analysis

We collected two data sets from the artists: English pangrams and line drawings (Figure 2.2). Five artists drew strokes in their own natural drawing style using a Wacom Intuos4 tablet with an art pen (inset) and Adobe Photoshop CS5 for immediate visual feedback and recording, sampled at 140 Hz. During recording, the tablet orientation remained consistent for each particular artist and is treated as part of their drawing style. Each 6D sample contains 2D position $\mathbf{x} = \langle x, y \rangle$ and 4D pose $\mathbf{h} = \langle \rho, \theta, \phi, \omega \rangle$ comprised of pressure ρ, tilt $\langle \theta, \phi \rangle$ and rotation ω. We refer to *trajectory* $\mathbf{T}$ as a sequence of sample positions $\mathbf{T} = \{\mathbf{x}_i\}$, and *pose* $\mathbf{H}$ as the corresponding hand pose sequence $\mathbf{H} = \{\mathbf{h}_i\}$. After capture, we resample each stroke uniformly in arc length (roughly 1 pixel spacing in 2000x2000 images).



Our system takes those recordings as input and builds a collection of 6D strokes that we call library $\mathbb{L}$. Each library stroke $\mathbf{L} \in \mathbb{L}$ consists of a set of position and pose samples, $\mathbf{L} = \{\mathbf{x}_i, \mathbf{h}_i\}$. Figure 2.2 shows Photoshop renderings from some examples of the data that we collected. While artists created multiple pages of input (e.g., Figure 2.1a), we treat each page as a separate library. Typical pangram libraries have between 100 and 200 strokes, each of which typically has between 40 and 80 samples, whereas drawings often contain some longer strokes.

2.3.1 Artist Distinctiveness

This project relies on several assumptions. The first is that shapes of strokes used in handwriting (for example) are part of the "style" of a given artist; different examples of similar shapes like loops or 'v' shapes will be more consistent from the same person as compared with examples from other artists. This assumption is widely accepted. For example, it forms the basis for forensic handwriting analysis. We rely on this assumption for the trajectory synthesis method presented in Section 2.5. But we also make a second assumption that is less well studied—that artists' hand poses are determined by their target trajectories, with some variance. An artist drawing the same shapes (with the same intention) multiple times tends to select poses from the same pool of possible gestures. This implies that given a stroke $\mathbf{L}$ with trajectory $\mathbf{T}$, if we find other strokes with similar trajectories their poses should have similar statistics as $\mathbf{T}$. This observation forms the underpinnings of the example-based pose synthesis methods presented in Section 2.5.

Therefore, to verify our pose assumption, we conduct a correlation-based analysis on the recorded library data, investigating how well the pose attributes are correlated between stroke patches with similar shapes. For each sample i in each stroke $\mathbf{L} \in \mathbb{L}$, we construct a patch $\mathbf{t}_i$ centered at sample i and composed of neighboring samples $\mathbf{t}_i = \{\mathbf{x}_j \mid n \geq |i-j|\}$. For a stroke with m samples, we have a total of $m - 2n$ overlapping patches, each of size $2n+1$. The size n provides a notion of what we mean by "local" and through moderate trial and error we found $n = 12$ to work well; we use this value throughout our experiments.

For each patch $\mathbf{t}_i \in \mathbf{L}$ we construct a simple feature vector $\mathbf{F}_i$ as the vector of local tangents at every sample, concatenated with a pair of coordinates $\langle x_i^-, x_i^+ \rangle$ that indicates whether the sample is near the beginning or end of the stroke (described fully in Section 2.5.1). Next for each patch $\mathbf{t}_i$ we find the most similar patch $\mathbf{t}_c$ from a different stroke within the library, measured by the L_2 distance $\|\mathbf{F}_i - \mathbf{F}_c\|$. We reject all pairs of patches whose distance is larger than a threshold (the mean feature distance over all pairs of patches). For each remaining pair of patches $\langle \mathbf{t}_i, \mathbf{t}_c \rangle$ we calculate the Pearson correlation coefficient between their corresponding pose attributes, $r_{i,c} = \mathrm{corr}(\mathbf{H}_i, \mathbf{H}_c)$. To aggregate across multiple patch pairs and multiple pose attributes, we convert Pearson's r coefficients into Fisher's z coefficients, compute the average, and then convert the result back into Pearson's r [126]. The aggregated r gives us an indication of how well a library's pose is correlated among similar strokes. As a baseline comparison, we also find *random* pairs of patches of the same size $2n+1$ within the library and calculate the correlation among them.

Figure 2.3 summarizes our findings. The bar chart shows that within libraries written by the same person, pose sequences of random pairs of patches are not well correlated whereas pose from strokes with similar trajectories have high correlation. The blue bars report aggregate correlation within the library shown in Figure 2.2b.

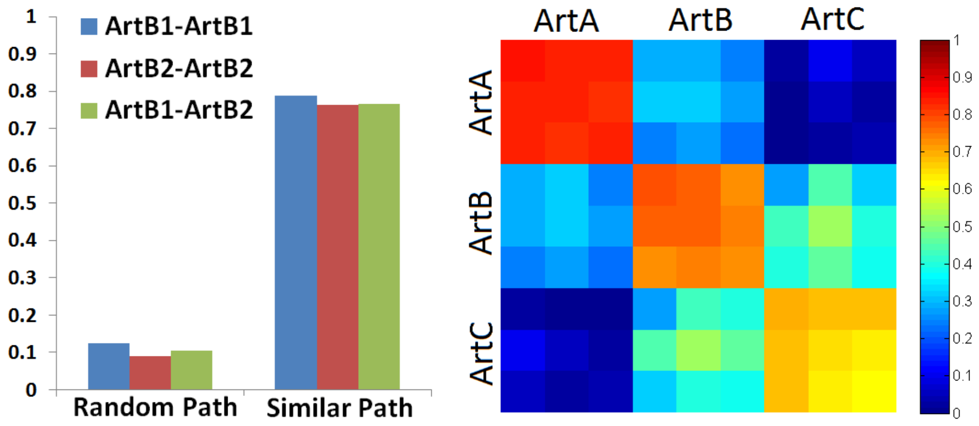


Figure 2.3: Hand-pose correlation. *Left:* Correlation is weak for random pairs and strong for similar shapes. *Right:* Libraries drawn by the same artist are highly correlated (diagonal blocks), relative to libraries from different artists.

Red bars are for a different library from the same artist, and green shows correlation between those two libraries. While correlation is low for random pairs it is still positive; many strokes have *some* correlation because pressure always starts and ends near zero, for example. As evidence that different people have different writing styles, we observe that their hand pose data also carry different statistics. The matrix on the right shows the same correlation analysis described above, comparing all pairs from a set of nine libraries (three by each of three artists). The block diagonal (same artist) exhibits obviously higher values than the off-diagonal (different artists).

Of course even within the same library, correlation is not perfect ($r < 1$) for at least three reasons. (1) During the recording process, which lasts about an hour, the volunteer may introduce hand pose variation setting down the stylus and picking up again with a different grip. (2) Pairs of similar patches do not have exactly the same trajectories. (3) The same artist exhibits some variation in hand pose, even when following the same path multiple times. The degree of variance depends on the individual and to some extent the artist's skill. Pose data from novice users tends to exhibit less consistency. Three of our five volunteers have more than two years of experience with a stylus (Artists A-C in Figure 2.3) while the other two are less experienced and have correlation statistics around 0.5—still higher than typical correlation between different artists (off-diagonal).

Figure 2.4a supports these claims with a visualization of the pressure magnitude of example characters drawn by an artist (the word "quiz" above with other examples of letters 'q' and 'u' below). Observe that the same shapes drawn at different moments on different parts of the tablet exhibit similar pressure progressions, although there is variation.

2.3.2 Spatial Variation

There are other influences related to pose, beyond shape and artist. One of the prominent factors is the position of the stroke on the tablet. This is to be expected

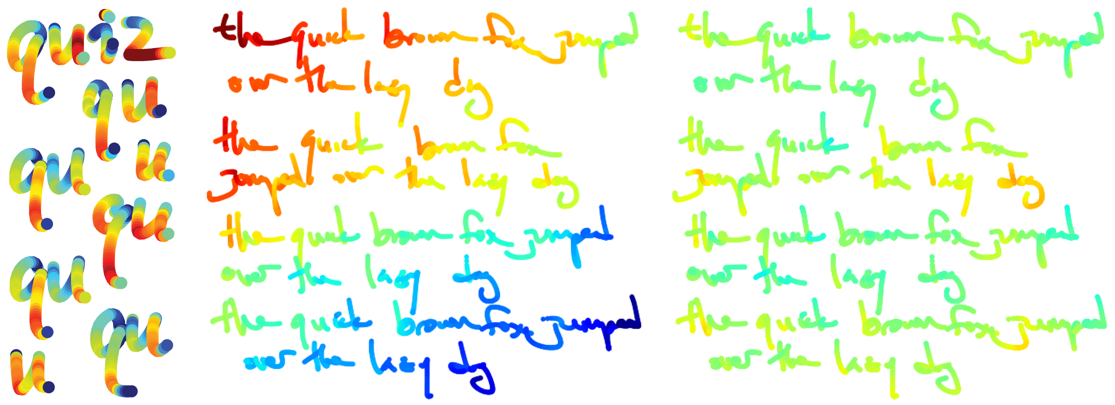

(a) ρ (b) rotation ω_i (c) rot: ω_i - $\Omega(\mathbf{x_i})$

Figure 2.4: Hand-pose visualizations (color range: blue=low – red=high). (a) Pen pressure follows similar progressions for similar shapes drawn at different times. (b) Rotation angle depends strongly on pen position. (c) Positional influence removed via linear regression.

based on body kinematics, because the pen pose changes when the person moves his hand from one position to another. This factor can have an observable effect. As an extreme example, Figure 2.4b shows that the rotation of the pose is strongly related to the position of the stroke, at least in this library where the range from blue to red is roughly 50°. It is possible to remove the strong spatial component prior to pose synthesis, as follows. We model the pose attributes at sample i, for example rotation ω_i, as a sum of two components: (1) local variation intended by the artist ω'_i, and (2) a positional component $\Omega(\mathbf{x_i})$. We observe that the the local variations are fairly uniformly distributed and may be viewed as noise relative to the more global positional signal. Therefore we approximate $\Omega(\mathbf{x_i})$ by a quadratic polynomial over the coordinates of $\mathbf{x_i}$, and perform a simple least squares fit to find the coefficients of the polynomial. Then we can remove the approximate positional influence simply by subtracting the polynomial from ω_i, but adding in its place the average value. Figure 2.4c shows the result, with a range of roughly 20°. We find that tilt tends to vary by position the way rotation does, but pressure is more consistent. Positional influence varies from library to library. Nevertheless, we have not found these effects to have a strong impact on the visual quality of the synthesis results. Therefore, our libraries have not been modified using the regression fit for any of the results that follow.

2.3.3 General Observations

Here we note a few general observations we found when studying the data in the example libraries supplied by our artists:

- **Continuity.** Pose tends to change smoothly along strokes.
- **Directionality.** Strokes drawn from left to right are not identical to strokes drawn right to left, for example.

Preprocess:

- Compute feature vectors for library samples: (§2.5.1)
  - choice
    - ∘ shape contexts
    - ∘ filtered velocities

Online:

- Compute feature vectors for queries, as above (§2.5.1)
- Find k nearest neighbors (k-NN) of each sample
- Select neighbors from k-NN for synthesis: (§2.5.2)
  - choice
    - ∘ closest neighbor
    - ∘ weighted average
    - ∘ optimal sequence
- Post process: (§2.6.1)
  - ∘ for optimal sequence: transition blending
  - ∘ for trajectory: shape optimization

Figure 2.5: Algorithm overview.

- **Gravity.** Statistics of stroke paths are not rotationally symmetric; up, down, left and right are not interchangeable.
- **Endpoints.** Special care must be taken for the beginnings and endings of strokes because the physical act of drawing makes these parts look different from the middle of the path.

2.4 Algorithm Overview

Data analysis in Section 2.3.1 tells us strokes with similar trajectory are drawn with similar hand pose. Given a query stroke trajectory, we can "hallucinate" plausible hand pose from segments of library strokes that have locally similar trajectory. The synthesis algorithm operates on a stroke by stroke basis, processing each stroke in isolation, so inter-stroke effects are not considered. In an interactive painting program, on mouse-up after a stroke is drawn, the synthesis algorithm is run and the new stroke replaces what the user drew (in a fraction of a second for typical strokes).

In both handwriting and line drawings, the hand pose attributes themselves do not fully represent an artist's style. We observe that one of the most important cues is the trajectory itself. Given a query trajectory that might be drawn with the shaky hand of an amateur user, we can find a new trajectory that follows the user's overall intention but has the style and expertise of the artist who drew the library. This application relies on the same algorithm developed for pose hallucination. We replace the query trajectory with similar, but not identical, patches from the library (striking a balance between following query intent and preserving library style).

As pre-processing, we calculate a feature vector for each sample of each exemplar stroke (Section 2.5.1). At runtime, for each query stroke, we first compute a feature vector for each sample and then optimally select appropriate exemplar samples for the query stroke (Section 2.5.2). We use "piecewise matching" to refer to the forementioned algorithm that performs feature matching and optimization to map the query stroke to consecutive segments of the exemplar strokes (Section 2.5). Chapter 3 and 4 reused and modified this algorithm respectively as the first step of their pipelines. After the mapping is established, we need to perform additional synthesis step to "hallucinate" the pose and trajectory attributes for the query stroke (Section 2.6).

For each stage of the synthesis pipeline (Section 2.5.1, 2.5.2 and 2.6.1), we consider different implementations to support various data and performance trade-offs. Figure 2.5 offers a concise overview of these stages and implementation options.

2.5 Piecewise Matching

Section 2.5.1 describes two different designs of feature vector, *shape contexts* and *filtered velocities*. Section 2.5.2 discusses three different methods, *closest neighbor*, *weighted average* and *optimal sequence*, for selecting appropriate exemplar samples as matches.

2.5.1 Feature Vectors

For the exemplar sample or the query sample, we compute a feature vector that describes the local shape, using either *shape contexts* (Section 2.5.1) or *filtered velocities* (Section 2.5.1). Then online, for a given query stroke, synthesis begins by computing feature vectors for the query samples, and then searching for similar features in the library.

One goal of the feature vector is to characterize the local trajectory of the query (or library) strokes. Therefore, our feature vector takes into account samples in the "recent history" and "near future" as in the patch construction from Section 2.3.1. The pose attributes at the start or end of a stroke have different distributions than those in the middle (Section 2.3.3). Therefore, a second goal for the feature vector is to encode closeness to the endpoints, where appropriate. We define the scalar $x_i^- = \min(1, \sqrt{d/d_h})$ where d is the arc length distance to the start of the stroke and d_h is the size of our recent history window. Likewise we have x_i^+ measuring distance to the end of the stroke relative to the future window. These values are clamped such that the interiors of the strokes are undifferentiated. Putting this all together we have feature vector $\mathbf{F}_i = \{x_i^-, x_i^+, W\{\mathbf{f}_j\} \mid j\}$, where $\{\mathbf{f}_j\}$ is the set of shape features and W is a weight that balances between the relative importance of shape features versus end features. The next two subsections consider alternatives for the shape features.

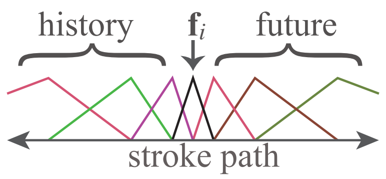
Shape Contexts

Introduced by Belongie et al. [15], a *shape context* is a log-polar histogram of sample positions on the curve, relative to the current sample. We have adapted a variant that performs well for our application, with two modifications. First, we consider *local* shape by using only a small number of history and future samples. Second, we further divide these into two separate histograms, one for history and one for future; this division allows us to distinguish, for example, a stroke drawn left to right versus right to left.

In our experiments we have used from 25 to 40 history and future samples each, depending on the data set. The shape features $\mathbf{f}_j$ are then the histogram bins, of which we use 6 angular bins and 5 radial bins for a total of $2 \times 6 \times 5 + 2 = 62$ dimensions for $\mathbf{F}_i$.

Filtered Velocities

Here we describe a feature vector that is based on velocity rather than shape. We use the sample velocities $\{\mathbf{v}_i\}$, where $\mathbf{v}_i = \mathbf{x}_i - \mathbf{x}_{i-1}, \mathbf{v}_0 = \mathbf{0}$. A naïve implementation of the feature vector would have each tap correspond directly to a velocity sample from the stroke: $\mathbf{f}_j = \mathbf{v}_j$. Because of the high sampling rate, this approach would require many taps to capture a significant portion of the trajectory, and would apply equal weight to samples regardless of how near or far they are to the current sample.



Instead, each filter tap is obtained by applying a discrete triangle filter (inset figure on the right) to the velocity samples, $\mathbf{f}_j = \sum w_k \mathbf{v}_k$, where w_k are the triangle filter weights, for $(i - N) \leq j \leq (i + M)$ taps. N and M are the number of history and future taps, respectively. The triangle filters overlap so that the start index and the end index of a triangle are the center indices of the adjacent triangles, which ensures that all the weights applied to a single velocity sample in a feature vector sum to one. The filter width increases exponentially with distance from sample i, which means individual samples that are distant contribute relatively less to the feature vector. When synthesizing the query samples near the start or the end of the query stroke, some of the feature taps will cover hypothetical samples that are outside of the query stroke. In that case, we only average the velocity of the samples that are valid. If all the samples under a triangle centered at j are located outside of the valid range, then we use $\mathbf{f}_j = 0$. Using triangle filters reduces the dimensionality of the feature vector needed to represent a large local neighborhood and filters out noise present in the raw samples.

Results in this chapter use triangle filters with equal sized history and future windows $M = N = 7$. Thus, considering 2D velocity and two extra "endpoint" dimensions, the overall dimension of the feature vector $\mathbf{F}_i$ is $2 \times (2 \times 7 + 1) + 2 = 32$. We also use a weight $W = 0.3$ balancing shape features against endpoint features. In our experiments we have found the method to be relatively insensitive to these parameters, and robust through a broad range. Section 2.8 discusses the relative merits of the two feature vectors we investigated.

2.5.2 Selecting Among Nearest Neighbors

Once we have computed a feature vector for each sample in a query stroke, we find its k nearest samples in the library as potential samples for synthesis. Brute force search is not practical because the library is potentially large and the search must be performed for many samples. Acceleration structures like K-D trees are known to lose efficiency in high dimensional searches, as in the case of our feature vectors. Therefore, we resort to algorithms for finding *approximate* k nearest neighbors, the output of which we have found acceptable in the stages that follow. We experimented with two algorithms, the FLANN implementation of Muja and Lowe [101], and an adaptation of the PatchMatch algorithm of Barnes et al. [8]. The PatchMatch approach requires a straightforward modification to search over the 1-D domain of samples, and we found that with our data the algorithm converges to a reasonable approximation in ~5 passes (as Barnes et al. found for image patches). Overall, the two approaches offer roughly equivalent performance for equivalent quality of output. The major tradeoff between the methods is that FLANN expends time and memory to build auxiliary structures during the offline library construction phase; on the other hand PatchMatch is only efficient when processing the entire query stroke at once and therefore is not suitable for on-the-fly synthesis during stroke input.

Once we have the k neighbors (k-NN) for each query sample i, the next step is to synthesize a new sample $\{\mathbf{x}'_i, \mathbf{h}'_i\}$ from them. The synthesized attributes should look like those of similar shapes in the library, especially at the start and end of strokes, and vary smoothly in time. We consider three approaches—*closest neighbor, weighted average,* and *optimal sequence*—discussed in the following subsections.

Closest Neighbor

The most straightforward approach is to select the one neighbor with the smallest distance. That is, just use the closest neighbor to the query sample. While simple, this method does not lead to good coherence properties along the length of the stroke. Suppose library sample $\mathbf{L}_j$ is the 1-NN of query sample $\mathbf{Q}_i$. For strong coherence we want $\mathbf{L}_{j+1}$ to be the 1-NN of $\mathbf{Q}_{i+1}$, which is only sometimes true (and it becomes more rare as the library size grows). However, $\mathbf{L}_{j+1}$ is *usually* among the k-NN, which means there is an opportunity to do better.

Nevertheless, we retain this case for comparison with the other methods in Section 2.8. Moreover, this case may be thought of as a stand-in for the general non-parametric texture synthesis approaches following that of Efros and Leung [39]. In particular, we also experimented with choosing randomly among the k-NN, and also the variant proposed by Ashikhmin [4] (which probabilistically chooses $\mathbf{L}_{j+1}$ in the scenario above). However, in our experiments, none of these led to results as coherent as those of the methods that follow.

Weighted Average

Building on the intuition of the closest neighbor approach, interpolation is a way to provide smooth transitions. Given k neighbors $\mathbf{n}_j$, with distances d_j from the

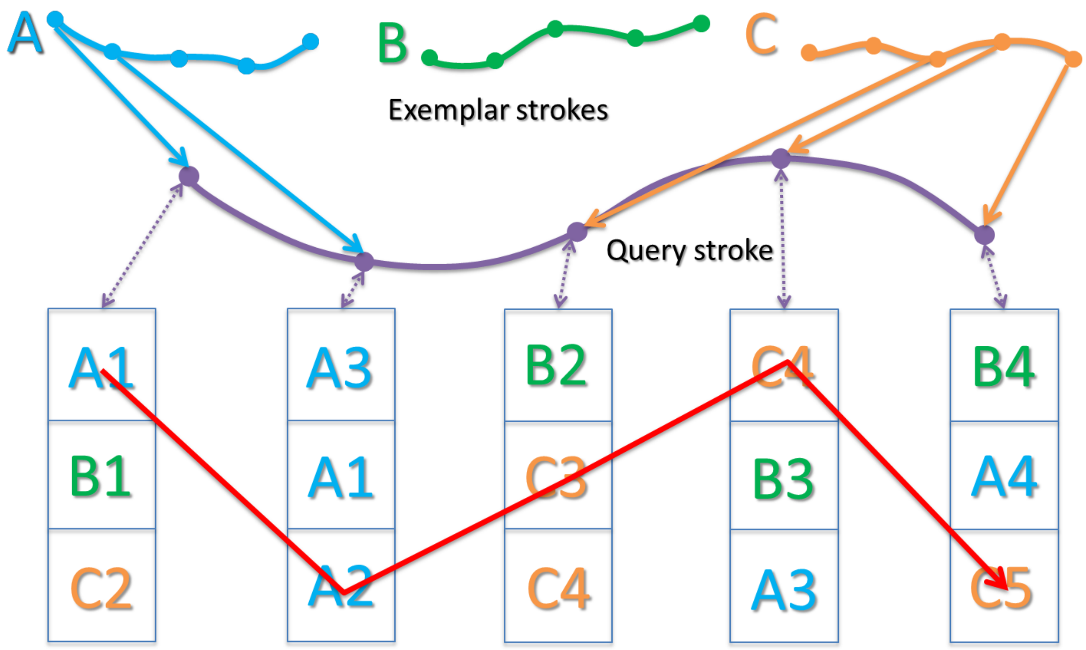


Figure 2.6: Piecewise matching illustration.

query sample in feature space, we synthesize a sample as their weighted average using weights $1/d_j$. This produces consecutive samples that vary smoothly, creating a continuous output stroke. However, the library typically contains many similar strokes (say multiple versions of the letter 'a'), each with slightly different pose and trajectory. The weighted average method therefore generates a stroke that is a compromise among the distinctive features of the library strokes, not quite reproducing any of them well. The result is that this method produces strokes that tend to be smoother than the artist's style.

Optimal Sequence

Both closest neighbor and weighted average are local solutions that have difficulty reproducing the smoothness and distinctiveness of the artists' style. To achieve both of these qualities, we propose a (per-stroke) global optimizing solution, which we solve using dynamic programming for efficient computation. One goal of the optimization is to select a few long *segments* of library strokes (sequences of near-consecutive samples) to be matched to segments of the query (Figure 2.7), avoiding many potential discontinuities. Figure 2.6 illustrates the general idea. Consider three exemplar strokes (blue, green and orange) and the query stroke of five samples (purple). In this example, each query sample finds $k = 3$ nearest samples from the exemplars. The goal is to optimize the transition path (red) that selects one nearest exemplar sample for each query sample.

We optimize for the sequence of transition indices $\{\mathbf{c}_i = \langle a_i, b_i\rangle \mid i = 1, 2, \cdots, s\}$ by minimizing a total synthesis cost over the s samples in the query stroke. For the i-th query sample: a_i is the index of the selected library stroke, and b_i is the index of

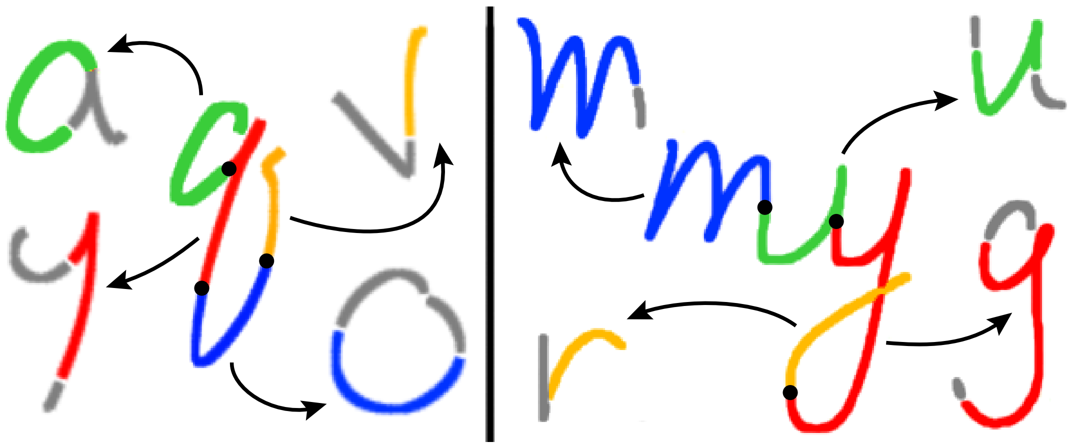


Figure 2.7: Optimal matching visualization. Query strokes 'q' and 'my' are broken into parts that map to segments of library strokes.

the selected sample on stroke a_i. We seek the optimal sequence $\{\mathbf{C}_i\}$ that minimizes the sum of four error terms, e_f, e_s, e_m and e_t. These terms address, respectively, matching feature vectors, avoiding short segments, matching stroke endpoints and choosing good transitions between segments—discussed below.

Feature Penalty. The term e_f simply sums (over all query samples i) the distance in feature space to the selected neighbor $\mathbf{c}_i$. If all other terms in the optimization had no impact then this term would drive the solution to the closest neighbor selection method. However there is typically a different neighbor among the k-NN that is *almost* as good as the closest neighbor and that has other desirable properties that would cause it to be selected over the closest neighbor, based on the terms that follow.

Short Penalty. We find that short segments often lead to visual discontinuities in the synthesized result. To avoid them we impose a penalty e_s taken as the sum of $(C_l + (L_{\min} - l_i))^2$ over all segments of length $l_i < L_{\min}$. In our experiments, we use $C_l = 3$ and $L_{\min} = 12$ samples.

Endpoint Penalty. As noted in Section 2.3.3, ends of strokes have unique characteristics. Therefore we prefer where possible to synthesize the beginning of a query stroke with the beginning of of a library stroke, and the same for ends. We impose a penalty term $e_m = C_e((b_1 - 1) + (s' - b_s))$ where b_1 is index of the neighbor selected for the first sample, b_s is the index for the last query, and s' is the number of samples in the library stroke a_s. This term essentially measures how far the neighbors selected for the ends of the query are from the ends of their respective strokes. $\mathbf{C}_e$ is the cost of not matching endpoints, which we set to $C_e = 3$.

Transition Penalty. The term e_t encourages using fewer segments. For consecutive query samples, we encourage selecting samples that are consecutive on a library stroke, since they have natural coherence in the hand pose attributes and capture the local style. If that is not possible for other reasons, we still encourage repeating the same library sample once, or skipping exactly one library sample—policies that allow the library segment to be stretched or shrunk to match the query. Otherwise we call the transition a *jump*, and assign a penalty that includes both a large constant cost and a cost based on pose discontinuity at the jump location. We compute the transition penalty e_t by summing over the query e_t^i, the transition cost of going from $\mathbf{c}_i$ to $\mathbf{c}_{i+1}$. According to the policies above there are four cases:

1. *Consecutive:* If $a_i = a_{i+1}$ and $b_{i+1} = 1 + b_i$, then $e_t^i = 0$.
2. *One repeat:* If $a_i = a_{i+1}$ and $b_{i+1} = b_i$ and $b_i \neq b_{i-1}$, then $e_t^i = C_r$, the cost of stretching.
3. *One skip:* If $a_i = a_{i+1}$ and $b_{i+1} = 2 + b_i$, then $e_t^i = C_s$, the cost of shrinking.
4. *Jump:* Otherwise, $e_t^i = C_t + C_p \left\| \mathbf{h}_{c_i} - \mathbf{h}_{c_{i+1}} \right\|$, the cost of a jump with deeper penalty for large pose discontinuities.

We use $C_r = 3$, $C_s = 3$, $C_t = 50$, and $C_p = 10$. To synthesize trajectory instead of pose, we modify the transition cost e_t slightly. Rather than penalizing pose difference at the jumps, we penalize direction changes. The term e_t^i becomes:

4b. *Trajectory jump:* $e_t^i = C_t + C_p \left(1 - \mathbf{v}'_{\mathbf{c}_i} \cdot \mathbf{v}'_{\mathbf{c}_{i+1}}\right)$ where $\mathbf{v}'_{\mathbf{c}_i}$ is the normalized velocity of sample b_i in library stroke a_i.

Optimization. We solve the optimization with dynamic programming over a transition table with k rows for the k-NN by n columns for the stroke samples. To fill each entry in column i, we consider all the entries in $i - 1$. Once full, we recover the optimal sequence by selecting the lowest cost path through the table.

2.6 Query Stroke Synthesis

After selecting a sequence of neighbor samples, we post-process them to create the final stroke data (Section 2.6.1). For best synthesis quality, the synthesis pipeline can be implemented as a two-step algorithm, where the input trajectory is first stylized and then the pose is synthesized based on the modified trajectory (Section 2.6.2).

2.6.1 Post Processing

For the case of optimal neighbor selection we apply *transition blending*, and for the case of trajectory synthesis we perform *shape optimization*, both discussed below.

Pose Transition Blending

The optimal sequence method (Section 2.5.2) can still have discontinuities at the jump locations, despite the penalty term e_t^i that attempts to minimize them. Therefore, we employ blending to remove remaining artifacts. We gather as many as

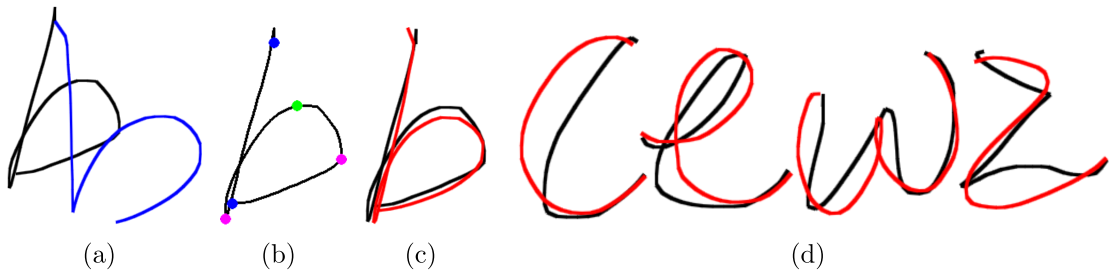


Figure 2.8: Trajectory stylization of tentatively-drawn input curves, in black. (a) Integration of the output velocities results in drift, shown in blue. (b) Anchor points define tangent and velocity constraints. (c) Optimizing to satisfy the constraints results in the final stylized trajectory, in red. (d) More examples.

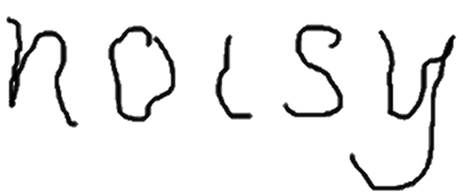
6 samples (if they exist) on either side of the jump from both library strokes that abut the jump, and perform a simple cross fade between their attributes (pose and/or trajectory) in this overlap region.

Trajectory Shape Optimization

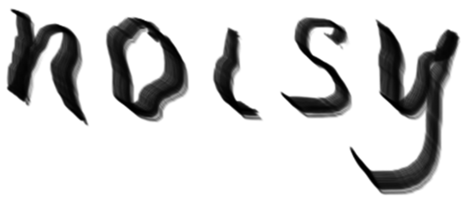
When synthesizing trajectories, the result can "drift" away from the query shape, so we correct it using shape optimization. Suppose $\mathbf{T} = \{\mathbf{x}_i\}$ is the input query trajectory. If we simply integrate the synthesized velocity $\mathbf{V} = \{\mathbf{v}_i\}$ starting from the position of the first sample $\mathbf{x}_1$, we get a new trajectory that looks like the library, but deviates from the query (Figure 2.8a black vs. blue).

We therefore design a simple linear optimization that attempts to simultaneously preserve the synthesized shape while matching a few key features of the query. We identify a set of *anchor points* on the query, of which we have three types. We place the first type of anchor point a few samples from the start and end locations (blue dots in Figure 2.8b). Second, we add locations with very rapid change in orientation, as follows. For each query sample, we calculate the stroke turning angle in a local window. We find all local maxima above a threshold angle (we use 60°) and call them *angle* points (magenta dots). Finally, between successive pairs of end points and angle points we approximate the path as a polyline using a dynamic programming optimization similar to that of McCrae and Singh [95]. The added vertices of the polylines are the third kind of anchor point (green dot).

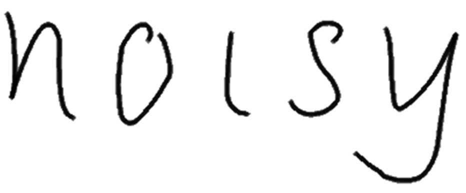
Next we set up a system of equations composed of three constraints, each of which we wish to satisfy in a least-squares sense, in order to construct the final curve (red in 2.8c, 2.8d). The first constraint attempts to match the final velocities $\bar{\mathbf{v}}_i$ to the synthesized velocities $\mathbf{v}_i$. The second constraint attempts to match the positions of the anchor points. To avoid second order discontinuities induced by optimization at the anchor points, the third constraint attempts to match the local curvature of the result to that of the synthesized velocities, near the anchors. With known synthesis velocities $\mathbf{v}_i$ and query positions $\mathbf{x}_i$ we seek the unknown positions $\bar{\mathbf{x}}_i$ via a system of



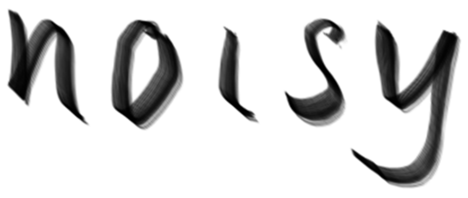
(a) Noisy trajectory



(b) Pose synthesis on noisy trajectory



(c) Neatened trajectory



(d) Pose synthesis on neatened trajectory

Figure 2.9: Stylization results on noisy input.

equations:

$$0 = \sum_{i=1}^{s} (\bar{\mathbf{v}}_i - \mathbf{v}_i)^2 \tag{2.1}$$

$$0 = C_a \sum_{i \in \mathbf{A}} (\bar{\mathbf{x}}_i - \mathbf{x}_i)^2 \tag{2.2}$$

$$0 = C_k \sum_{i \in \mathbf{A}} \sum_{j=i-\varepsilon}^{j=i+\varepsilon} (\kappa(\bar{\mathbf{x}}_j) - \kappa(\mathbf{x}_j))^2 \tag{2.3}$$

where $\bar{\mathbf{v}}_i = \bar{\mathbf{x}}_i - \bar{\mathbf{x}}_{i-1}$, and $\kappa(\mathbf{x}_i) = \mathbf{x}_{i+1} + \mathbf{x}_{i-1} - 2\mathbf{x}_i$. The set $\mathbf{A}$ contains the indices of the anchor points. ε is the number of surrounding samples involved in the second order constraints. In our experiments, we use $\varepsilon = 2$, $C_a = 0.25$, and $C_k = 2$.

2.6.2 Synthesizing Both Pose and Trajectory

The output of our synthesis process so far is a sequence of poses and/or trajectories that can drive a virtual brush model. However, our pose synthesis framework uses trajectory as input. If both pose and trajectory are being synthesized, the algorithm can be run in either one or two passes. In the one pass version, the query trajectory is used to directly synthesize both the pose and new trajectory and thus the combined output does not have ideal correlation. In the two pass version, first the input stroke (Figure 2.9a) is used to synthesize a new trajectory (Figure 2.9c), and then that output trajectory is used to synthesize pose data (Figure 2.9d), which creates a data dependency but produces better correlation. Note that, compared to the pose synthesis result on the noisy input trajectory (Figure 2.9a), synthesizing pose after trajectory stylization produces more pleasing result (Figure 2.9d). Another advantage to the two-pass version is that it allows us to use separate parameters for synthesizing

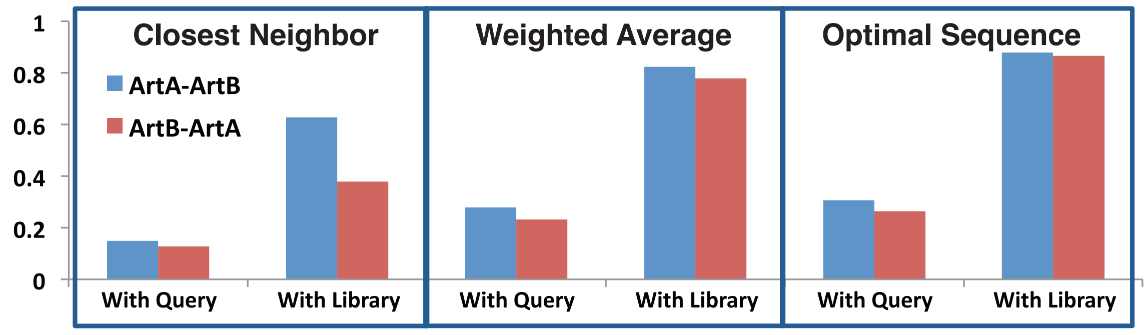


Figure 2.10: Synthesized hand-pose correlation. We apply ArtA to ArtB, and vice-versa, and see that the output is well correlated with the library and not the query, and optimal sequence performs best.

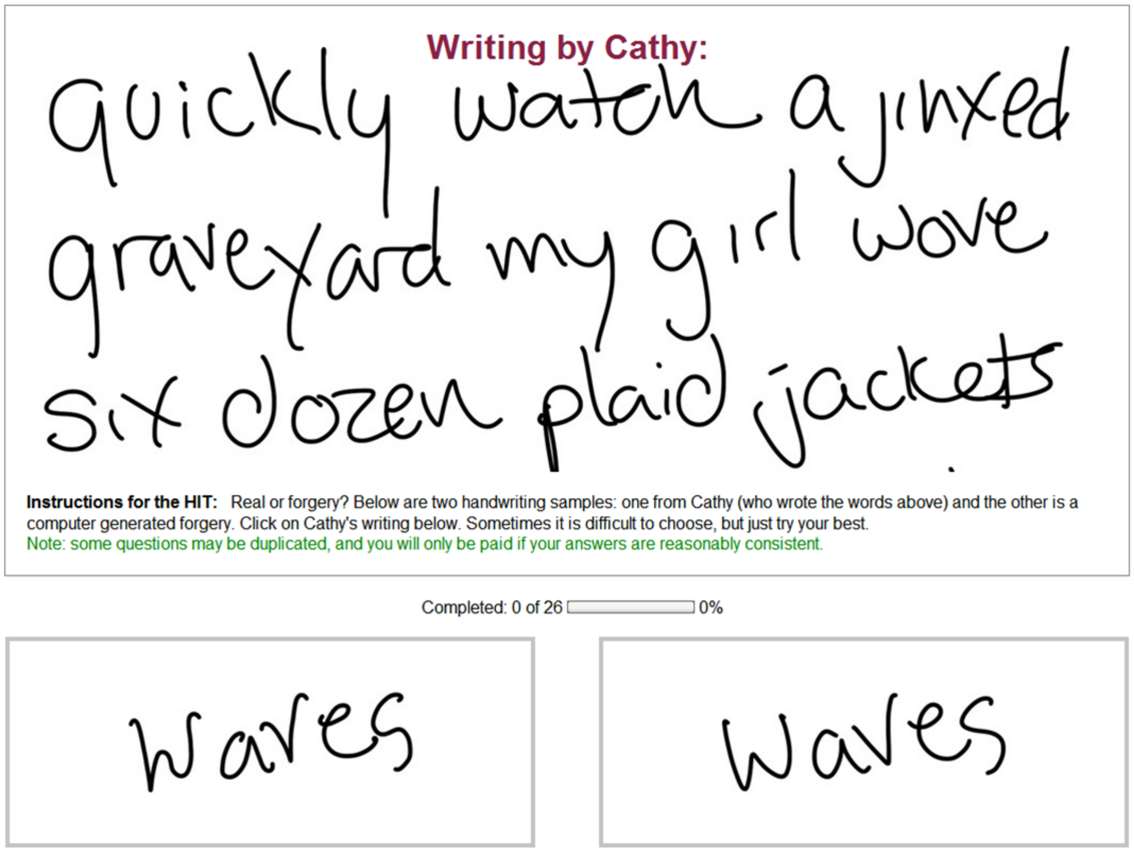
trajectory and pose. For example we generally tune for fewer segments in trajectory synthesis than in pose synthesis in the two-pass case.

2.7 Evaluation

This section describes the studies we conducted to evaluate our results. As a quantitative measure of the pose synthesis quality, we computed the correlation between the synthesized results and ground truth. However, there is no equivalent measure for trajectory synthesis and ultimately we care most about how users judge our results. Therefore, we also conducted user studies on both pose and trajectory synthesis.

2.7.1 Quantitative Analysis

Section 2.3.1 shows that an artist exhibits some natural variation in hand pose even when drawing the same path multiple times. Therefore, there is no "gold standard" that we can compare our synthesis results against—L_2 measurement of reconstruction error against the ground truth is not a good indication of success. Correlation on the other hand is a better evaluation metric in this case. For pose synthesis, we can apply analysis similar to Section 2.3.1 to evaluate our results. If the style of the library is successfully transferred onto the query strokes, every local patch of the output should have a similar pose profile to similar patches in the library. We compared the closest neighbor, weighted average, and optimal sequence variants of our algorithm (Section 2.5.2). Figure 2.10 contains the correlation results. It shows that the optimal algorithm achieves the highest pose correlation in both synthesis cases. Furthermore, since we have ground truth pose data for the query strokes, we compare them to the synthesis and find they are not well correlated, as expected. Note that we have no analogous quality measure for trajectory synthesis because the goal is a blend of local features with global shape, and a fair objective function is more difficult to formulate.


quickly watch a jinxed graveyard my girl wove six dozen plaid jackets
Waves
Waves

Writing by Cathy:

Instructions for the HIT: Real or forgery? Below are two handwriting samples: one from Cathy (who wrote the words above) and the other is a computer generated forgery. Click on Cathy's writing below. Sometimes it is difficult to choose, but just try your best.
Note: some questions may be duplicated, and you will only be paid if your answers are reasonably consistent.

Completed: 0 of 26 0%

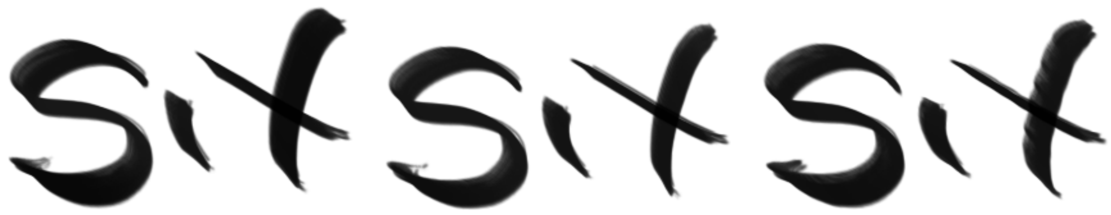
Figure 2.11: User study interface.

2.7.2 User Studies

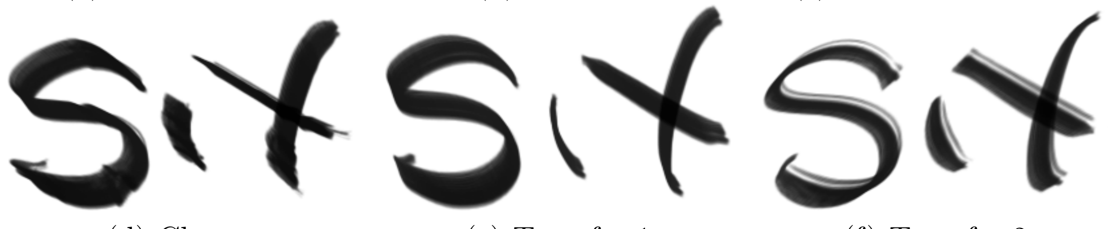
We conducted two user studies with the goal of evaluating whether a user can distinguish between our results and ground truth. Pose and trajectory are studied separately because users are overwhelmingly sensitive to trajectory over pose, which would make pose evaluation difficult if combined. Both studies use the same methodology; only the images are different.

Study Design. We show the subject three lines of English text from an artist's library—the *exemplar* (we used two libraries from each of the artists shown in Figure 2.2a, 2.2b). Below the exemplar appear two test images containing the same (3-9 letter) phrase—one *ground truth* originally written by the artist, and one *forgery* synthesized by one of our algorithms. The subject is instructed that one image is an "original" and the other is a "forgery," and is asked to identify the original by comparing both with the exemplar. Figure 2.11 shows the design of the interface. In the pose study, the ground truth and forgery have the same trajectory (Figure 2.12). In the trajectory study, the ground truth trajectory is used for synthesis, and pose data is omitted from all images (Figure 2.13).

One trial consists of 26 such tasks (each with the same exemplar) comparing (a) a random ground truth phrase, and a forgery selected from one of five conditions: (b) optimal sequence, (c) weighted average, (d) closest neighbor, (e) style transfer,

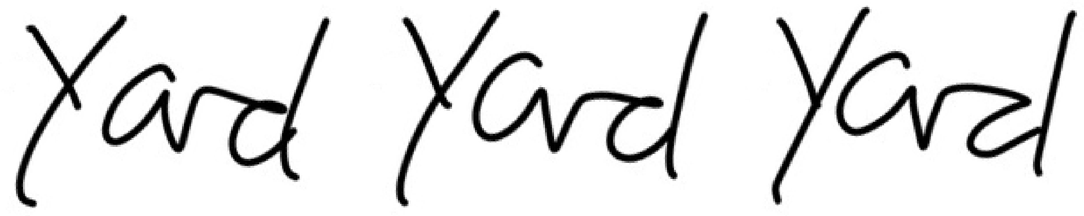


(a) Ground truth (b) Optimal (c) Weighted average

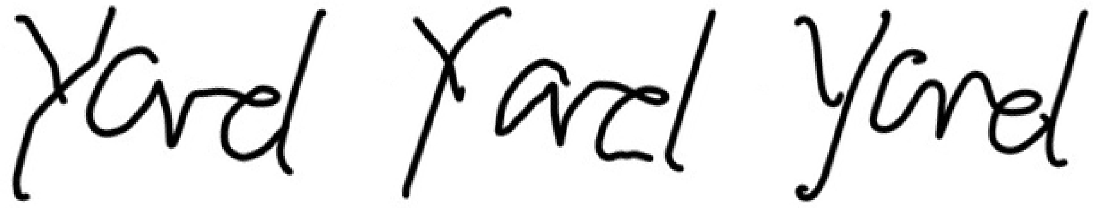


(d) Closest (e) Transfer 1 (f) Transfer 2

Figure 2.12: Example images from pose study. (b-c) are algorithms being tested, and (e-f) have pose transferred from other artists.



(a) Ground truth (b) Optimal (c) Weighted average



(d) Closest (e) Transfer 1 (f) Transfer 2

Figure 2.13: Example images from trajectory study, organized as in Figure 2.12.

and (f) style transfer for validation. Conditions (b-d) are the algorithms being tested, whereas (e-f) are baselines that synthesize the forgery with another artist's library (so should be very easy to identify). Of the 26 tasks, four are from each of (b-e) and ten are from (f), randomly shuffled. For any particular subject, (e-f) are selected from two different library styles, randomly. We only retain data from a trial when 9 out of 10 of the validation conditions (f) are correct, so we know the user understands the task and is actually trying to succeed ($p = 0.01$). Then style transfer results are only reported for (e).

Subjects were "workers" on the Amazon Mechanical Turk, a micro-job marketplace that has been used for a growing variety of such studies [22]. Our studies were restricted to US-only workers who were paid \$0.20 per task (a "HIT" containing the 26 comparisons, which they typically completed in a few minutes). Across the two studies, 70 workers completed a total of 214 HITs. Subjects were allowed to complete

	correct / count (%)	
method	pose study	trajectory study
average	153 / 322 (48%)	96 / 302 (32%)
optimal	169 / 316 (53%)	172 / 305 (56%)
closest	269 / 304 (88%)	220 / 292 (75%)
transfer	293 / 301 (97%)	274 / 288 (95%)

Table 2.1: Pose and trajectory user study results. Each entry is the number of times ground truth was correctly chosen out of a number of paired comparisons for each condition.

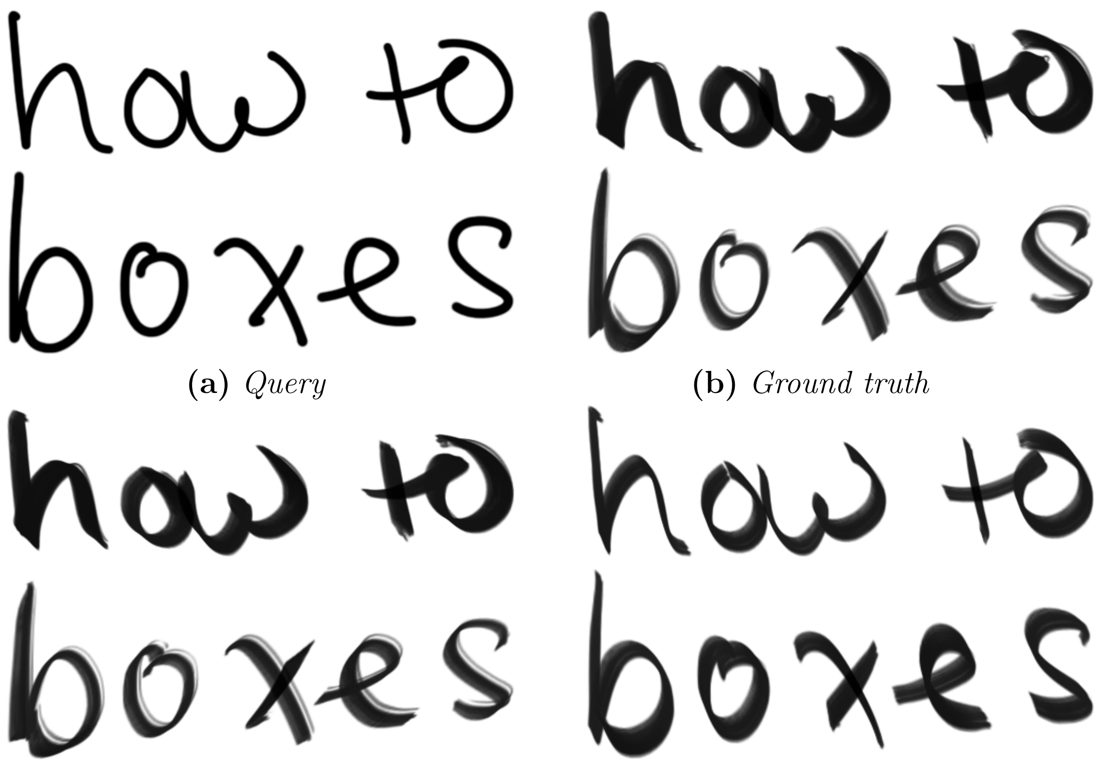
as many as 10 HITs per study, but most workers did just one. For cases where a particular pair was repeated by a particular worker due to the randomized selection, we only retained the first such response, for total of 4,170 responses.

Study Results. The results are reported in Table 2.1 as the frequency with which the ground truth was correctly identified. A Bonferroni corrected, randomized permutation test on the distributions for each consecutive pair of rows in the table shows that they are different with statistical significance ($p \ll 0.01$) except in the case of the average and optimal conditions in the pose study ($p = 0.34$).

The ideal forgery is indistinguishable from ground truth so subjects will choose randomly, resulting in a correct answer near 50% of the time. When the forgery is easy to identify (Figure 2.13e), we expect scores near 100%. The *transfer* condition is near 100% in both studies, which shows people can perform the task in easy cases. People tend to identify the *closest* method (88% and 75%), so it is generally implausible. In the pose study, *average* and *optimal* are both near 50%, and therefore plausibly synthesize pose data. *Optimal* performs well in the trajectory study, but *average* is significantly below 50% which means subjects tend to incorrectly identify the forgery as the "original." As mentioned in Section 2.5.2 the weighted average method produces a very smooth output trajectory, and we believe people have a bias towards smoother curves. In such cases people choose average over ground truth, even though it is actually smoother than the exemplar and thus fails to mimic the exemplar's style.

2.8 Results and Discussion

The running time of our synthesis algorithm is sublinear with the number of library samples, due to the approximate k-NN search. We found through experimentation that 150 strokes is sufficient for generating plausible results, and confirmed this through correlation analysis similar to that of Section 2.3. For such a library, the average running time for both pose and trajectory synthesis is 0.08 seconds per stroke. We experimented with "enriching" the library up to about 1000 strokes by including the same strokes scaled to different sizes (±50%), to increase matching scale-invariance, but did not see a significant improvement, and performance was reduced. We also tried using very small (fewer than 10) and very large (more



(a) *Query* **(b)** *Ground truth*

(c) *Synthesis* **(d)** *Alternate style*

Figure 2.14: Handwriting pose synthesis results. Query strokes (a) are ground truth strokes (b) without the pose data. (c) Our synthesis results are visually indistinguishable. (d) Applying another artist's style yields characteristically different visual appearance.

than 500) libraries. With small libraries, feature matching cannot find similar enough trajectories and therefore picks random strokes to copy. Since the optimal sequence algorithm encourages long segments, the pose results still look plausible, but for trajectories, poor matches may result in complete loss of semantics in the results. Large libraries do not affect pose synthesis, whereas trajectory modification is more likely to find very similar patches to the query, reducing the effect of style transfer.

All the results shown here are rendered using Adobe Photoshop's bristle brush tool [31] with the same brush settings. The differences in the synthesized pose data induce the visibly different shape and texture of the strokes.

Figure 2.14 shows the plausibility of pose synthesis on handwriting. An artist's 6-DOF strokes are stripped of their pose data and the trajectories are used as queries with the same artist's library applied. The synthesized output is visually indistinguishable from the ground truth, whereas applying another artist's library creates a visually distinct result. Pose synthesis on line drawings is demonstrated in Figure 2.15.

Examples of trajectory stylization are in Figure 2.18 and Figure 2.16. We show robustness to noisy input and the utility of trajectory stylization by demonstrating how it can neaten mouse-written text, which has characteristic shakiness. Similarly,

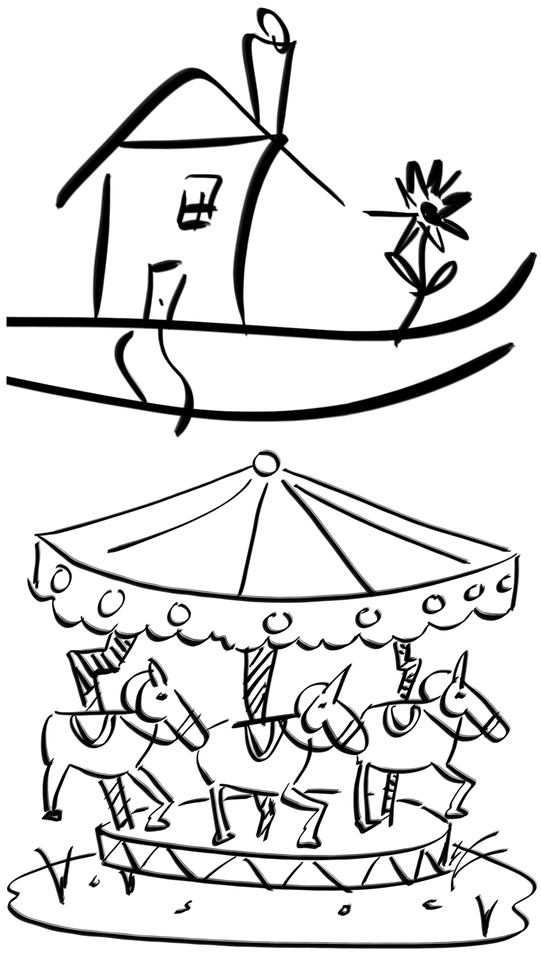
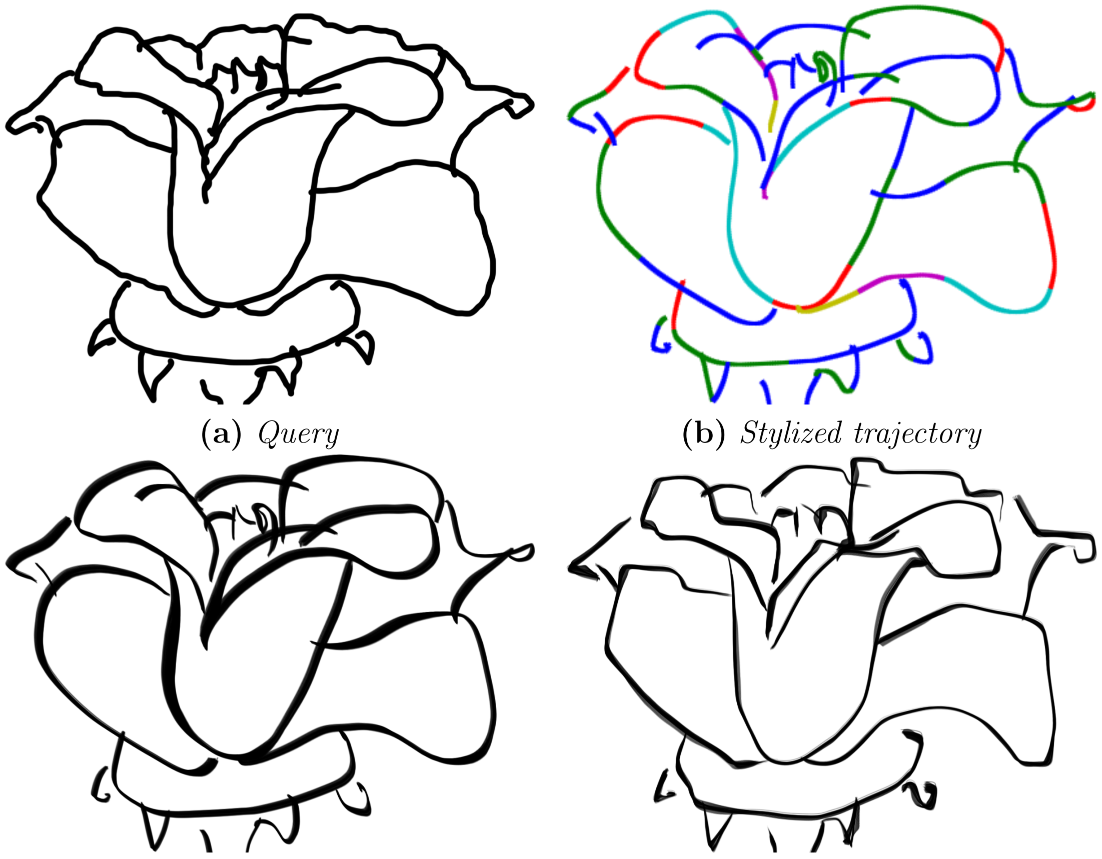


Figure 2.15: Line drawing pose synthesis results with two different styles.

a line drawing made by an artist with an unsure hand can be made to have more confident strokes. Our algorithm can also apply stylistic flourishes such as serifs or block lettering. Figures 2.1, 2.16, 2.17, and 2.18 show the two-pass algorithm stylizing both pose and trajectory.

Discussion. Many of our results are handwriting samples, though we do not make special accommodations for them, because handwriting provides an easy to understand and readily available source of data. Handwriting is uniquely stylized by the writer's hand pose, the local trajectory variations, and the global features such as slant angle. Handwriting also provides readily-evaluated test sets; in contrast, it is more difficult to measure performance on strokes in a drawing. We consider that the pose and local trajectory are stylistic results of motor reflexes, whereas the global shape is intentional. Our algorithm attempts to transfer style but maintain intent (via the trajectory shape optimization). For handwriting, we could use auxiliary information such as the semantic content of the text to improve stylization, but that would not be generalizable to other types of artistic strokes. Similarly, while an artist's style may consistently place higher weight on strokes in a drawing that are


(a) *Query* **(b)** *Stylized trajectory*

(c) *Trajectory and pose* **(d)** *Alternate style*

Figure 2.16: Line drawing pose and trajectory synthesis results. The flower is drawn by an amateur user. The different colors represent different consecutive segments.

nearer to the viewpoint, we do not attempt to model these variations. In this way, we provide the most general result possible.

We examined both shape contexts and filtered velocities for feature vectors. Our results use filtered velocities, though it is difficult to say which is absolutely better. Figure 2.19 shows their limitations. Filtered velocities tend to perform very well, but can sometimes get "confused" and match a letter to a different but similar letter. This may be fixable with a handwriting specific algorithm, but we make no assumptions about the types of strokes being made. Shape contexts have fewer of these errors, but have difficulty balancing local style transfer with global shape, resulting in output that appears less plausible and noisy. Also it is generally more difficult to select parameters for shape contexts, whereas filtered velocities are more robust through a broader range of settings. Finally, to get high quality results, the shape contexts need to have more dimensions (see Section 2.5.1) and this impacts performance.

We also compared strategies for choosing among nearest neighbors: closest neighbor, weighted average, and optimal sequence. The result of our pose evaluations is that quantitatively, optimal sequence is best, but to the casual observer, weighted average is as effective. However, careful inspection can reveal subtle differences between the two (e.g. Figure 2.12b-c). For trajectory synthesis, weighted average

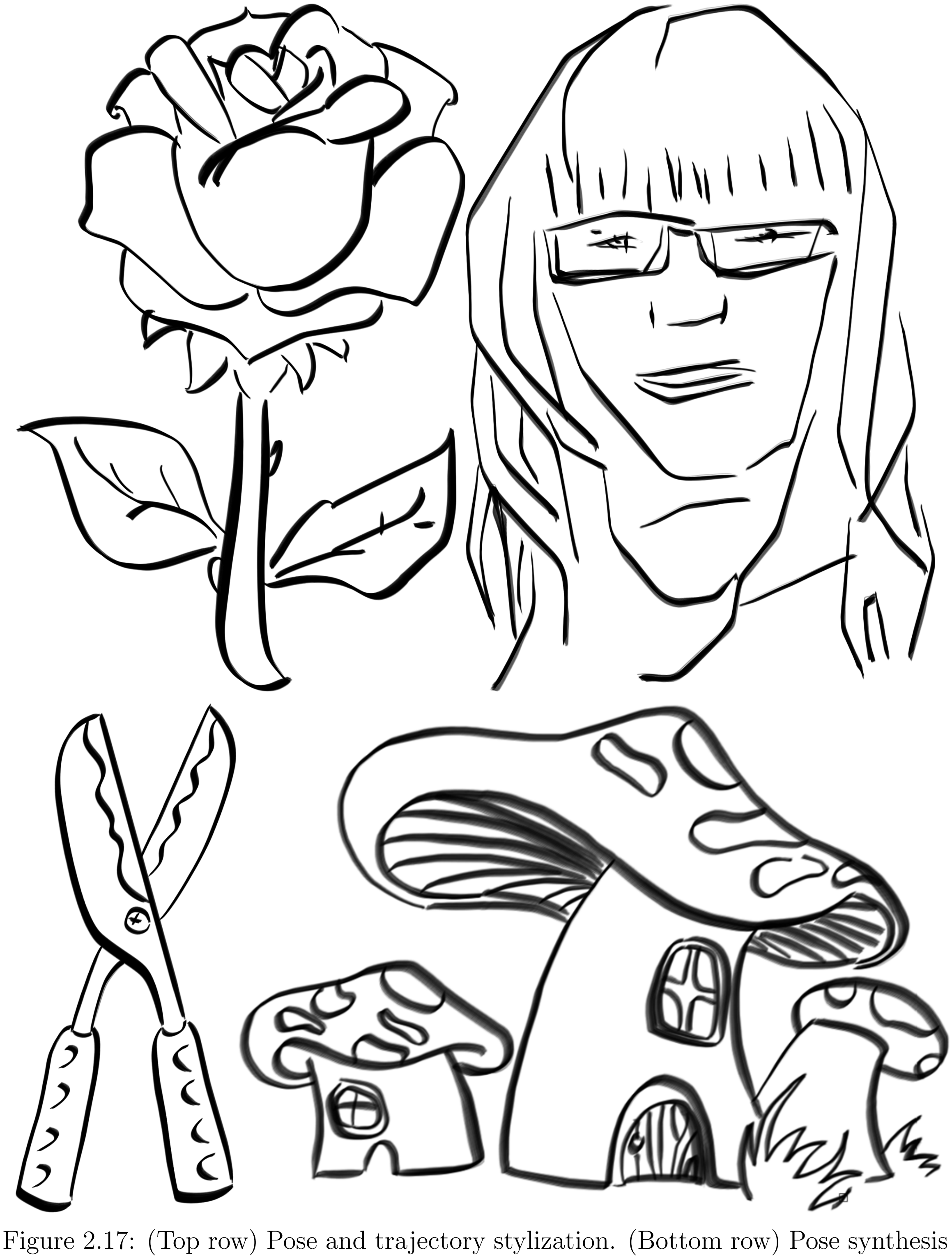
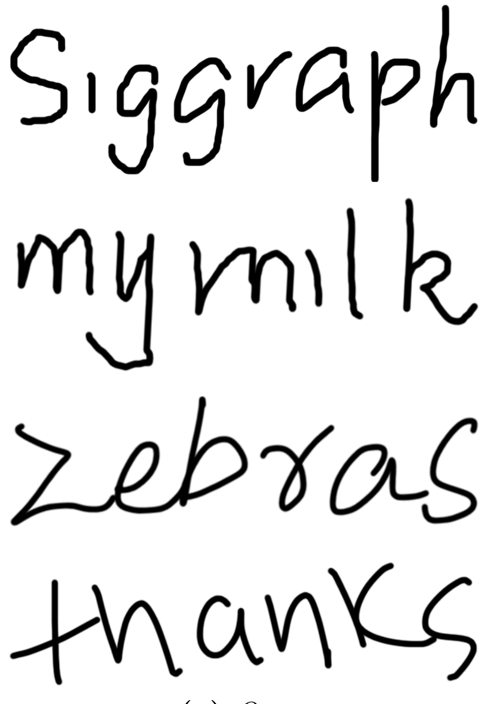
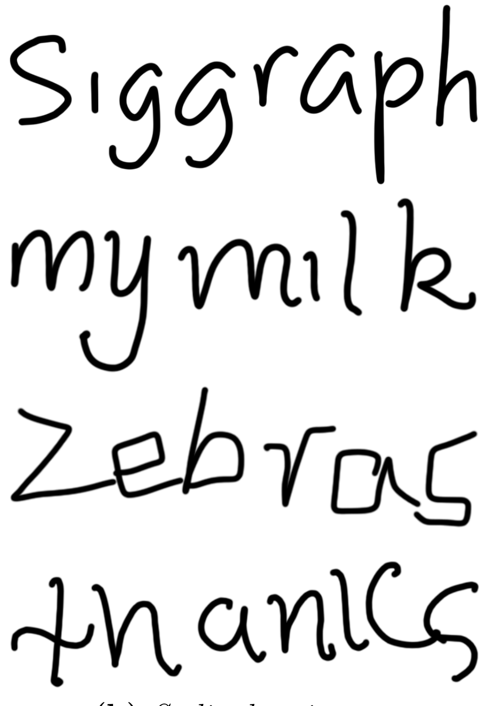


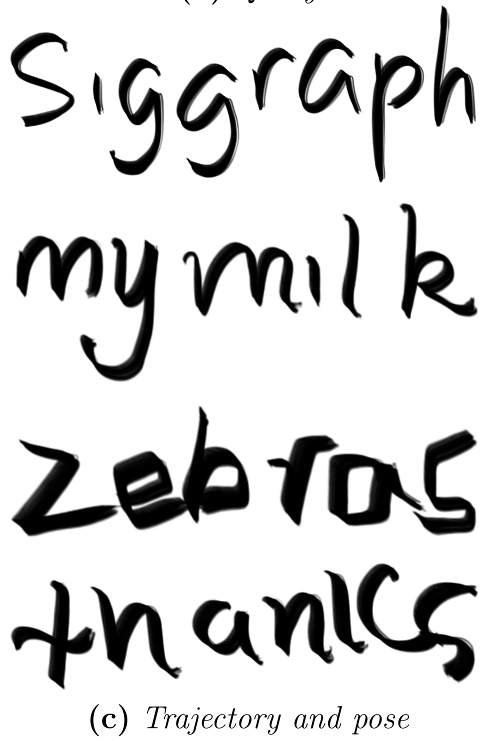
Figure 2.17: (Top row) Pose and trajectory stylization. (Bottom row) Pose synthesis for vector line art from Adobe Illustrator.



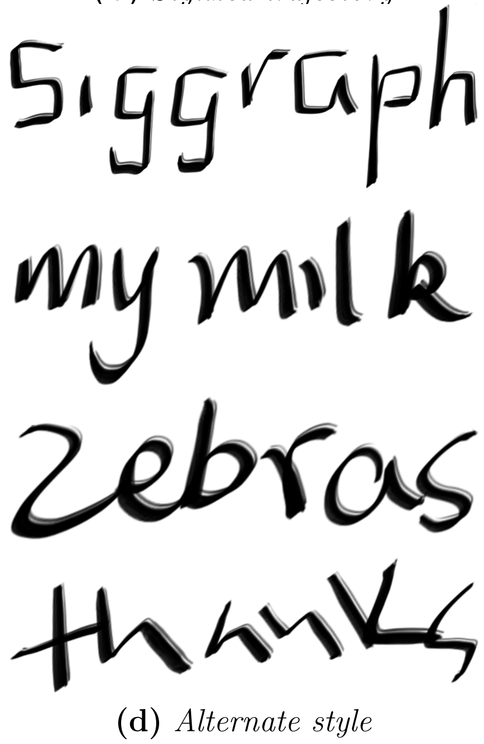
(a) *Query*



(b) *Stylized trajectory*



(c) *Trajectory and pose*



(d) *Alternate style*

Figure 2.18: Handwriting pose and trajectory synthesis results. The words "Siggraph" and "my milk" are written with the mouse. The words "zebras" and "thanks" are written using a 2DOF stylus.

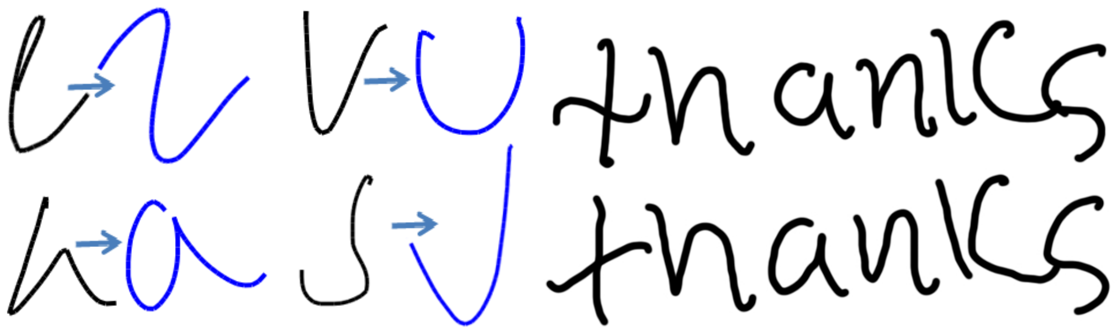


Figure 2.19: Synthesis limitations. (left) Filtered velocities can cause a query letter (black) to map to a different library letter (blue) with similar trajectory. (right) Filtered velocities (above) generally produce more visually pleasing results than shape contexts (below).

is obviously more smooth, and in some cases too smooth. The advantage weighted average has over optimal sequence is that it does not require the query stroke be completed before it begins synthesizing, so it can be used in an "on-the-fly" implementation that synthesizes pose data while the user is still creating the stroke. Otherwise, optimal sequence results in the highest quality output.

Finally, our approach does not require hand-drawn input. The method applies to any kind of 2D path, such as Bézier curves (see Figure 2.17). The paths are first sampled uniformly in arc length and then treated as constant-velocity brush strokes by our unmodified algorithm. This approach would also work for lines extracted from 3D models or other automatic processes.

2.9 Limitations and Future Work

Our algorithm considers each stroke independently, but artistic styles also include interactions among nearby strokes, both temporally and spatially, which we hope to model in future work.

Our results are all rendered by Adobe Photoshop's bristle brush, which uses all 6-DOF of the stroke data. However the visual impact of each DOF may not be as obvious as it would be with a real physical brush. Because our algorithm only generates stroke data, it can be used with any digital paint engine, so other brush simulations may be able to use the data more effectively.

Figure 2.19 highlights some trajectory synthesis failures where the query shape was poorly matched by our algorithm. Our approach is local and has no higher-level notion of intent on the part of the artist. While the cognitive process is too complex to model there may be some intermediate levels that could help. Moreover, these failures could be addressed in our current framework by choosing different parameter settings. Our approach works well, especially for pose synthesis for a broad range of settings, but finding the *ideal* values for an arbitrary problem is difficult.

Finally, we treat the artist's library as a single monolithic style. In practice, artists may choose among multiple styles, even in a single drawing. A style is really more multi-modal than is characterized by our process. We can partially address this issue by offering the user the choice of different styles, but finding an effective interface for this remains an interesting problem. More challenging would be to try to characterize the multi-modal nature within a library.

2.10 Conclusion

We present HelpingHand [91], a data-driven approach for synthesizing plausible pose data that can be used to generate expressive handwriting and line drawings. The same framework can also be used to transfer stroke trajectory style. We collect a library of strokes drawn by artists on a 6-DOF tablet in Adobe Photoshop and analyze the pose data as insight for future research in this area. We have made our program executable and the exemplar database available on-line in the hope that they will be helpful for future research.

Chapter 3

RealBrush: Painting with Examples of Physical Media

Conventional digital painting systems rely on procedural rules and physical simulation to render paint strokes. This chapter presents an interactive, data-driven painting system that uses scanned images of real natural media to synthesize both new strokes and complex stroke interactions, obviating the need for physical simulation. First, users capture images of real media, including examples of isolated strokes, pairs of overlapping strokes, and smudged strokes. Online, the user inputs a new stroke path, and our system synthesizes its 2D texture appearance with optional smearing or smudging when strokes overlap. This work builds on the matching algorithm introduced in Chapter 2 and uses it to construct full stroke by example (not just hand-pose). We demonstrate high-fidelity paintings that closely resemble the captured media style, and also quantitatively evaluate our synthesis quality via user studies.

3.1 Background

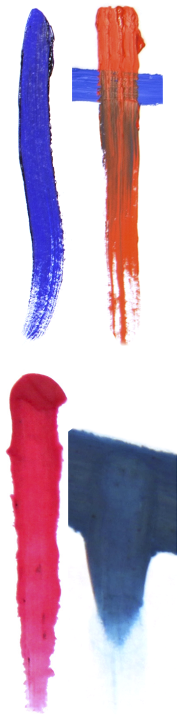
Traditional artists working with natural media take advantage of a great abundance of different materials. These can have different chemical properties, such as wet, dry, or cracked paint. Pigments can be scratched or mixed with binders or thinners. Materials can have 3D relief or embedded particles. The artist can express her creativity utilizing materials found in the wild, limited only by her imagination. In contrast, digital artists are heavily restricted. High quality approximations of commonly used media such as oil and watercolor have been achieved in research systems [13, 21]. However, these systems rely on complex simulations that cannot easily generalize to new media. Commercial painting tools achieve a wider range of effects with procedural algorithms, at the cost of requiring significantly more effort to generate a similar level of result fidelity. Ultimately, artists are limited by the built-in assumptions embodied by digital painting software.

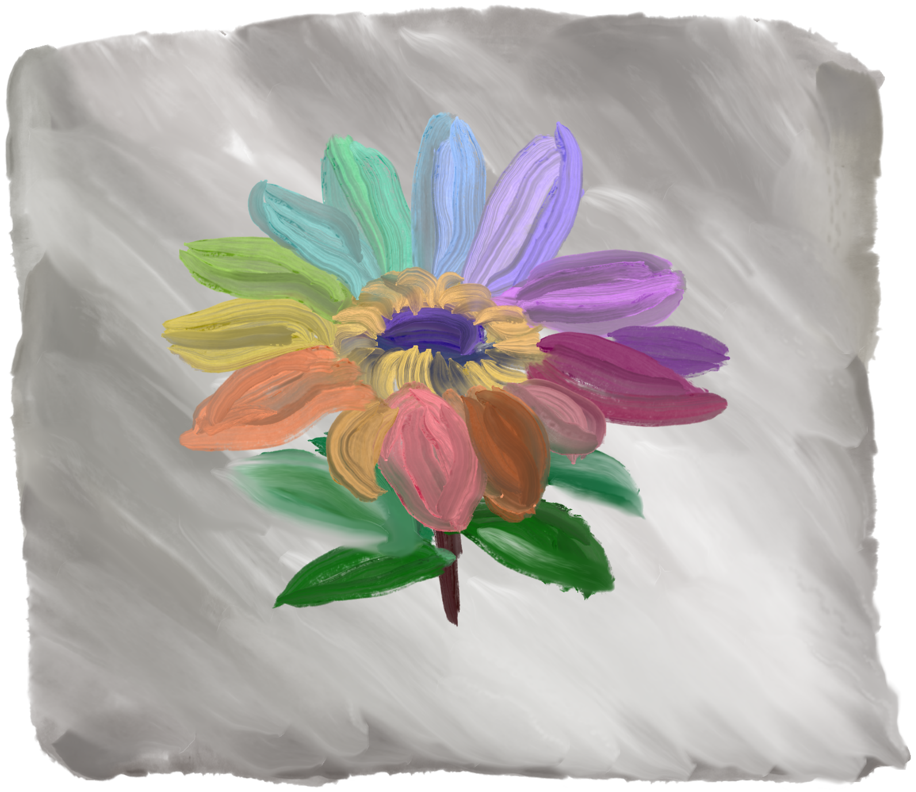
This chapter introduces “RealBrush”, a system that allows artists to paint digitally with the expressive qualities of any physical medium. As input to the system, the artist first makes some example strokes demonstrating the medium’s behaviors and scans them in. The scanned images are processed to make a “library” (Figure 3.1a). Within RealBrush, the user can select this library and make new strokes that are synthesized using the library to create novel marks in the same style (Figure 3.1b). In this manner, the artist can digitally paint with whatever physical media she feels most comfortable, without requiring the development of a custom simulation algorithm (see Figure 3.2). We accomplish this by using a data-driven approach that derives the full knowledge of a medium from the example library, making only the minimal necessary assumptions about physicality and locality.

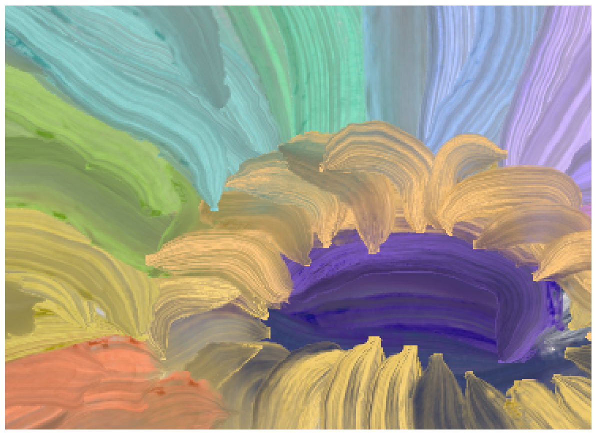
Expressive media can achieve different appearances based on the artist’s skill and technique. In constructing an oil paint library, the artist may thickly apply paint, or may make smooth, textureless strokes. A watercolor artist may paint strokes in a calligraphic style. Therefore, libraries encode not only the physical properties of the media, but also their application in a specific style by a trained artist. With a calligraphic watercolor library, a novice user may make strokes of much higher quality than he could with a real brush.

The shape of individual strokes is only a small part of the behavior of physical media—traditional painting is not possible without the complex interaction between strokes. Wet pigment can be advected by subsequent strokes to create smears (Figure 3.1c), or particles may be pushed around the canvas by finger smudges (Figure 3.1d). The myriad of potential effects creates an undue burden to attempt to capture examples of all different behaviors for a library. A generic data-driven algorithm that is not tailored for natural media could easily become impractical due to requiring intractably large amounts of data.

Our main contribution is the plausible reproduction of natural media painting in a practical and fully data-driven system. This is possible because we factorize the range of physical behaviors into a tractable set of orthogonal components that can be treated independently, which is motivated by our analysis of the space of natural


(a)


(b)


(c)

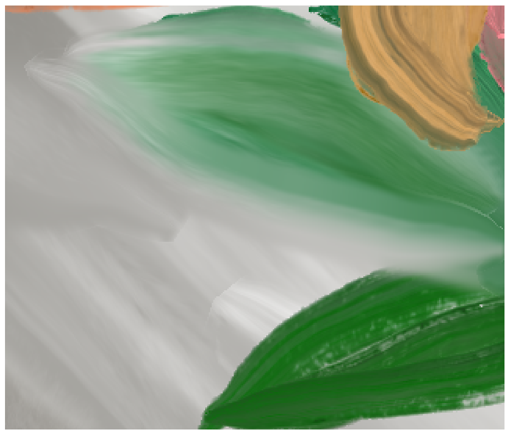

(d)

Figure 3.1: A simple painting created by our system. Left to right: (a) shows oil (left) and plasticine (right) exemplars which are used to synthesize the painting in (b). The foreground flower strokes use oil exemplars, while the background strokes use plasticine and are smoothed with our smudging tool using. (c) (d) show close-ups of the smearing and smudging effects.

media. We discuss the implications of this factorization, and present algorithms for capturing and reproducing each of these behaviors. Our results demonstrate the algorithm with images painted by artists using a variety of different captured media and with a quantitative evaluation of the realism achieved by RealBrush.

3.2 Related Work

Despite the remarkable progress made by digital painting software to perform increasingly sophisticated image processing, it remains difficult to achieve results that mimic the textural qualities of real natural media. Established results fall into three broad categories.

Procedural approaches rely on heuristics explicitly designed to mimic specific natural media behaviors and effects, based on the developer's intuition [32]. Commercial packages such as Adobe Photoshop use these techniques. They provide a wide range of effects for digital artists, but they have difficulty reproducing the visual fidelity of natural media and can require significant user effort to achieve a convincing level of realism.

For higher fidelity results, researchers have developed **simulation approaches** that numerically model the physical interaction between the virtual brush, the canvas, and the pigment medium. Specifically, modeling deformable 3D brushes and simulating paint fluid flow through a canvas are increasingly common, and have been applied successfully for particular natural media, including watercolor [21], oil paint [13, 31], and pastels [135], with impressive results. Commercially, Microsoft's Fresh Paint application implements recent work [9, 19] to create an oil paint system. However, these results are all carefully tailored to their respective natural media, and extending any of them to other types of artistic tools is difficult. Our focus is a more general digital painting system that can support arbitrary natural media as supplied by the artist while maintaining high visual fidelity.

Our work belongs to the class of **data-driven approaches**, which includes such work as modeling virtual brushes [9, 154], generating brush stroke paths [148, 150], and simulating pigment effects [152]. Perhaps most straightforward is using scanned marks as basic drawing units (footprints), interpolated along a path to produce novel strokes [149]. However these techniques all still depend on physical simulations and procedural rules which limit the fidelity and generality of their results. For realistic synthesis of arbitrary media, we use a solely data-driven approach and avoid any physical model or procedural rules.

Researchers have deformed strokes to match new paths. Zeng et al.'s work on applying painterly styles to photographs [162], and Xu et al.'s decomposition and deformation of Chinese paintings [155] both warp images of real strokes to create novel shapes. Earlier work by Hsu et al. [57] developed a system based on stylizing lines with warped pieces of art, which has become a core algorithm of commercial programs such as Adobe Illustrator. All of these systems suffer from a common limitation though, that they can only support a small range of deformations of example strokes before

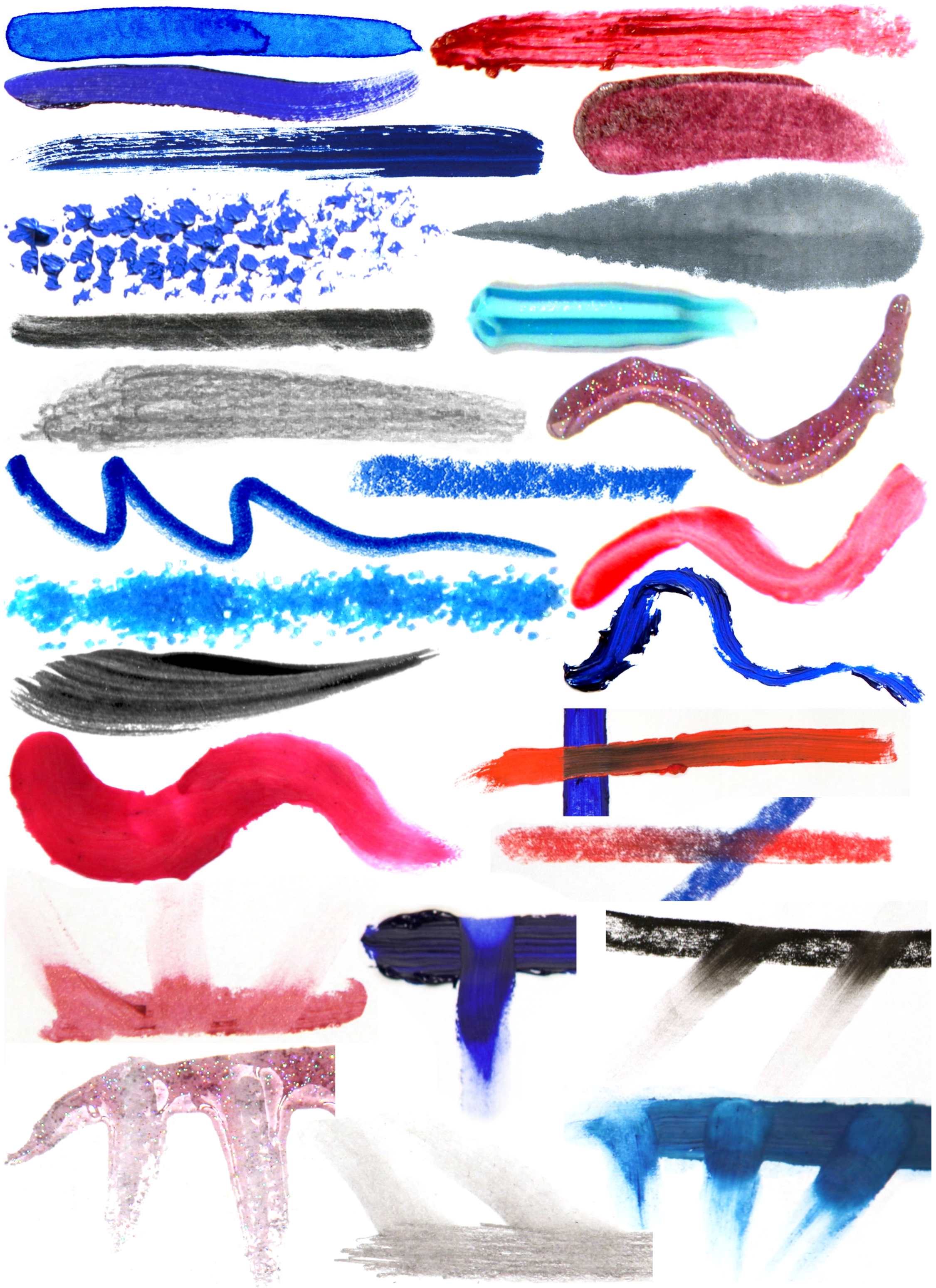


Figure 3.2: Acquired natural media samples, including paint, charcoal, pastel, marker, lip gloss, plasticine, toothpaste, and glitter.

the altered appearance becomes unrealistic. Zhou et al. [163] achieve a wider range of deformations, but only from individual exemplars with uniform texture.

Most similar to our approach is the work of Ando and Tsuruno [3], and Kim and Shin [70], which use images of real strokes to stylize user input lines. These works focus on single stroke synthesis, which is a part of our approach, but we additionally explore data-driven smearing and smudging. Both approaches make aggressive simplifications, losing fidelity to gain performance. That each system relies on small patches that are not deformed fundamentally limits these techniques by forcing many patch boundaries within a stroke, which is frequently where artifacts occur. The lack of deformation also reduces the range of shapes they can synthesize from few example strokes, and therefore increases the computational burden per stroke by requiring more search. These small patches further limit the scale of texture features that can be reliably reproduced from example media, because patch boundaries can cause jarring discontinuities. RealBrush has two significant advantages over these approaches. First, we are able to support a wider array of artistic media. Second, we gain higher fidelity: we are able to reproduce strokes that are indistinguishable from real examples, as demonstrated by our user study.

Also related is work in **texture synthesis**. Wang et al. [141] generate textures sampled from paintings to stylize 2D images. Yan et al. [156] apply a similar technique for shading 3D geometry. Neither of these approaches provide a means for interactive, paint-style control. Conversely, Schretter [123] and Ritter et al. [113] both demonstrate brush-based painting systems that use texture synthesis to fill arbitrary regions with acquired samples. However, they do not support oriented anisotropic textures, and they do not consider interaction between overlapping textured regions.

3.3 Understanding Natural Media

Natural media exhibit a tremendous number of different types of behaviors (see Figure 3.2). Simulation based approaches rely on complex physical models to control these effects from a small number of parameters. They pose a daunting problem for data-driven approaches, because of the huge number of different examples that need to be acquired to reasonably approximate the medium.

3.3.1 Factorization

We factor the effect space into four orthogonal bases—“shape,” “smear,” “smudge,” and “composite”—that make the data acquisition and search problem tractable. Shape refers to the silhouette and texture of a single stroke made in isolation. Many obvious media features are part of shape including watercolor’s darkened edges. However, most of the important qualities of natural media are due to the interactions among multiple strokes. Smear is exemplified by applying a new stroke across wet paint, “dirtying” the brush and smearing the two paint colors together (Figure 3.3b,c). Smearing is the core component of color blending on

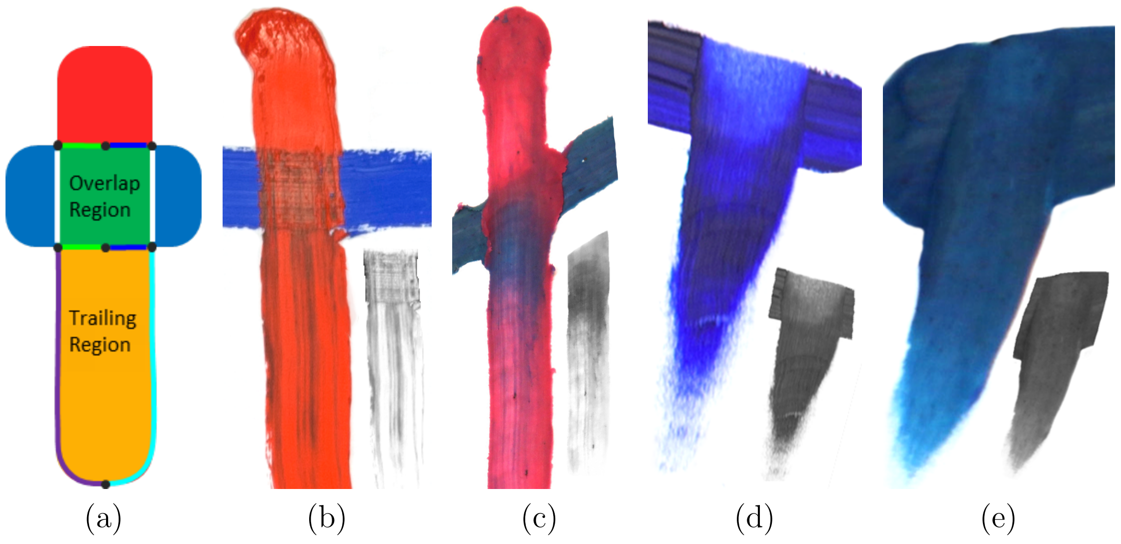


Figure 3.3: Smearing and smudging exemplars. (a) We detect the overlap and the trailing regions. They are highlighted in green and yellow and are represented by eight polyline segments; (b) (c) show oil and plasticine smearing exemplars; (d) (e) show oil and plasticine smudging exemplars. The extracted smearing and smudging operators are inset.

canvas, and is also referred to as "bidirectional" pigment transfer (from the brush to the canvas, and vice versa). Smudge specifically refers to stroking over wet paint with a clean brush or finger to make a smudge mark, but we use it more generally to refer to any action that modifies pigment on canvas without applying more pigment, including using other implements such as salt, a blowdryer, or tissue paper (Figure 3.3d,e). Finally, composite refers to the way colors mixed or applied atop one another optically combine to form the final reflected color, such as in glazing or overlapping transparent strokes, or during bidirectional pigment transfer.

Each basis can be captured in isolation, reducing the burden for data-driven acquisition and synthesis. Shape exemplars are isolated strokes. Smear exemplars come from crossing paint strokes of two different colors, so the mixing proportion can be determined by color. Smudge exemplars do not have a foreground color, so registered images of the canvas paint are needed, before and after the smudge is applied. Composite exemplars are overlapping strokes of different colors, so the combined color can be sampled.

In proposing this factorization, we reduce the library size that must be acquired from n^4 to $4n$, where n is the number of examples needed for each basis. We leave composite as future work, and focus on achieving realistic results for shape, smear, and smudge. See Figure 3.2 and Section 3.5 for examples of these effects.

3.3.2 Varieties in Media Texture

One complicating aspect of natural media, even in light of our factorization, is the wide variety of types of texture and appearance that are exhibited within each of our bases. Consider the following media and the characteristics of their shapes: watercolor has smooth internal textures with some granularity and sharp silhouettes. Oil paint has directional texture and sparse large features (blobs). Toothpaste has strong 3D structure. Glitter has noisy texture and sporadic application. Lip gloss has 3D structure and sparse noise texture. This is just a selection of possible variations.

Any single technique that could reliably synthesize all of these media efficiently would be a major advance in texture synthesis. Therefore it is clear that a small toolbox of core algorithms will be necessary to produce realistic results across all media.

3.4 Algorithm Overview

Our factorization of natural media behaviors suggests a factored algorithm (Figure 3.4). RealBrush is a data-driven painting implementation of this theory including shape, smear, and smudge behaviors, which are each treated separately and then combined.

First, shape, smear, and smudge examples are collected and processed to generate media libraries (Section 3.5). During painting, each stroke input by the user is processed individually. Its shape and texture are synthesized in isolation from a single stroke library (Section 3.6). Then, if the stroke covers paint already on the canvas, a smear effect is synthesized from a smear library and applied to the stroke (Section 3.7). When smudging, no new stroke is being generated, so a smudge effect is synthesized from a smudge library and directly applied to the paint on the canvas.

Preprocessing (§3.5)

Single Stroke Synthesis:

- Find optimal sequence of library samples (§3.6.1)
- choice {
  - Alpha blending (§3.6.2)
  - Texture synthesis (§3.6.3)

Stroke Interaction Synthesis:

- Detect overlap regions in query stroke (§3.7.1)
- Find closest neighbor example (§3.7.2)
- Vectorize interaction regions (§3.7.3)
- Integrate color in interaction regions (§3.7.4)
- choice {
  - Apply smear operator (§3.7.5)
  - Apply smudge operator (§3.7.6)

Figure 3.4: Algorithm overview.

Algorithm Assumptions. Procedural and simulation based approaches to digital painting encode explicit assumptions about the character of natural media

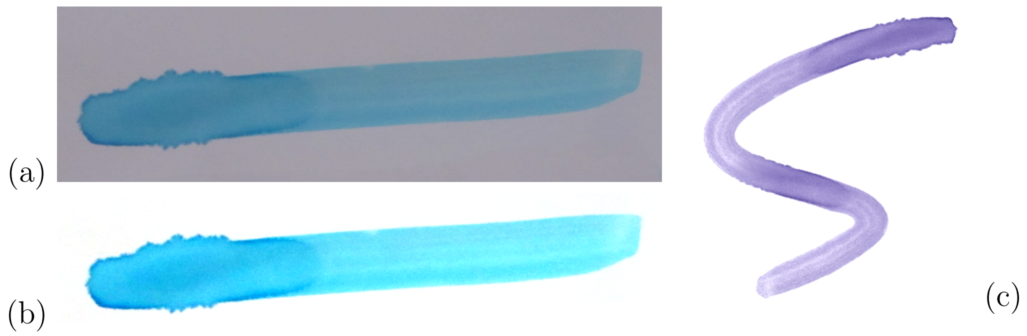


Figure 3.5: Workflow overview. (a) Paint strokes are acquired with a handheld, point-and-shoot camera with indoor lighting. (b) A few simple edits isolate and white balance the stroke. (c) The library stroke is piecewise matched and deformed to the input curve.

into their algorithms. Simulations assume properties of the underlying physical system, while procedural rules assume specific behaviors the media exhibits. Data-driven techniques have the ability to ease these assumptions, but completely removing them may not be possible. We enumerate the basic assumptions we rely on in data acquisition and synthesis.

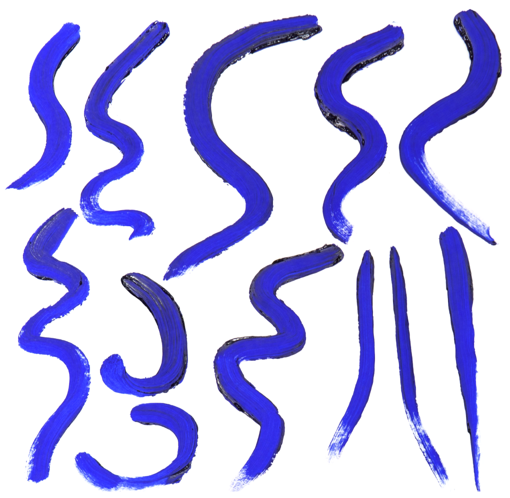
We call our first assumption *stroke causality*—that pixels that have been touched will not be modified after the stroke has passed. This means we explicitly cannot support effects like watercolor backruns, where excess water advects pigment into areas of the stroke that have already been painted.

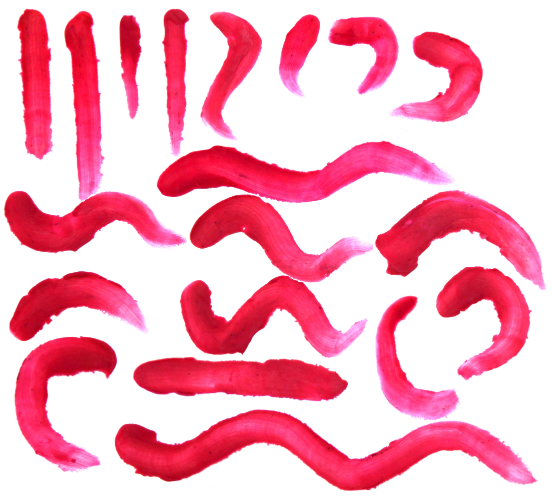
The second assumption is *effect locality*—that a stroke's effect does not extend beyond its silhouette. This means that for effects like feathered watercolor where the pigment has advected away from the brush-canvas contact area, the brush silhouette must be made artificially large to encompass the entire final extent of the stroke.

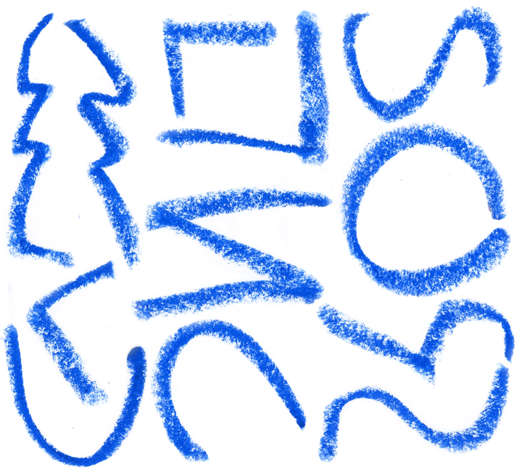
Our final assumption is *recursion*. Given n strokes on a canvas, the $n + 1$ stroke may have interactions that depend on the full, ordered set of previous strokes. Instead, we treat each stroke as interacting with a single, flattened canvas raster, which elides details about occlusions and ordering. This assumption enables efficient stroke interaction synthesis for paintings of thousands of strokes or more.

3.5 Data Collection and Processing

As default options for casual users, we collect libraries of exemplars made by real physical media (Figures 3.2 and 3.6). Advanced users can acquire their own libraries in the same manner. In light of our natural media factorization, we collect separate, different examples for each of shape, smear, and smudge. Our exemplars were captured using a DSLR camera mounted on a tripod under normal in-door lighting without any camera calibration. However, capture a handheld point-and-shoot camera also yields good results (Figure 3.5). We scan the exemplars into raster textures, and semi-automatically


(a) oil


(b) playdoh


(c) pastel

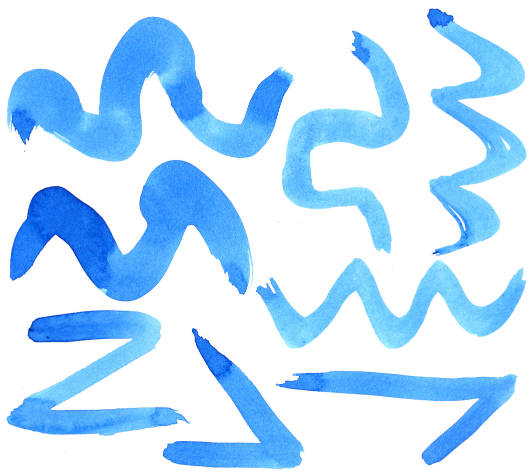
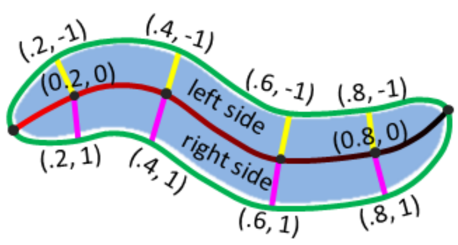

(d) watercolor

Figure 3.6: Single stroke libraries.

process them to create libraries. Single stroke (shape) exemplars are discussed in Section 3.5.1, while smear and smudge exemplars are in Section 3.5.2.

3.5.1 Single Stroke Processing



For each medium, we collect isolated brush strokes of different shape, curvature and thickness to build a single stroke library, $\mathbb{L}$, typically between 20 and 30 strokes. For each stroke, the user specifies a rough spine from start to end (inset, red to black line), which is smoothed and discretized into samples spaced a few pixels apart (inset, six black dots along spine). The stroke outline (green) is automatically extracted plus a small margin to preserve boundary effects. Ribs connecting the spine to the outline are added greedily, avoiding intersections (inset, yellow/magenta lines). Each stroke $\mathbf{L} \in \mathbb{L}$ consists of between 30 and 100 samples, $\mathbf{L} = \{\mathbf{t}\}$, where a sample $\mathbf{t} = \{\mathbf{x}, \mathbf{l}, \mathbf{r}\}$ consists of the 2D positions of the spine, left outline, and right outline points respectively. The stroke is parameterized by $u \in [0, 1]$ along the spine and $v \in [-1, 1]$ from left to right along each rib. We use the Multilevel B-splines Approximation (MBA) library [53] to interpolate u, v coordinates for every pixel in the stroke. The output of this stage consists of the gray-scale intensity image of the exemplar, the vector representation, and the uv parameterization.

3.5.2 Overlapping Stroke Processing

Smearing is the result of transferring pigment from the canvas to the brush during a stroke, causing a mixing of colors (see Figure 3.3). Different physical media properties cause unique smearing behaviors. For example, thick wet oil paint smears significantly, pastels smear somewhat, and dry watercolors do not smear at all. Unconventional media such as plasticine smear differently without mixing pigments but still smudging them along. We use the term smearing for all stroke interactions containing overlapping strokes of different colors.

Artists can also smudge strokes on the canvas without applying additional pigment, commonly with a finger or clean brush. Such smudging exemplars contain two strokes of the same color—a background, and a smudged foreground.

For smearing and smudging separately, we collect between 10 and 20 pairs of overlapping strokes in several different media. For smearing, we collected oil, plasticine, and pastel. For smudging, we collected oil, plasticine, lipstick, charcoal, and pencil. Each pair of strokes overlap at different crossing angles and with different relative thickness. We only collect pairs of strokes interacting—our synthesis algorithm can plausibly reproduce the interactions among many strokes from this data. To process the scanned overlapping strokes, the user needs to specify the stroke spines (in our system) and create binary foreground and background masks, $\mathbf{F}$ and $\mathbf{B}$ (with Photoshop threshold and brush tool). The system then automatically vectorizes the foreground and background strokes (Section 3.5.1), and extracts information specific to smearing and smudging. Though the exemplars shown here are near-orthogonal crossings, our synthesis algorithm can handle all possible overlap cases.

Interaction Region Vectorization: As input for the on-line synthesis of smearing or smudging, we extract binary masks for the overlap region $\mathbf{\Omega}$ and the trailing region $\mathbf{\Psi}$ (green and yellow regions in Figure 3.3a) by doing logic operations on the foreground and the background stroke masks. Specifically, $\mathbf{\Omega} = \mathbf{F} \cap \mathbf{B}$, where $\mathbf{F}$ and $\mathbf{B}$ are the foreground and background stroke masks. Let $\mathbf{u}(\mathbf{\Omega})$ define the average u coordinate of a connected region, then $\mathbf{\Psi} = \{\mathbf{A} \mid \mathbf{A} \subseteq \mathbf{F} \setminus \mathbf{\Omega}, \mathbf{u}(\mathbf{A}) > \mathbf{u}(\mathbf{\Omega})\}$ We refer to interaction region $\mathbf{\Pi} = \mathbf{\Omega} \cup \mathbf{\Psi}$ as the union of the overlap and the trailing regions. We then extract the contours of the interaction region as eight polylines (colored line segments in Figure 3.3a) which are used for synthesis (Section 3.7.3).

Smearing Operator: In the interaction region, the background blue pigments get mixed with the foreground red pigments resulting in vertical streaks of brownish color along the stroking direction (Figure 3.3b,c). Since the physical smearing process is more or less independent of the specific colors being mixed, it is safe to require the foreground to be red and background to be blue and obtain the mixing behaviors that are universal for paints of any colors. For each exemplar, we extract a *smearing operator* that is a 1 channel, 2D texture indicating the amount of background paint (from 0 to 100 percent) at each pixel. Since blue and red are more or less orthogonal after capturing, we simply use one minus the blue channel (lower right corner of Figure 3.3b,c). At runtime, we use the smearing operator to synthesize the smearing appearance of arbitrary colors (Section 3.7.5).

Smudging Operator: The foreground smudged strokes consist of advected pigment from the background strokes (Figure 3.3d,e). We extract a *smudging operator* (the lower right corners of Figure 3.3d,e) as the inverse intensity indicating the amount of background pigment at every pixel. In addition, we also extract a blending attenuation mask automatically by blurring the left, right, and upper side of the overlap region binary mask $\mathbf{\Omega}$. For better synthesis quality, a more precise blending attenuation mask can be provided by the user using standard image editing tools. The blending attenuation mask is used in the synthesis to provide gradual transition at the overlap region boundaries (Section 3.7.6).

Output. After processing is complete, the following information is passed to the on-line synthesis stage. Each of the smearing and smudging exemplars consist of the vectorization of both the foreground and background strokes and the vector representation of the interaction region. In addition, the smearing exemplars contain the smearing operator. The smudging exemplar contains the smudging operator and the blending attenuation mask. Optionally, the user can also provide as input the background strokes themselves before being smeared or smudged by the foreground strokes, which can facilitate the exemplar matching process (Section 3.7.2).

3.6 Single Stroke Synthesis

ealBrush starts by generating the appearance of a single, isolated stroke. Given an input query spine (a curve-like spline), we first calculate the instantaneous stroke thickness at every path sample (proportional to the pressure values or inversely proportional to the stroking velocity). We sweep line segments of the estimated length along and perpendicular to the stroke path to obtain a rough 2D shape and u, v parameterization (see Section 3.5.1). To fill in the pixel colors within the 2D region, we apply a piecewise matching algorithm to find segments of library exemplars that have similar shape with the query (Section 3.6.1). Finally, we warp and merge the matched grayscale exemplar segments together (Section 3.6.2) to determine the lightness, $\mathbf{L}$, of the synthesized stroke. Alternatively, for complex structured exemplars, such as sponges and glitters, we can use an off-line texture synthesis step for higher synthesis quality (Section 3.6.3).

At runtime, given a user-specified query color, we convert it to CIELAB space, $(\mathbf{L}_t, \mathbf{a}_t, \mathbf{b}_t)$. Then the synthesized color $\mathbf{C}_f = (\mathbf{L}_f, \mathbf{a}_f, \mathbf{b}_f)$ of each pixel is: $\mathbf{L}_f = \mathbf{L}$, $\mathbf{a}_f = \alpha \mathbf{a}_t$ and $\mathbf{b}_f = \alpha \mathbf{b}_t$, where $\alpha = (100 - \mathbf{L})/100$. The synthesis outputs are a binary query mask $\mathbf{Q}$ indicating the regions of the canvas that belong to the query, the u, v parameterization, and the colors $\mathbf{C}_f$ that we call foreground color for the rest of this chapter.

3.6.1 Piecewise Matching

We modify the algorithm introduced in Section 2.5 to find an optimal sequence of similar segments from the shape library. The following sections highlight the differences in the design of feature vector and optimization.

Feature Vector

The feature vector (similar to Ando and Tsuruno [3]) contains a *shape feature* $\mathbf{S}_i$ using samples of path turning angle (reflects the changes in hand-pose) and the instantaneous thickness (implies pressure). To preserve the unique appearance at the beginning and end parts of the exemplar strokes, the feature vector also contains an *end feature* $\mathbf{E}_i$ (same as Section 2.5.1), which encodes the closeness of the current sample to the endpoints.

For the current query sample i, the shape feature is defined as: $\mathbf{S}_i = \{\mathbf{f}_j \mid n \geq |i - j|\}$. The number of shape samples $\mathbf{f}_j$ is $2n+1$, n in the history and n in the future. We obtain each shape sample by applying triangle filtering (described in Section 2.5.1) to the turning angles and the thickness values in the local neighborhood. The instantaneous turning angle θ_k and stroke thickness λ_k are defined as follows:

$$\theta_k = \mathrm{sgn}_k \arccos \frac{\overrightarrow{\mathbf{v}_{k+1}} \cdot \overrightarrow{\mathbf{v}_k}}{\|\overrightarrow{\mathbf{v}_{k+1}}\| \|\overrightarrow{\mathbf{v}_k}\|} \tag{3.1}$$

$$\overrightarrow{\mathbf{v}_k} = \mathbf{x}_k - \mathbf{x}_{k-1}, \overrightarrow{\mathbf{v}_0} = \mathbf{0} \tag{3.2}$$

$$\mathrm{sgn}_k = \begin{cases} 1 & \mathbf{x}_{k+1} \textit{ is on the left of } \overrightarrow{\mathbf{v}_k} \\ -1 & \mathbf{x}_{k+1} \textit{ is on the right of } \overrightarrow{\mathbf{v}_k} \end{cases} \tag{3.3}$$

$$\lambda_k = \|\mathbf{x}_k - \mathbf{lx}_k\| + \|\mathbf{x}_k - \mathbf{rx}_k\| \tag{3.4}$$

Each shape sample contains two scalars, $\mathbf{f}_j = \{\mathbf{\Theta}_j, \mathbf{\Lambda}_j\}$. $\mathbf{\Theta}_j$ represents the filtered turning angle, $\mathbf{\Theta}_j = \textit{triangleFilter}(\{\theta_k, |k - j| < \delta\})$. $\mathbf{\Lambda}_j$ represents the filtered thickness, $\mathbf{\Lambda}_j = \textit{triangleFilter}(\{\lambda_k, |k - j| < \delta\})$.

Putting these all together we have feature vector $\mathbf{F}_i = \{\mathbf{S}_i, w\mathbf{E}_i\}$ where w is a weight that balances between the relative importance of shape features versus end features. We use $n = 7$, $w = 0.5$. Therefore, the overall dimension of the feature vector $\mathbf{F}_i$ is $2 \times (2 \times 7 + 1) + 2 = 32$.

Matching Optimization

Once we have computed a feature vector, we look for k nearest exemplar samples for each query sample. Then we use a global optimization framework, similar to Section 2.5.2, to break the query stroke into a few long segments each of which matches a consecutive library stroke segment. We keep the first three energy terms,

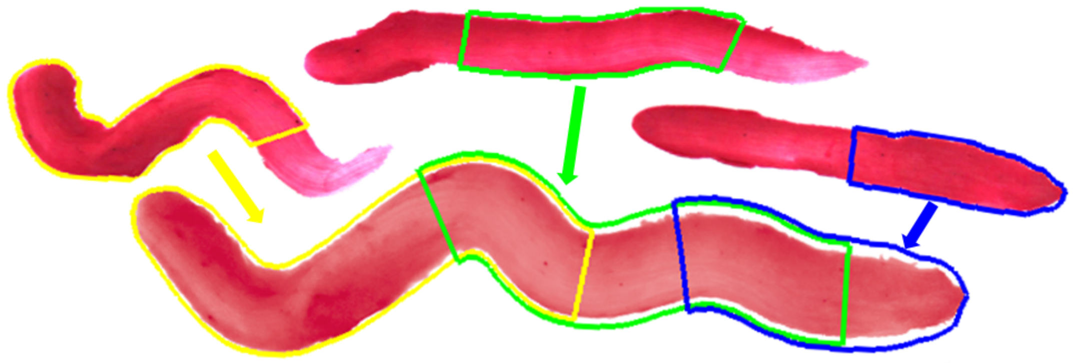


Figure 3.7: Single stroke piecewise matching and warping. The bigger query stroke at the bottom is broken into three overlapping segments each of which matches an exemplar segment. The three segments are highlighted in yellow, green and blue.

e_f (feature penalty), e_s (short penalty), e_m (endpoint penalty), and tailor the design of the transition penalty term e_t to account for textured 2D strokes.

At the query sample i, the transition penalty of going from library samples $\mathbf{c}_i$ to $\mathbf{c}_{i+1}$ include the following terms:

1. *Appearance term:* We sample a 2D region on the intensity texture of the exemplar stroke centered at $\mathbf{c}_i$ and we sample another 2D region with the same size on the exemplar stroke centered at $\mathbf{c}_{i+1}$. We calculate the L2 distance between this two regions. The idea is to prevent jumping from a library exemplar to another which have very different local texture details and intensities.
2. *Thickness term:* We measure the difference in the instantaneous thickness between library samples $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$. The motivation is to find library segments that have similar thickness with each other to avoid sudden changes in texture frequencies in the synthesized stroke appearance.
3. *Curvature term:* If the query sample i has large filtered turning angle (high curvature), we discourage the transition from one library stroke to another to happen here. The reason is to minimize warping and blending artifacts around sharp corners, since blending flat regions of strokes tend to produce less noticeable visual artifacts.

3.6.2 Warping and Merging

The matched segments will not exactly match the shape of the query stroke and so must be warped to fit. This is done by sampling the exemplar textures according to the u, v parameterization computed by the MBA library [53]. Because the segments are already close in shape to the query, warping usually produces a high quality output.

To merge adjacent segments, alpha blending simply linearly interpolates between the two segments in their overlap region, creating a cross-dissolve. For many types of media with low frequency textures such as watercolor and oil paint, this produces high quality results. Alpha blending is easy to implement and efficient, so we use it for our results unless otherwise indicated. Alpha blending can result in two types of artifacts: ghosting and blurring (Figure 3.8a). Ghosting occurs when matching cannot find segments with similar appearance at the boundary, and blurring is caused by interpolating mismatched structured texture. In both cases, graph cuts can be used to find an optimal seam between segments, followed by gradient domain compositing to smooth the transition [78] (Figure 3.8b). In many cases, graph cuts generate a superior result and is still relatively efficient, so we reserve it as an alternative merging option. Figure 3.7 shows the piecewise matching and blending result. The query stroke at the bottom is broken into three overlapping segments each of which matches an exemplar segment (highlighted in yellow, green and blue).

3.6.3 Texture Synthesis

To better deal with highly textured media (Figure 3.8), we have prototyped an optional texture synthesis module which runs off-line to produce higher stroke quality. We adapt image melding [26] to the problem of stroke synthesis. As a review, image melding works through multiscale texture synthesis over small square patches. The "melding" component allows for smooth transitions between different textures without losing detail in the transition region. We use this to transition between different brush exemplars. We also use the "texture preserving warps" which improve the texture distortion introduced by warping.

We apply image melding to an initial guess which is the warped image after graph cut. However, naive melding alone produces inferior visual results. Therefore we add constraints to image melding to better guide synthesis. These include index masks, color corrections, and rotation constraints. The constraints are illustrated in Figure 3.10. The importance of constraints is shown in Figure 3.9, which gives a comparison with naive image melding.

Index masks. We apply index masks to constrain interior regions in the result to be synthesized from interior regions in the exemplars. We define interior regions to be those that are more than a threshold distance $\tau = 1/4$ from the stroke boundary, where the maximum distance is defined to be unity. The interior regions are shown in Figure 3.10, regions 1 and 3. Similarly to image melding, we also apply distance constraints, which prevent matched patches from moving too far from the initial correspondence given by warping.

Color corrections. Image melding can introduce undesired fluctuations in color because of its gradient energies. We thus constrain the color outside the initial guess to be transparent and white. To prevent fluctuations in paint density due to any averaging, at coarse scales after each iteration we use color transfer [108] to match colors to the initial guess. We do this locally by running the color transfer on each of 5x5 sub-windows that are overlapping, and smoothly interpolating the result in the overlapped regions.

Rotation constraints. We constrain rotations to be similar to the principal brush angle. These angles follow the direction of the brush stroke and are shown

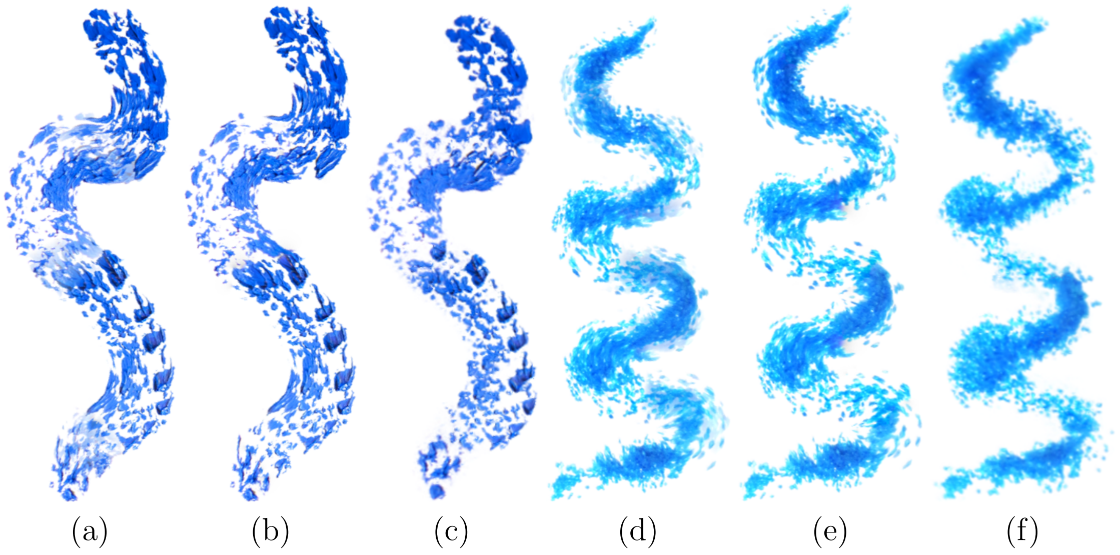


Figure 3.8: Artifacts in synthesis and methods to fix them. While simple alpha blending works for most media, some media may not be able to find well-registered matches and produce ghosting artifacts (a, d). Graph cuts often improves this (b, e). Texture distortion artifacts can also be present for highly textured media. These artifacts can be improved with our off-line texture synthesis module (c, f).

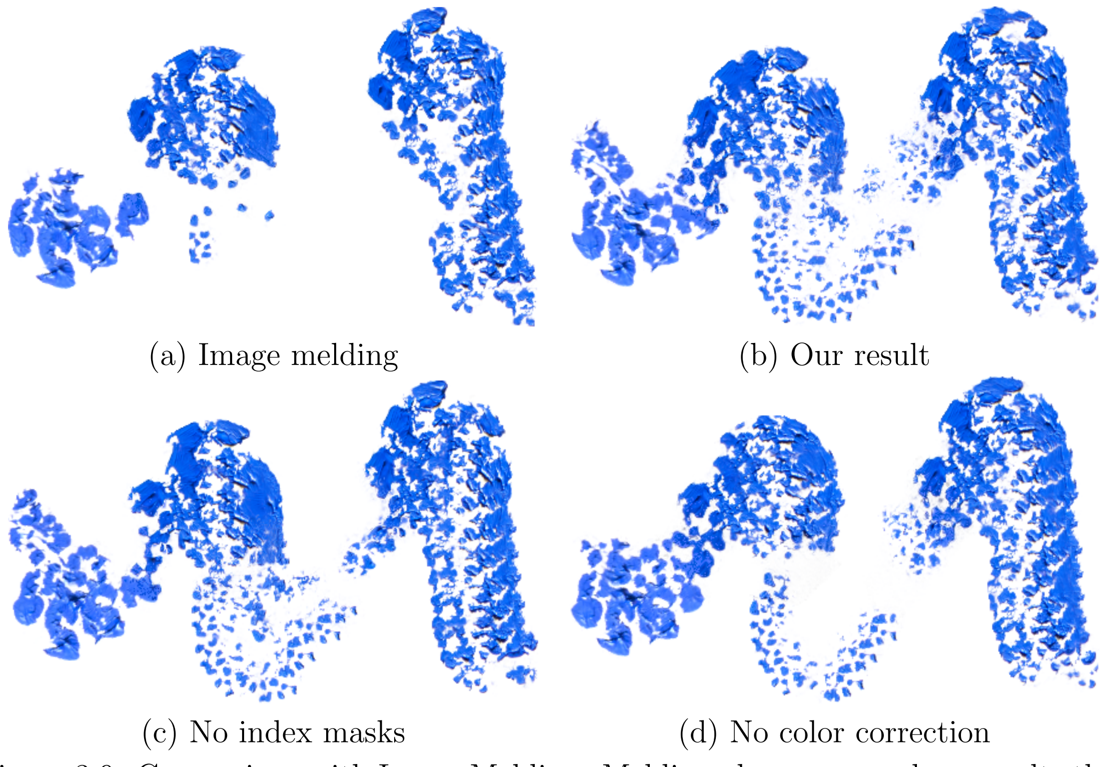


Figure 3.9: Comparison with Image Melding. Melding alone can produce results that do not closely follow the brush stroke (a). Our texture synthesis gives an improved result (b). The effect of removing index masks from our algorithm is shown (c), as well as the effect of removing color correction (d). Without our constraints, the texture can deviate from the brush stroke.

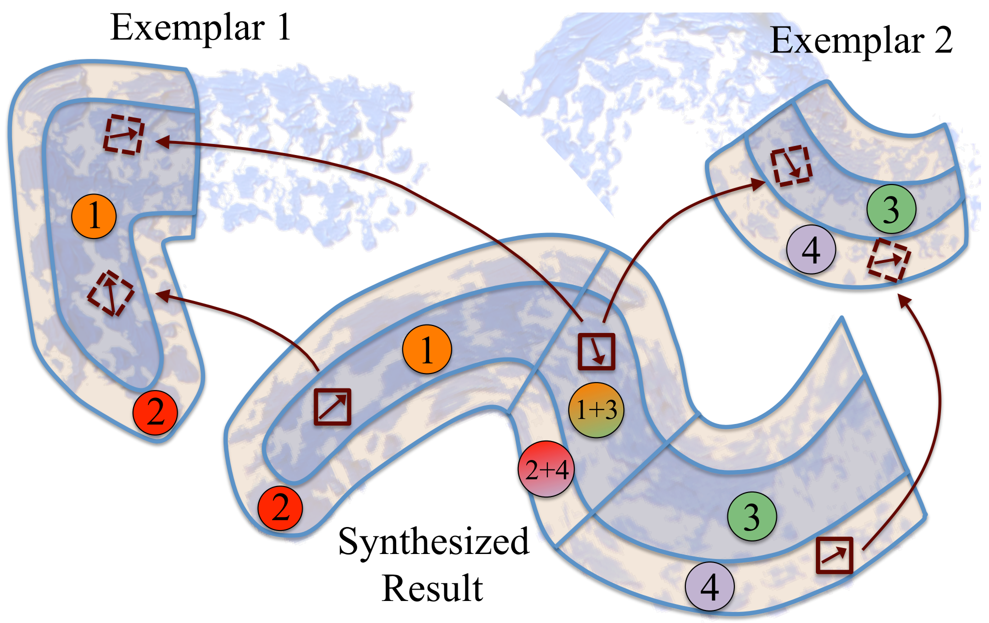


Figure 3.10: Explanation of the optional texture synthesis module. Two exemplars (top) are matched and an initial guess is constructed by warping. Texture synthesis improves any texture distortion. Each upright patch (small square) in the result is matched during multiscale synthesis to an appropriately rotated patch in an exemplar. Rotation constraints orient patches to follow the brush principal angle (indicated by small arrows inside the patches). Distance constraints prevent matched patches from moving too far from the initial correspondence given by warping. Patches are constrained to only come from the same index mask: patches in region 1 match patches in region 1, and so forth. The overlap regions 1 + 3 and 2 + 4 match both exemplars and are melded for a smooth transition.

as small arrows inside the patches in Figure 3.10. For each patch of each warped exemplar, we find a principal angle. We determine for a square patch in the output synthesized image which quadrilateral it corresponds to in the exemplar image. For a square quad each edge vector would have the same angle, after rotation by 0, 90, 180, and 270 degrees. We define the principal angle of a general quad similarly by averaging the four edge vectors after these same rotations. Patches in the synthesized image are always upright, whereas matched patches in the source exemplars rotate. We allow some deviation from the exact brush principal angle by allowing for matches in an angular window. We use a larger angular window of 30 degrees in the interior region, and a smaller angular window of 10 degrees on the boundary.

3.7 Stroke Interaction Synthesis

Single stroke synthesis does not consider what is on the canvas. When strokes overlap, independent synthesis will produce artificial results. When real media overlap, their appearances change predictably. RealBrush reproduces two types of these interactions: smearing and smudging.

We observe that smearing and smudging happen mostly within the interaction region (defined in Section 3.5.2). We factor the interaction effect into three components: color, texture and shape. In most natural media, the impact to the foreground stroke shape is not obvious, so we only handle the change of color and texture. This is a consequence of our natural media factorization—first we perform single stroke synthesis (Section 3.6), and input the appearance into the smearing and smudging algorithms.

When the user paints a new stroke, our system first detects a set of overlap regions where the canvas beneath already has paint (Section 3.7.1). Per overlap region, we match to a smudging or smearing exemplar for synthesis (Section 3.7.2). To warp the exemplar, we vectorize the interaction regions (Section 3.7.3). To determine the colors in the interaction region, we apply weighted color integration (Section 3.7.4). The only difference between smearing and smudging is how we apply the matched exemplar (Section 3.7.5 and Section 3.7.6).

3.7.1 Overlap Regions Extraction

As strokes accumulate on the canvas, a new stroke might overlap many previous strokes creating an exponential number of overlap regions. However, our recursion assumption allows us to treat the canvas as a single, flattened paint raster, which makes the approach efficient and scalable. Specifically, the smearing or smudging behavior of the current time step t is only dependent on the raster canvas image of all strokes at $t-1$, the time of the previous stroke.

At every time step, we update a binary coverage mask $\mathbf{C}_t$ to keep track of whether a pixel on the canvas has paint. After the user paints a new query stroke, the system detects a set of overlap regions $\mathbf{\Omega}_t = \{\mathbf{\Omega}_t^i\}$ by intersecting the query mask $\mathbf{Q}_t$ with the coverage mask: $\mathbf{\Omega}_t = \mathbf{Q}_t \cap \mathbf{C}_t$. Each overlap region $\mathbf{\Omega}_t^i$ is restricted to lie inside the query stroke and forms a connected component in the coverage mask (Figure 3.13a). We break long, irregularly shaped overlap regions into smaller ones by scanning the connected component along the stroke spine to identify cusps and cutting at the cusp perpendicular to the spine. For example, the first two overlap regions in Figure 3.13a originate from a single connected component. We also discard small overlap regions, since their influence on the final painting are minimal. The rest of the overlap regions are classified into several categories (Figure 3.11) for vectorization.

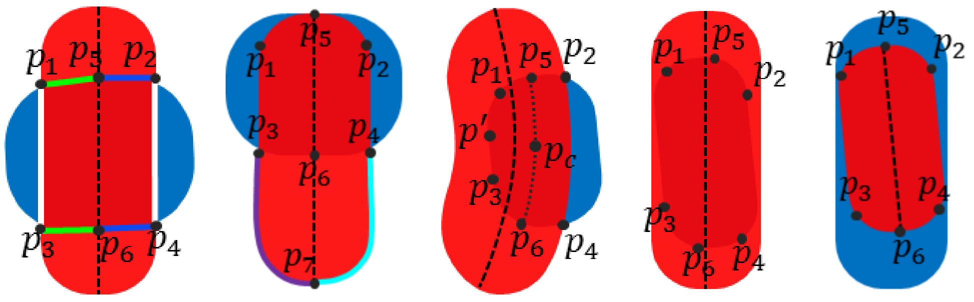

p_1 p_5 p_2 p_3 p_6 p_4
p_1 p_5 p_2 p_3 p_4 p_6 p_7
p_1 p_5 p_2 p' p_c p_3 p_6 p_4
p_1 p_5 p_2 p_3 p_6 p_4
p_5 p_1 p_2 p_3 p_4 p_6
(a)
(b)
(c)
(d)
(e)

Figure 3.11: Stroke overlap situations. (a) - (e) The query stroke (red) might overlap with the background stroke (blue) in several different configurations. The overlap region might (a) run across the query stroke; (b) be at the beginning or the end of query stroke; (c) not separate the query stroke into disjoint regions; (d) be entirely inside the query stroke; (e) be the entire query stroke; The colored line segments in (a) indicate the overlap region vectorization. In (b), they indicate the trailing region.

3.7.2 Exemplar Matching and Feature Vectors

We synthesize the stroke interactions by warping the captured interaction exemplars. To reduce warping artifacts, we search for exemplar strokes that cross at similar angles and have similar thickness. For simplicity and robustness, we design a raster feature vector to capture information of the overlap region and its surroundings. For each overlap region in the query stroke, the feature vector is a two-channel square texture (50 x 50). The square is oriented along the average foreground stroke spine orientation in the overlap region. It contains the entire overlap region with padding so the surroundings can be captured. The first texture channel contains the alpha matte of the background in the overlap region. The second texture channel contains a mask of whether strokes are present in the surrounding region. The feature vector distance is the L^2 distance between the squares. When providing the exemplars, the user can optionally input photographs of the background strokes before foreground strokes are applied. After manually aligning the "before" and "after" textures, we sample intensities of the "before" overlap region as the first channel of our feature vector. The use of the "before" image allows the system to search for crossing exemplars not only similar in shape, but also in texture and intensity. Without the "before" image, we reduce the feature vector to one channel. The search is not sensitive to the size of the square or the precision of the "before" and "after" alignment. The amount of distortion even in the case of very different exemplars is hardly noticeable in the context of a whole painting. Figure 3.12 shows the query stroke feature vector and the most similar overlapping exemplar.

3.7.3 Interaction Region Vectorization

After identifying the most similar interaction exemplar, we need to warp the matched exemplar to fit the query interaction region exactly for synthesis. Section 3.7.1 identifies the overlap regions in raster format. We then need to vectorize

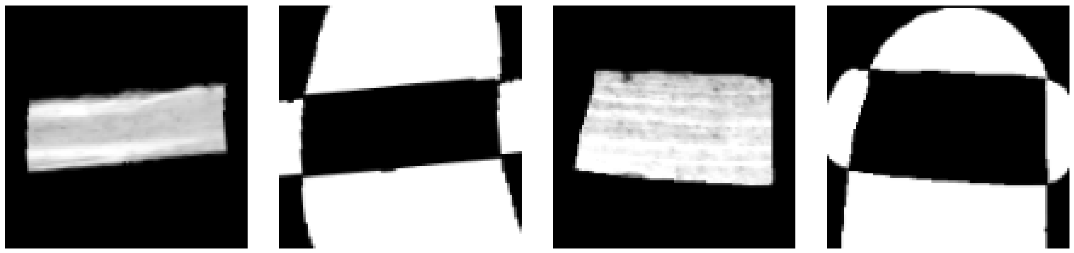
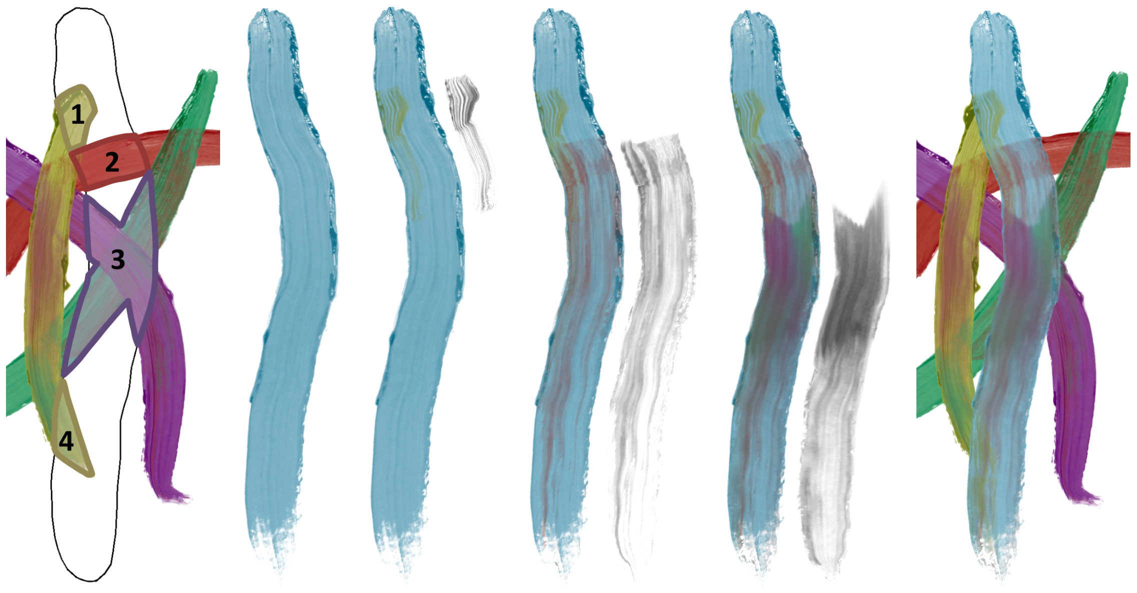


Figure 3.12: Overlap region feature vector. The overlap regions of the query and exemplar strokes are characterized by a squared two-channel texture. The query feature vector contains the alpha matte of the background in channel 1 (a) and a binary mask of surrounding strokes in channel 2 (b). The matching exemplar is shown in (c, d).

the regions and identify the trailing regions. We extract the six polylines (green, blue and white lines in Figure 3.11a) for the overlap regions and two polylines (purple and cyan lines in Figure 3.11b) for the trailing regions, corresponding to that of the exemplar (Figure 3.3a).

From the overlap region mask, we extract the overlap contour using OpenCV. We scan the overlap contour to identify six "key points" that divide the contour into six polylines. For overlap regions that cross the query stroke (Figure 3.11a), we first intersect the stroke spine with the contour to identify $\mathbf{p}_5$ and $\mathbf{p}_6$. We then locate the $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$ by tracing from $\mathbf{p}_5$ and $\mathbf{p}_6$ clockwise and counterclockwise along the overlap contour until reaching the query stroke boundary. When the overlap region is at the beginning (or end) of the query stroke (Figure 3.11b), we identify $\mathbf{p}_5$ to be the first sample on the query spine and apply an offset from $\mathbf{p}_5$ to locate $\mathbf{p}_1$ and $\mathbf{p}_2$ on the query contour. The offset is determined by the average thickness of the query stroke. Then $\mathbf{p}_3$, $\mathbf{p}_4$, $\mathbf{p}_6$ are extracted the same as case (a). There are also cases where the overlap region does not separate the query stroke into disjoint segments (Figure 3.11c). In this case, we find the overlap contour sample $\mathbf{p}'$ with smallest v coordinate. We offset from $\mathbf{p}'$ to obtain $\mathbf{p}_1$ and $\mathbf{p}_3$ on the overlap contour and use the center of mass $\mathbf{p}_c$ to locate $\mathbf{p}_5$ and $\mathbf{p}_6$ following the stroke spine direction. Then $\mathbf{p}_2$ and $\mathbf{p}_4$ are traced from $\mathbf{p}_5$ and $\mathbf{p}_6$, the same as in case (a). When the entire overlap region is within the query stroke (Figure 3.11d), none of the points can be located exactly, so we identify points with maximum and minimum u coordinates as $\mathbf{p}_5$ and $\mathbf{p}_6$ and use the offset idea to find the other points. In the final case where the entire query stroke is inside a background stroke (Figure 3.11e), $\mathbf{p}_5$ and $\mathbf{p}_6$ are the start and end of the stroke spine, and we apply offsets to locate the other points. In real painting scenario, the case in Figure 3.11c happens frequently resulting in many thin interaction regions. To avoid aliasing and ensure the stableness of the points $\mathbf{p}_i$, we discard overlap regions that are smaller than some threshold. For larger interaction regions, experiments show that the detection of points $\mathbf{p}_i$ is very robust to all possible overlap shapes and angles.



(a) (b) (c) (d) (e) (f)

Figure 3.13: Stroke interaction synthesis illustration. (a) The query stroke overlaps with the background in four overlap regions; (b) The single stroke synthesis result without smearing; (c) (d) (e) The background colors in the overlap regions are smeared into the subsequent query stroke regions. The warped smearing operators are shown on the right of the query stroke; (f) The four overlap regions are synthesized, alpha composited and rendered in the context of the background.

To finish vectorizing the interaction region $\mathbf{\Pi}_t^i$, we also detect and vectorize a trailing region $\mathbf{\Psi}_t^i$ that follows the overlap region $\mathbf{\Omega}_t^i$. The trailing region length is determined by the ratio of trailing region length to overlap region length in the exemplar, $L(\mathbf{\Psi}_{query}) = L(\mathbf{\Omega}_{query})L(\mathbf{\Psi}_{exemplar})/L(\mathbf{\Omega}_{exemplar})$, where $L(\mathbf{\Omega}) = (\mathrm{d}u(\mathbf{p}_1, \mathbf{p}_3) + \mathrm{d}u(\mathbf{p}_5, \mathbf{p}_6) + \mathrm{d}u(\mathbf{p}_2, \mathbf{p}_4))/3$ and $L(\mathbf{\Psi}) = (\mathrm{d}u(\mathbf{p}_7, \mathbf{p}_3) + \mathrm{d}u(\mathbf{p}_7, \mathbf{p}_6) + \mathrm{d}u(\mathbf{p}_7, \mathbf{p}_4))/3$. $\mathrm{d}u(\mathbf{p}_1, \mathbf{p}_2)$ defines the absolute difference of u coordinates between two points. We locate the point $\mathbf{p}_7$ by tracing from point $\mathbf{p}_6$ along the stroke spine direction. Then, we trace two polylines from $\mathbf{p}_3$ and $\mathbf{p}_4$ following the stroke spine direction and curve in to reach $\mathbf{p}_7$ (Figure 3.11b). The trailing region is constrained to lie within the query stroke boundary. When an overlap region is at the end of the stroke, there is no trailing region.

3.7.4 Interaction Region Color Integration

For smearing or smudging, in each interaction region, the background material colors are picked up by the brush or the finger and smeared into the succeeding foreground stroke regions. The synthesized query stroke color might have contributions from multiple background strokes. For each pixel in the interaction region, we find the background color $\mathbf{C}_b$ by integrating the colors from the pixels on and above (with smaller u coordinates) the current pixel $\mathbf{p}_c$ along the stroke spine direction. The set of the background pixels that contribute to the color of the current pixel

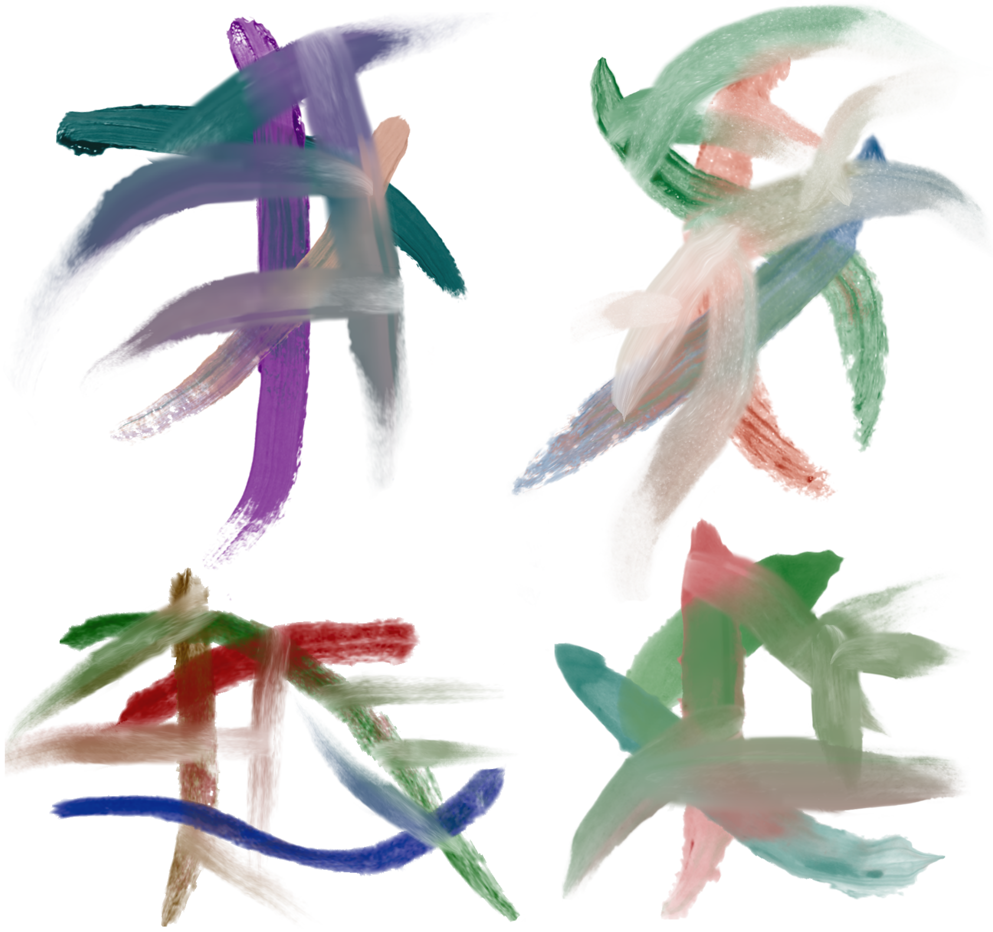


Figure 3.14: Smearing and smudging synthesis results. From left to right, the first row synthesizes the oil and the lipstick media. The second row synthesizes the charcoal and the plasticine media. A few strokes are painted in the background with the smearing effect and then are smudged by a few strokes in the foreground.

is $\mathbf{S} = \{\mathbf{p}_i \mid u(\mathbf{p}_i) < u(\mathbf{p}_c), v(\mathbf{p}_i) = v(\mathbf{p}_c), \mathbf{p}_i \text{ has paint}\}$. Then the integrated background color is calculated as $\mathbf{C}_b = \sum \mathbf{w}_i \mathbf{c}_i / \sum \mathbf{w_i}, \forall \mathbf{p}_i \in \mathbf{S}$, where $\mathbf{c}_i$ is the color of the background pixel $\mathbf{p}_i$ and $\mathbf{w}_i = 1/(\mathrm{d}u(\mathbf{p}_i, \mathbf{p}_c) + \epsilon), \epsilon = 0.06$. We exclude background pixels that have no paint to contribute.

3.7.5 Smearing Effect Synthesis

As the outcomes of the Single Stroke Rendering pipeline, each pixel of the query stroke has a foreground color $\mathbf{C}_f$. To simulate smearing, at every pixel, some amount of the background color $\mathbf{C}_b$ (defined in Section 3.7.4) is mixed with the foreground color $\mathbf{C}_f$ (defined in Section 3.6) to create paint streaking effect. We use the matched smearing operator to estimate the color mixing proportion α at every pixel. Then the synthesized color of each query pixel is a blending of the foreground and background colors, $\mathbf{C}_p = \alpha \mathbf{C}_b + (1 - \alpha)\mathbf{C}_f$. Specifically, we determine the α for every pixel by warping the smearing operator. We specify control points on the detected polylines (Figure 3.11a,b) and apply interpolation for all other pixels.

Note that each query stroke can have several overlapping interaction regions. We sort the interaction regions by ascending u coordinates (4 interaction regions in Figure 3.13a). We synthesize them in order by using color integration, followed by alpha compositing. Let $\mathbf{C}_b^i$ represent the integrated background color for the interaction region i. The smearing result of multiple interaction regions is therefore a recursive alpha compositing of the integrated background color $\mathbf{C}_b^i$ onto the query stroke (Figure 3.13c-e). $\mathbf{C}^n = \alpha\mathbf{C}_b^n + (1-\alpha)\mathbf{C}^{n-1}$, where $\mathbf{C}^0 = \mathbf{C}_f$ and n is the number of interaction regions. Finally, the updated query stroke colors $\mathbf{C}^n$ are alpha composited onto the canvas (Figure 3.13f).

3.7.6 Smudging Effect Synthesis

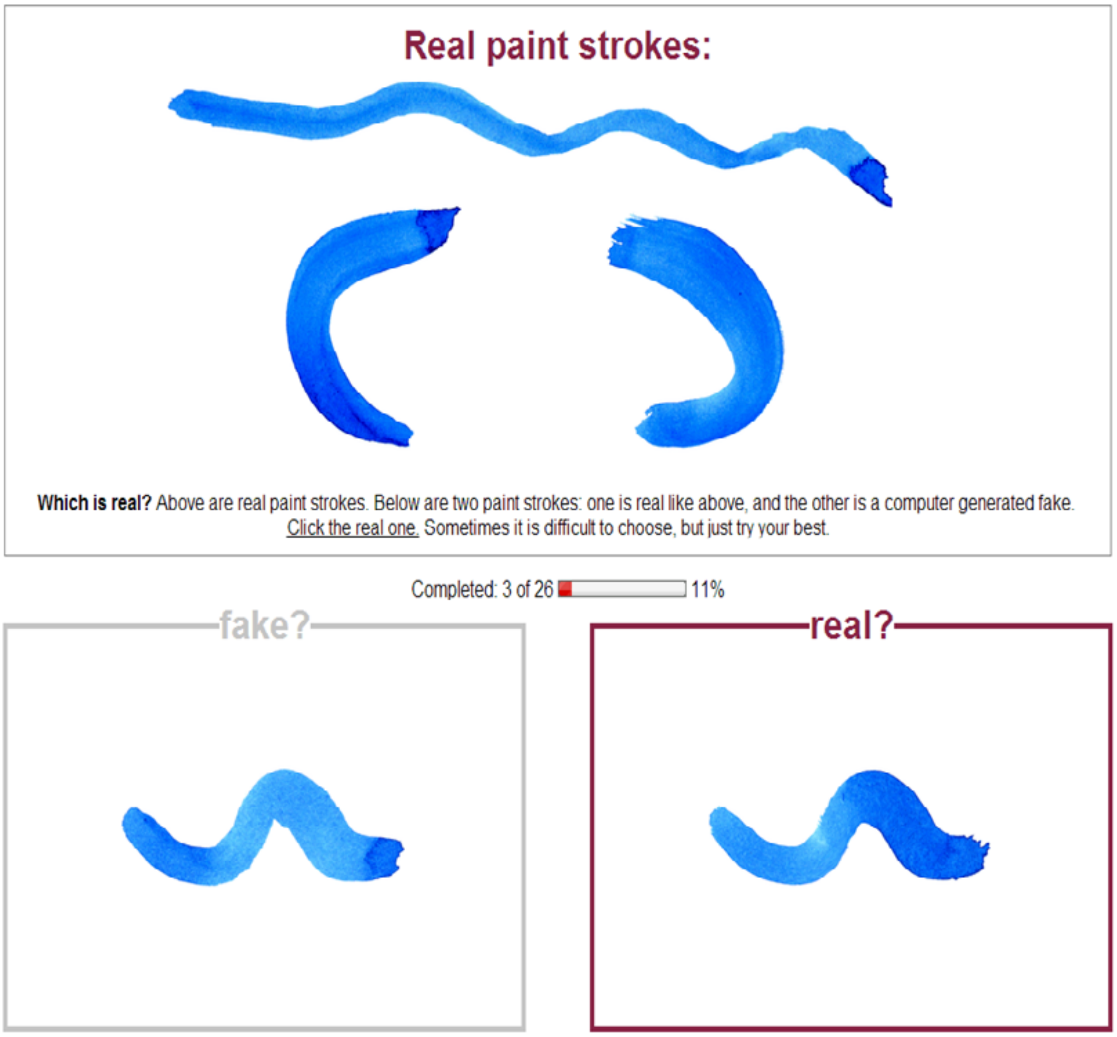
Smudging proceeds similarly to smearing, except that the foreground query stroke is transparent, so the single stroke rendering pipeline is skipped. Instead, the smudging operator intensities and the integrated background colors $\mathbf{C}_b$ directly determine the smudged colors: the alpha component is taken from the warped exemplar, while the color is $\mathbf{C}_b$. A warped blending attenuation mask is used to decrease alpha at the boundary of the query stroke to avoid boundary artifacts. Finally, the smudged colors are alpha composited onto the canvas. Figure 3.14 shows a few stroke scribbles demonstrating the smearing and smudging effects. From left to right, the first row is synthesized based on the oil and the lipstick exemplars. The second row uses the pastel and the plasticine exemplars.

3.8 Evaluation

Here, we describe three user studies we conducted to determine how faithfully our methods reproduce acquired media. Experts can accurately identify our results as computer-generated given sufficient time, so we focus on casual viewers. Talented artists can always work around a system's limitations to make realistic digital paintings by carefully applying many strokes. The resulting images can often fool casual viewers, so evaluating finished paintings is unlikely to yield meaningful data. A stricter standard is to evaluate each algorithm step in isolation. This design controls factors impacting viewers' opinions but outside the scope of the thesis, such as stroke trajectories and subjective color preferences.

3.8.1 Study Design

We employ the same basic methodology in all three studies, adapted from Section 2.7.2, which shared a similar overall goal. The idea is to show both real strokes selected from a library, and synthetic strokes based on the same library, asking subjects which ones are real (using a two-alternative forced choice design). If the



Which is real? Above are real paint strokes. Below are two paint strokes: one is real like above, and the other is a computer generated fake. Click the real one. Sometimes it is difficult to choose, but just try your best.

Figure 3.15: User study interface.

synthetic strokes effectively match the library, the subject will be unable to differentiate them and will choose randomly, while a less successful synthesis approach may be recognized as fake, and the subject will correctly identify the real stroke more often than 50% of the time. To avoid frustrating the participants, we chose the two media (watercolor and oil) that most people are familiar with and can reliably identify as computer-generated without undue effort. We first describe the study for single stroke synthesis, and after, describe the differences for the other two studies.

Study 1: Single-stroke.

Subjects are shown an image (the *guide*) with three photographic examples of real paint strokes in either oil or watercolor. Below it appear two test images, one containing a photograph of a real paint stroke (*ground truth*), and the other showing a (*synthetic*) stroke generated to match the approximate shape of the ground truth stroke. The subject is told that the guide image contains real strokes and asked to pick which of the test images below it is also real (as opposed to a computer-generated

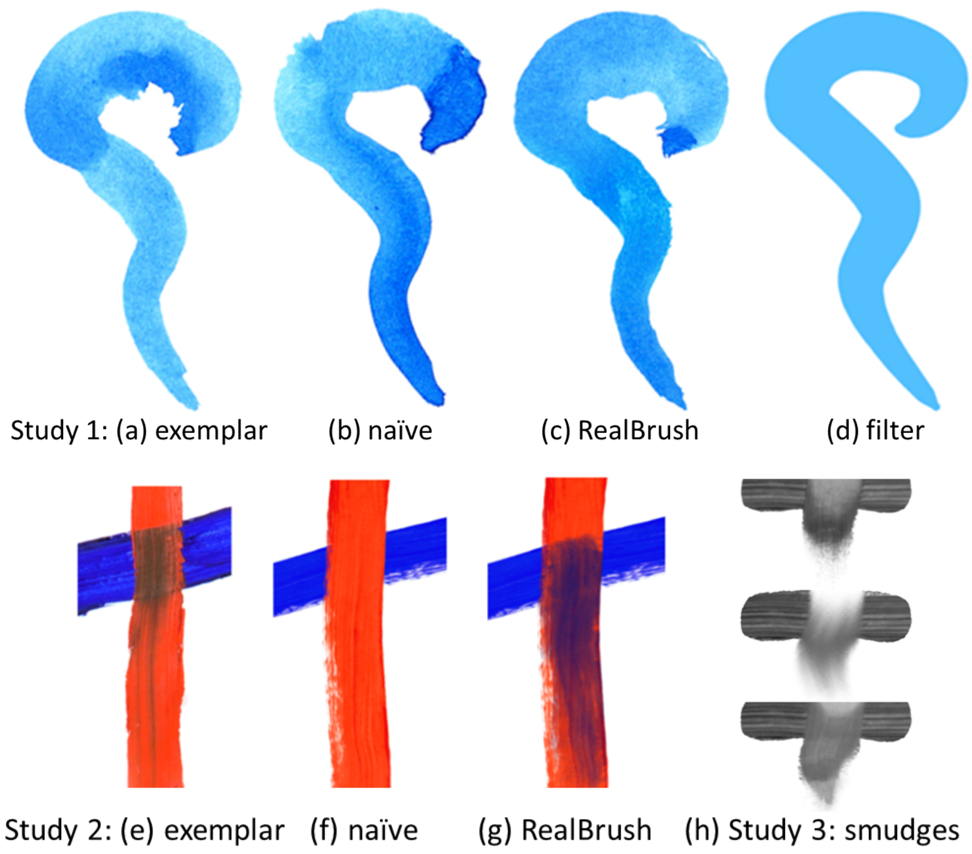


Figure 3.16: Example images from the user study.

fake). When the subject clicks on one of the two test images (believed to be real) they are replaced with a new pair, but the guide image remains unchanged. A progress bar indicates advancement through a series of 26 test pairs until the task is complete. Figure 3.15 shows the design of the interface.

Of the 26 synthetic images, eight are rendered using the ("*RealBrush*") method of Section 3.6 (Figure 3.16c), eight are rendered using a *"naive"* method (Figure 3.16b), and ten are obvious *"filter"* images (Figure 3.16d) used to ensure the subject understands the task and is trying. Each test pair is randomly swapped (left-right); a random flip (horizontal, vertical, both, or neither) is applied to both images; and the 26 pairs appear in random order. The eight strokes selected for the RealBrush condition are chosen randomly without repeats to match a pool of 17 (for oil) or 19 (for watercolor) possible ground truth paths. Likewise for the naive and filter conditions. Thus, even though synthetic strokes may only appear once during a task, a particular ground truth path may possibly be seen by the subject as many as three times. However, we find in such cases it is difficult to recognize because of the random flipping and swapping in a long sequence of comparisons (and even if so, it might only weaken our statistical findings).

The strokes in the RealBrush condition follow the paths of their ground truth comparators, and are rendered using a library of 30 strokes, some of which may be used as ground truth in other tests but are prevented from synthesizing their own paths in a hold-one-out scheme. The library strokes are, however, disjoint from the guide image strokes shown to the subject above the test pair. In the naive method, each test stretches a single library example chosen randomly from a set of ten relatively straight library strokes, rather than fitting multiple parts based on shape as in the RealBrush condition. The strokes in the filter condition, whose goal is to be easily recognized, have constant color over the entire stroke, roughly matched to the ground truth. The strokes are all rendered in blue to prevent color preferences from affecting the results.

Study 2: Smearing. The second study attempts to evaluate the effectiveness of using smearing operator to approximate paint streaking behavior in oil media. To avoid the influence of the color choice, we show crossing strokes using blue as background and red as foreground consistently as with all of our exemplars (Figure 3.16e-g). The ground truth pool contained 10 examples, and the corresponding RealBrush conditions were drawn in our application to roughly match the crossing shape of the ground truth, using 9 possible exemplars in a hold-one-out scheme. The instructions described, "paint strokes crossing over each other." In other respects the study design was as in the first study above.

Study 3: Smudging. In the third study, the guide image showed five smudges of blue paint. However, the test images were shown in black-and-white because the color model used in our system is insufficient to capture the subtlety of this effect well enough to avoid detection when compared with real smudges (Figure 3.19d,e). Using grayscale allows us to factor out the color influence and only evaluate the effectiveness of the smudging operator. Accordingly the instructions are modified describing "black-and-white photos below." The RealBrush condition smudges are based on a library of 10 exemplars, whereas the naive approach uses the "smudge" tool in Adobe Photoshop. (see Figure 3.16h.)

3.8.2 Study Results

Our subjects were recruited on Amazon's Mechanical Turk, and restricted to US-only "workers" who were paid \$0.20 per task ("HIT"), which they typically completed in a few minutes. In the first study, a single worker could perform as many as five HITs for oil and five for watercolor, while in the second and third studies workers could perform up to five HITs per study. Most workers did just one HIT in any study, and in total 139 unique subjects participated. For each of the three studies 100 HITs were posted, from which, respectively, data from 92, 86 and 95 were retained after omitting HITs from workers who failed to correctly identify the 10 filter images.

Table 3.1 shows the results of these studies. In each, a Bonferroni corrected, randomized permutation test on the distributions shows that the naive approach and the RealBrush methods perform differently with statistical significance ($p \ll 0.01$). In the case of single strokes the RealBrush method appears to have been

	correct / count (%)	
study	naive method	RealBrush
1: single stroke	462 / 736 (63%)	360 / 736 (49%)
2: smearing	703 / 760 (93%)	487 / 760 (64%)
3: smudging	673 / 688 (98%)	413 / 688 (60%)

Table 3.1: Results for three user studies. Subjects were asked to identify real vs. synthetic strokes. In one condition the synthesis method was naive, while in the other condition strokes were generated by methods in Sections 3.6–3.7. Results are reported as ratios: pairs correctly identified over pairs shown. If the synthesis method effectively matches the medium, subjects will be forced to guess, leading to an expected 50% ratio. On the other hand, less effective approaches will be correctly identified more often, leading to higher ratios. In each study the RealBrush condition is more effective than the naive approach, with statistical significance.

indistinguishable from real photographs. While less effective, the smearing and smudging methods still perform well and clearly outperform the naive methods.

3.9 Results and Discussion

We gave our prototype painting application to artists to experiment with the media we have acquired. Figure 3.20, 3.21, 3.22, 3.23 show a selection of paintings, which exhibit a wide range of styles and effects. The flowers and ocean sunset use plasticine for single strokes, smearing, and smudging, resulting in smooth textures. The tearful face and the woman in curly hair are made of many transparent watercolor strokes. The young girl's face and the car consist of a combination of oil and watercolor strokes creating abstract contemporary style. The colorful tiger uses oil paint and the line drawing tiger is synthesized with heavy gel exemplars resulting in distinct artistic styles. The feathers of the bird explores the smudging tool to significant effect. The line drawing of the dancer demonstrates the synthesis quality for long curvy strokes. The train and landscape mix different natural media, including oil, watercolor, pencil, and lipstick, which underscores the utility of a general data-driven pipeline supporting a diverse range of media. Figure 3.17 shows that our smudging tool can be used to manipulate photographs for a painterly effect.

Figure 3.18 shows stroke synthesis results from Microsoft Fresh Paint, a commercial oil paint application based on state of the art research [9, 19], for comparison with Figure 3.14. We achieve comparable visual quality, while our data-driven approach produces richer textures and organic appearance. An important advantage of our approach is our support of a broader range of media. Conversely, our synthesized appearance is harder to predict or control—a small change in the stroke path might lead to different matched exemplars with disparate appearances. However, this is arguably similar to the physical painting process where strokes can have unpredictable appearances, especially for novices. Additionally, since our exemplar

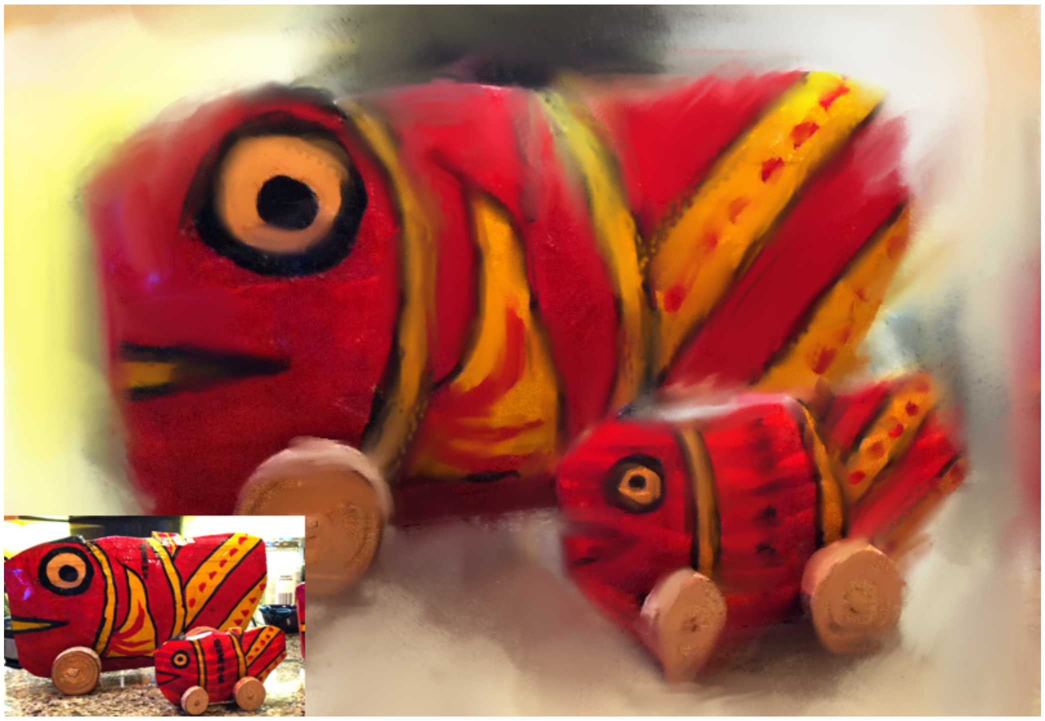


Figure 3.17: An additional benefit of our recursion assumption (Section 3.3) is manipulating photographs. Here, lipstick smudges were used for a painterly effect. The original photo is inset.

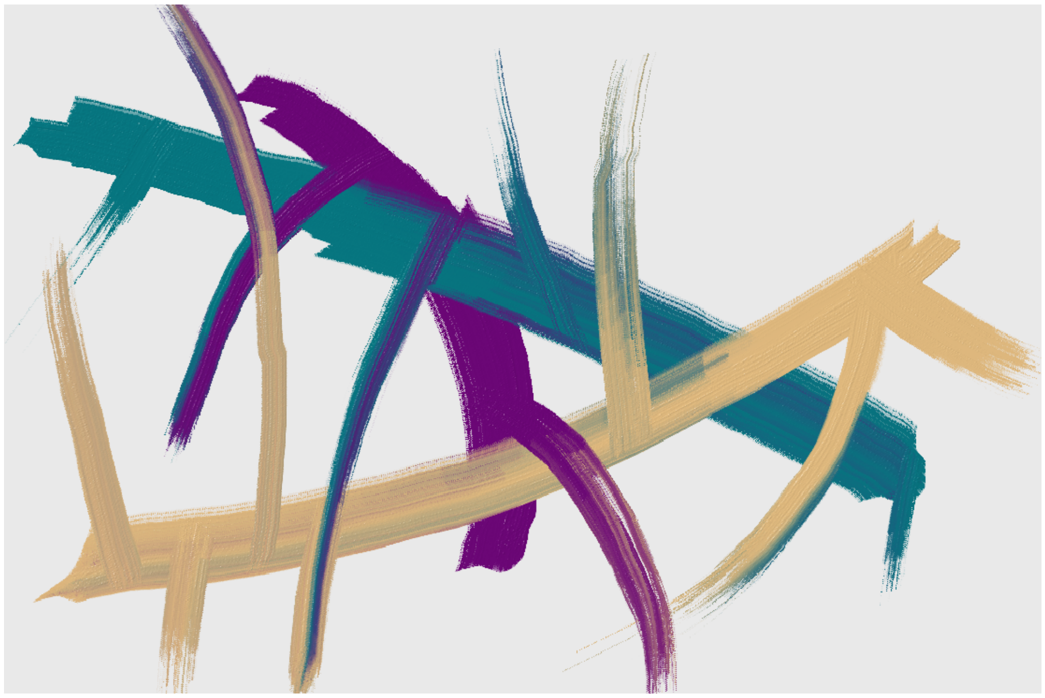


Figure 3.18: Test strokes in Microsoft Fresh Paint [9, 19], generating oil strokes, smeared colors, and smudging with a dry brush, for comparison with Figure 3.14.

strokes encode the experts' techniques, we often found that poorly made query paths result in aesthetically pleasing synthesized strokes.

For a 1000x1000 canvas, synthesizing one stroke takes from 0.5 to 1 seconds, depending on the stroke length and library size. Due to the rotation invariant nature of the matching feature vector, roughly 20 strokes with different thickness and curvature profiles are enough to avoid noticeable repetitions of the same exemplar segments. Though increasing the library size diversifies the synthesis appearance, we do not find significant perceptual quality improvement. The matching performance scales sub-linearly with the library size depending on the nearest neighbor search package. For our chosen canvas size, the performance bottom-neck lies in the image manipulations for stroke vectorization, color integration, and exemplar warping. For synthesizing a query stroke, the performance is constant in the number of existing strokes on the canvas, but heavily depends on the canvas resolution. On the contrary, a vector-based approach that does not flatten the canvas might perform badly as the stroke count goes up, which often happen in real painting scenarios. All our pixel manipulation is implemented in unoptimized C++ on a single core, so could be improved significantly. The texture synthesis takes minutes per stroke, but large portions are implemented in MATLAB and may be optimized.

3.10 Limitations and Future Work

The first result of our evaluation is that for common natural media (oil and watercolor), synthesized individual strokes are indistinguishable from real examples to casual viewers. Though not part of the evaluation, other media with similar texture properties such as plasticine, pencil, charcoal, etc. can also be plausibly reproduced. However, under close examination, several limitations reveal. One is that we do not explicitly synthesize lighting effect. During exemplar acquisition, we do not isolate and handle the lighting artifacts, instead, we treat them as part of the media texture. For example, thick media such as toothpaste (see Figure 3.19a) may cast shadows, and wet media such as lip gloss (see Figure 3.19b) may have specular highlights. Since our matching is rotation invariant, synthesized strokes may have inconsistent lighting effects as a result. For our exemplar libraries, we capture the media using overhead lighting to mitigate these artifacts. Another limitation lies in our color replacement model. We only use the lightness of the exemplars to modulate the synthesized strokes, therefore any color differences within the exemplars will be lost. For example, the sparkles in Figure 3.19b are not well reproduced in the result in Figure 3.19c. Texture distortion due to warping is reduced by our piecewise matching technique, but can still be observed in some regions, such as Figure 3.19c and the oil sponge and glitter in Figure 3.8a.

The second result of our evaluation is that synthesized smears and smudges often look plausible, but are not indistinguishable from real examples. We believe this is largely due to the difference in color mixing, which we leave as future work. For smearing, Figure 3.16f,g shows that the synthesized color in the interaction region is

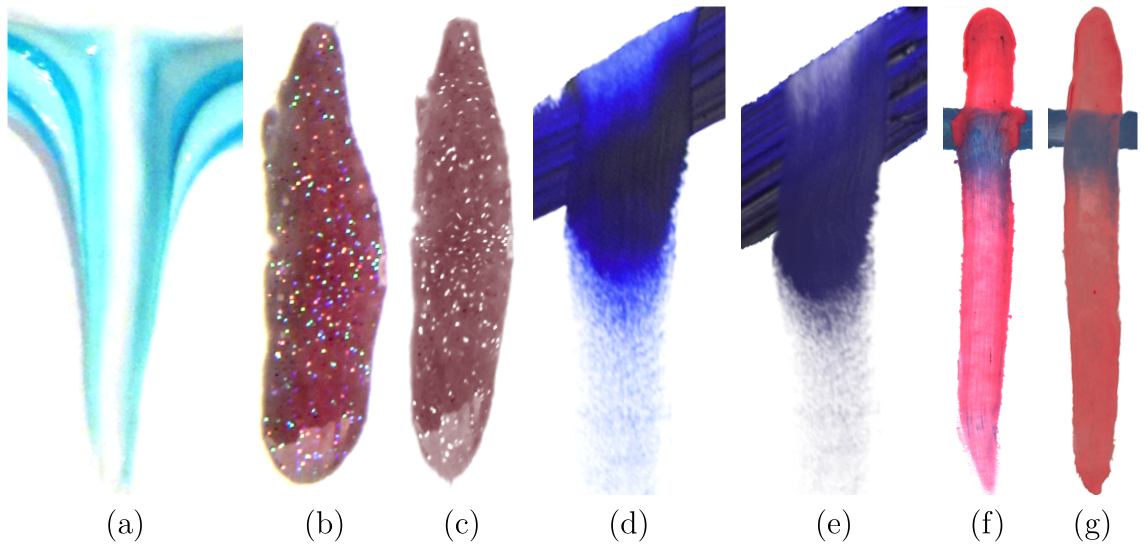


Figure 3.19: Synthesis limitations. (a) shows a toothpaste smudging example that will not be successfully reproduced using our synthesis approach. (b-g) shows side by side comparison of an exemplar (left) with the synthesis result (right). (b,c) lip gloss. Only using the lightness of exemplars fails to reproduce the color of the sparkles. (d,e) oil smudging. The textures are faithfully reproduced, but our simple color model fails to mimic hue changes inside the interaction region. (f,g) plasticine smearing. The textures and colors in the interaction region are plausibly reproduced, but the foreground stroke boundary remains untouched leaving an artificially clean look.

different from that of the exemplar. For smudging, a hue shift (from dark blue to cyan) can be observed from the exemplar in Figure 3.19d. Our use of simple alpha blending in the color integration step results in transition from dark blue to gray in Figure 3.19e. Regardless, in practice this is not a serious limitation as smudges and smears generally occur in complex paintings amid many strokes, where fine details are less noticeable (see Figures 3.14 and 3.21). For some media interactions, our factorized model assumptions do not hold. We assume that smearing synthesis does not alter anything outside the foreground stroke's silhouette. However, for some media, the silhouette may be changed slightly, as with plasticine stroke in Figure 3.19f. Our approach plausibly reproduces the textures and colors within the silhouette, but leaves the foreground stroke shape untouched which gives an artificially clean look (Figure 3.19g). For smudging, some challenging media such as toothpaste are not handled well by blending the smudging operator with the background strokes. Due to strong textures in toothpaste, the synthesized texture in the interaction region may not match the textures in the background stroke (see Figure 3.19a).

Currently, we do not support "scribble strokes," where the stroke overlaps itself, due to the use of MBA to compute the u, v parameterization (Section 3.6.2), which can limit paint mixing on-canvas. This is not a fundamental limitation, as alternative mechanisms for computing the parameterization may not have this problem. Meanwhile, our smudging functionality provides an easy alternative for paint mixing (see Figures 3.1b and 3.22a). Another limitation is that synthesis is performed on mouse-up. Both the piecewise matching optimization and the detection of overlap regions require the knowledge of the whole query stroke. Future work might investigate synthesizing mid-stroke, possibly using a sliding window for optimization.

3.11 Conclusion

We present RealBrush [88], a fully data-driven system for reproducing high fidelity natural media effects in real-time. We have shown that RealBrush is usable by artists and creates results that casual users cannot distinguish from photographs. Furthermore, we propose a factorized representation of natural media painting with explicit assumptions about physical properties that can motivate future research, and we make our database available to aid that goal.

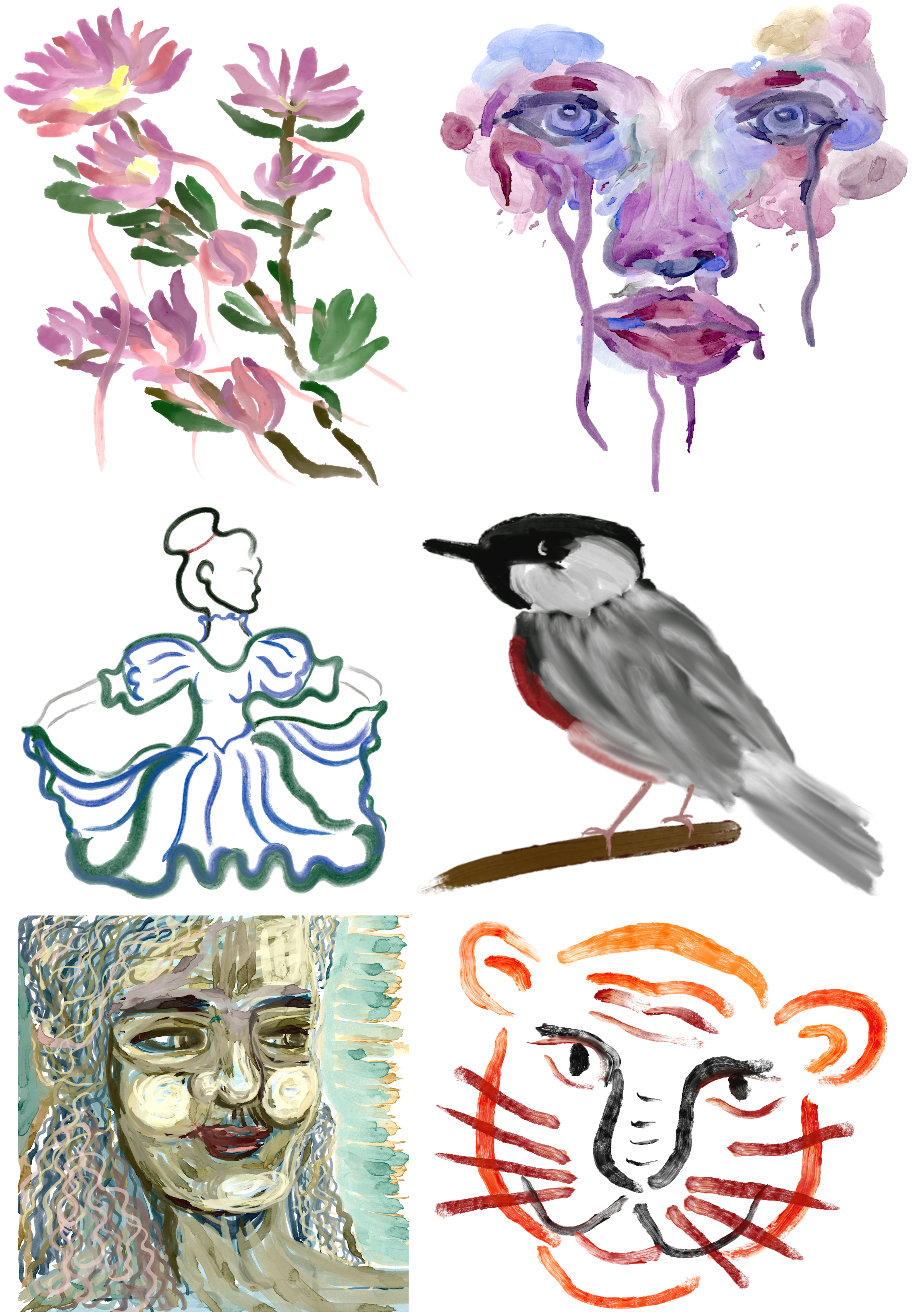


Figure 3.20: Example artwork by contributing artists. Used with permission.

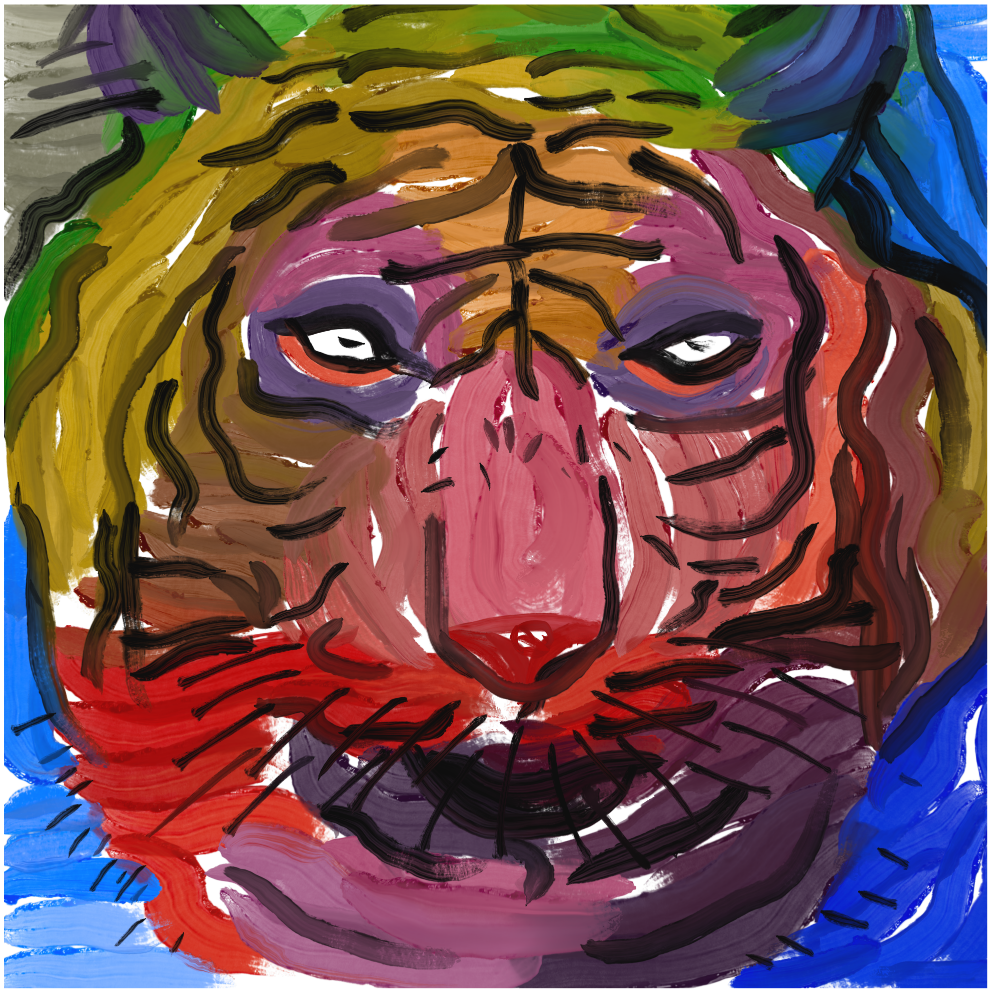
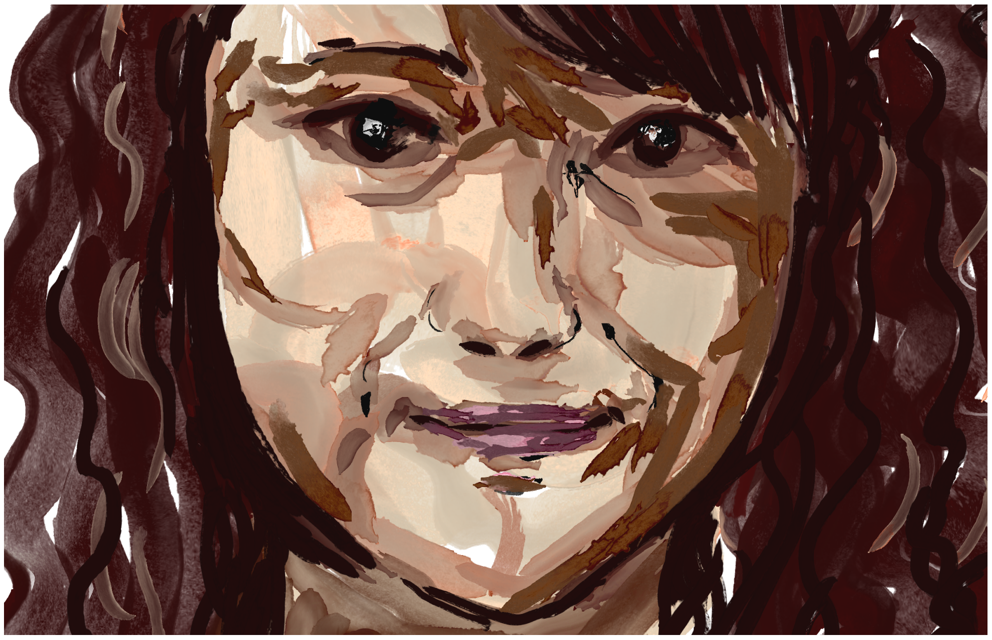



Figure 3.21: Example artwork by contributing artists. Used with permission.

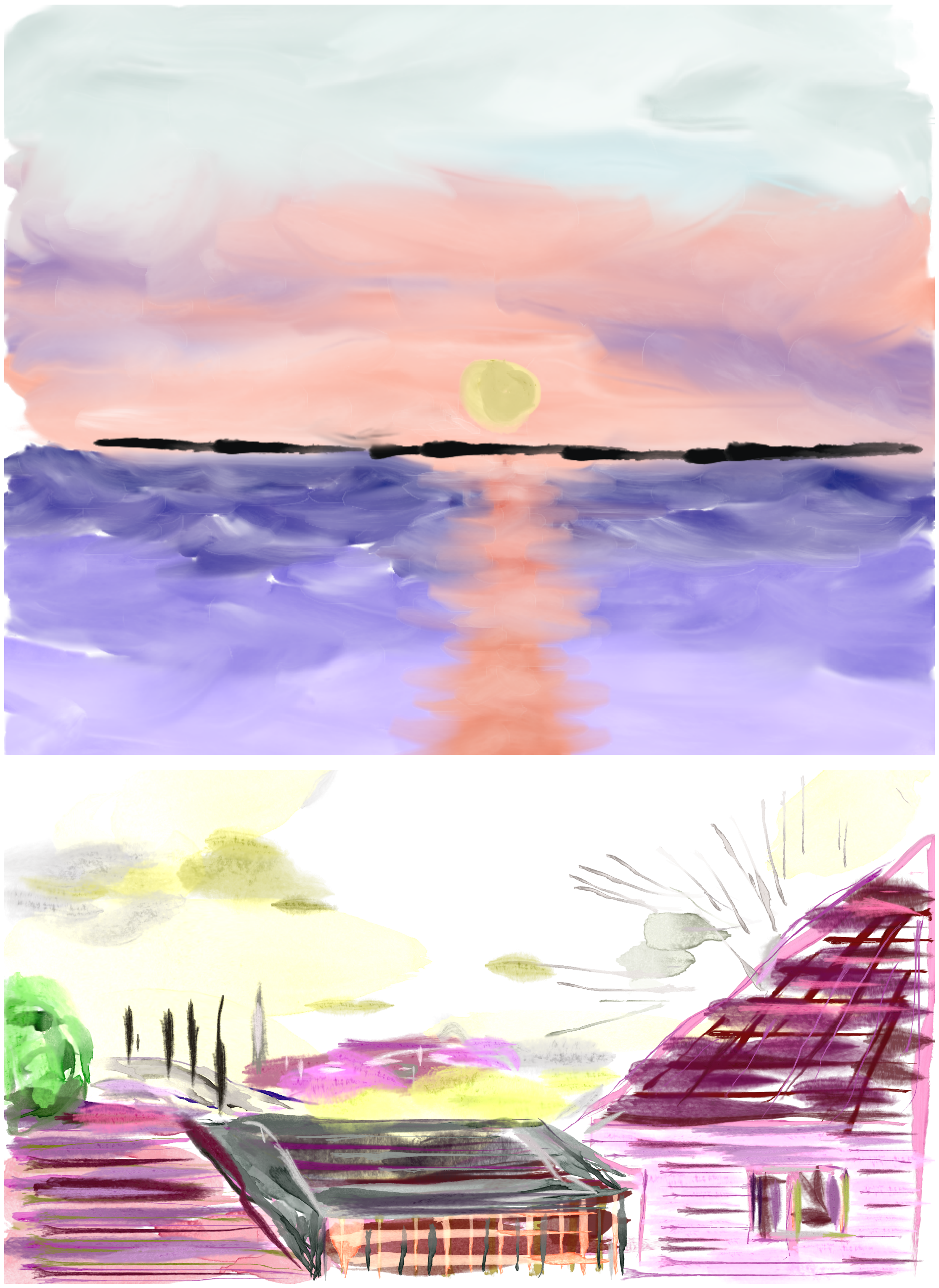


Figure 3.22: Example artwork by contributing artists. Used with permission.

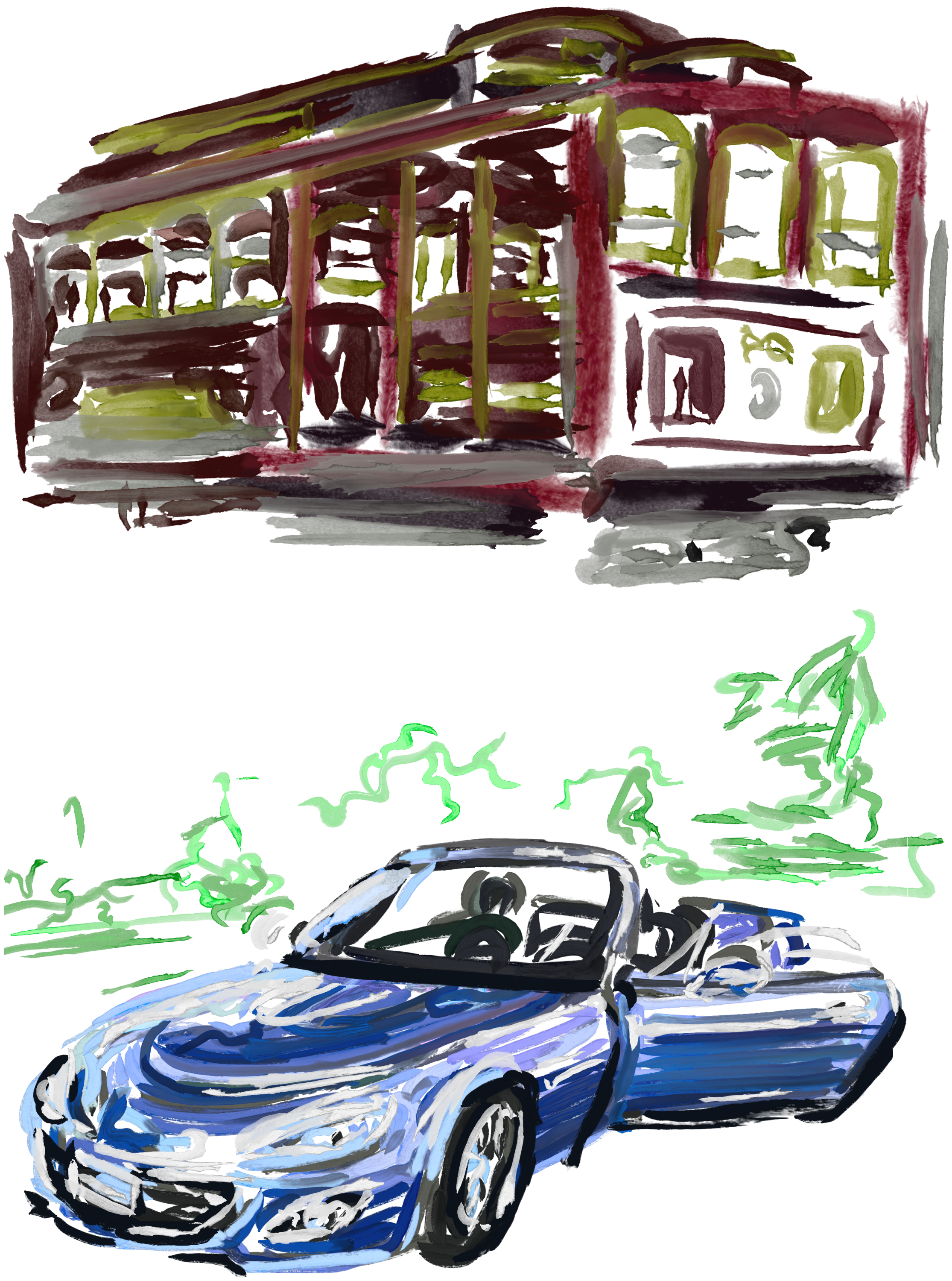
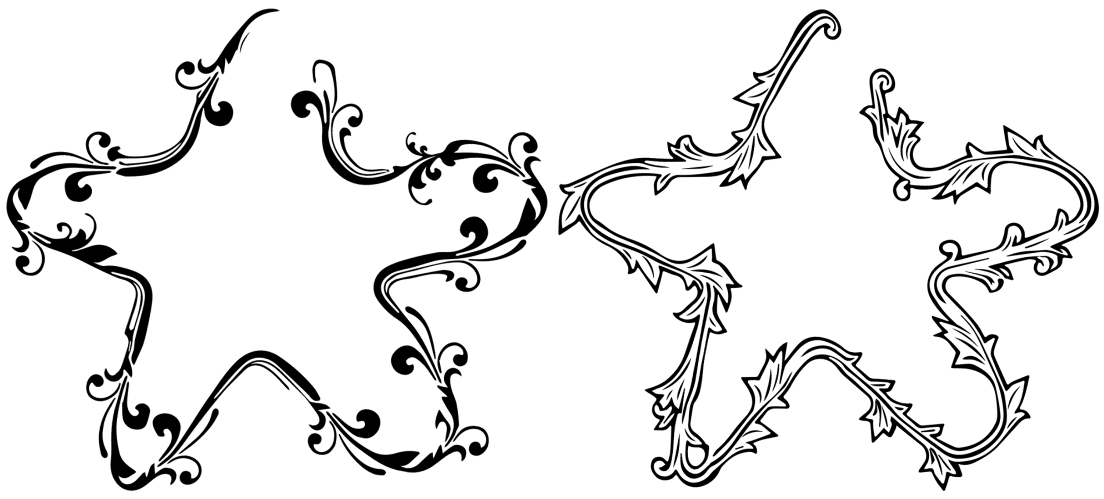


Figure 3.23: Example artwork by contributing artists. Used with permission.

Chapter 4

DecoBrush: Drawing Structured Decorative Patterns by Example

Structured decorative patterns are common ornamentations in a variety of media like web pages, greeting cards, interior design, etc. Creating such art from scratch using conventional software is time consuming for experts and daunting for novices. This chapter introduces DecoBrush, a data-driven drawing system that generalizes the conventional digital "painting" concept beyond the scope of natural media to allow synthesis of structured decorative patterns following user-sketched paths. The user simply selects an example library and draws the overall shape of a pattern. DecoBrush then synthesizes a shape in the style of the exemplars but roughly matching the overall shape. If the designer wishes to alter the result, DecoBrush supports user-guided refinement via simple drawing and erasing tools. This work improves on the synthesis method described in Chapter 3 by combining the advances of two texture synthesis paradigms, using a quilting-like approach for initialization followed by a pixel-based synthesis technique for optimization. For a variety of example styles, we demonstrate high-quality user-constrained synthesized patterns that visually resemble the exemplars while exhibiting plausible structural variations.



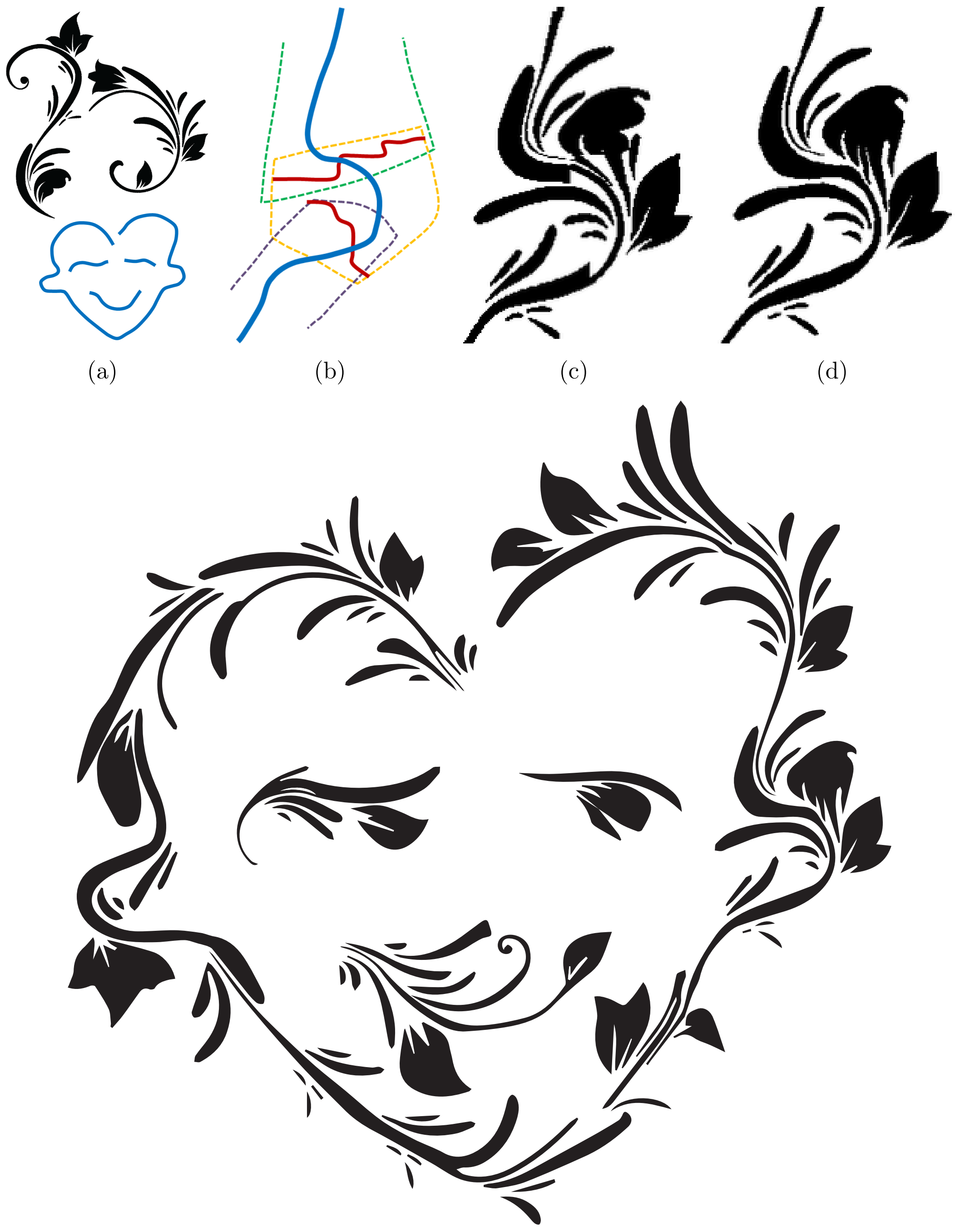
4.1 Background

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

This chapter introduces "DecoBrush", a data-driven drawing system that allows designers to create highly structured patterns simply by choosing a style and specifying an approximate overall path. The input to our system is a set of pre-designed patterns (*exemplars*) drawn in a consistent style (Figure 4.1a). Off-line, we process these exemplars to build a stroke library. During the drawing process, the user selects a library, and then sketches curves to indicate an intended layout. The system transforms the query curves into structured patterns with a style similar to the exemplar library but following the sketched paths. DecoBrush allows both experts and non-experts to quickly and easily craft patterns that would have been time consuming for the experts and difficult or impossible for novices to create from scratch.

Inspired by the work of Risser et al. [112] that enables synthesis of structural variation, this chapter builds on top of Chapter 3 to allow "drawing" structured decorative patterns. The decorative patterns shown in this chapter present a unique challenge for example-based texture synthesis, where most of the methods have been optimized for continuous-tone imagery where artifacts are largely hidden by fine-scale texture.

Our pattern synthesis works as follows. We first divide the query path into segments that are matched to similar segments among the exemplars, using dynamic programming to efficiently optimize this assignment. We then warp the segments using as-rigid-as-possible deformation to approximate the query path. Next at the *joints* where neighboring segments overlap, we rely on graph cuts (shown red in Figure 4.1b) to form a single, coherent figure that approximates the hand-drawn path. Despite the use of nearly-rigid warps combined with graph cuts at the joints, the resulting figure generally suffers from small stretching artifacts from the warp as well as misalignments at the joints (Figure 4.1c). Therefore we adapt structure-preserving hierarchical texture synthesis techniques to repair, refine and diversify the query stroke appearance. Our texture synthesis pipeline both ameliorates feature distortion within the warped exemplar segments and repairs broken local structures at the joints (Figure 4.1d). We also show that when synthesizing from coarse levels the pipeline brings additional structural variation into the design, consistent with the style of the exemplars. If the designer is not completely satisfied with the result of this process, our system also supports user-guided refinement of the synthesized result – a few new



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

(e)

Figure 4.1: The “flower boy” drawing (e) was created by synthesizing decorative patterns by example along user-specified paths (a)-bottom. (a)-top and Figure 4.2d show a subset of the exemplars for this floral style. (b), (c), and (d) show a closeup of the right ear. In (b), portions of the exemplars, shown outlined with dashes in green, yellow and purple, are matched along the user's strokes. Graph-cuts shown in red reduce but do not completely eliminate artifacts in the overlap regions, as seen in (c). Starting with this initialization, we run texture synthesis by example (d) and then vectorize the result (e).

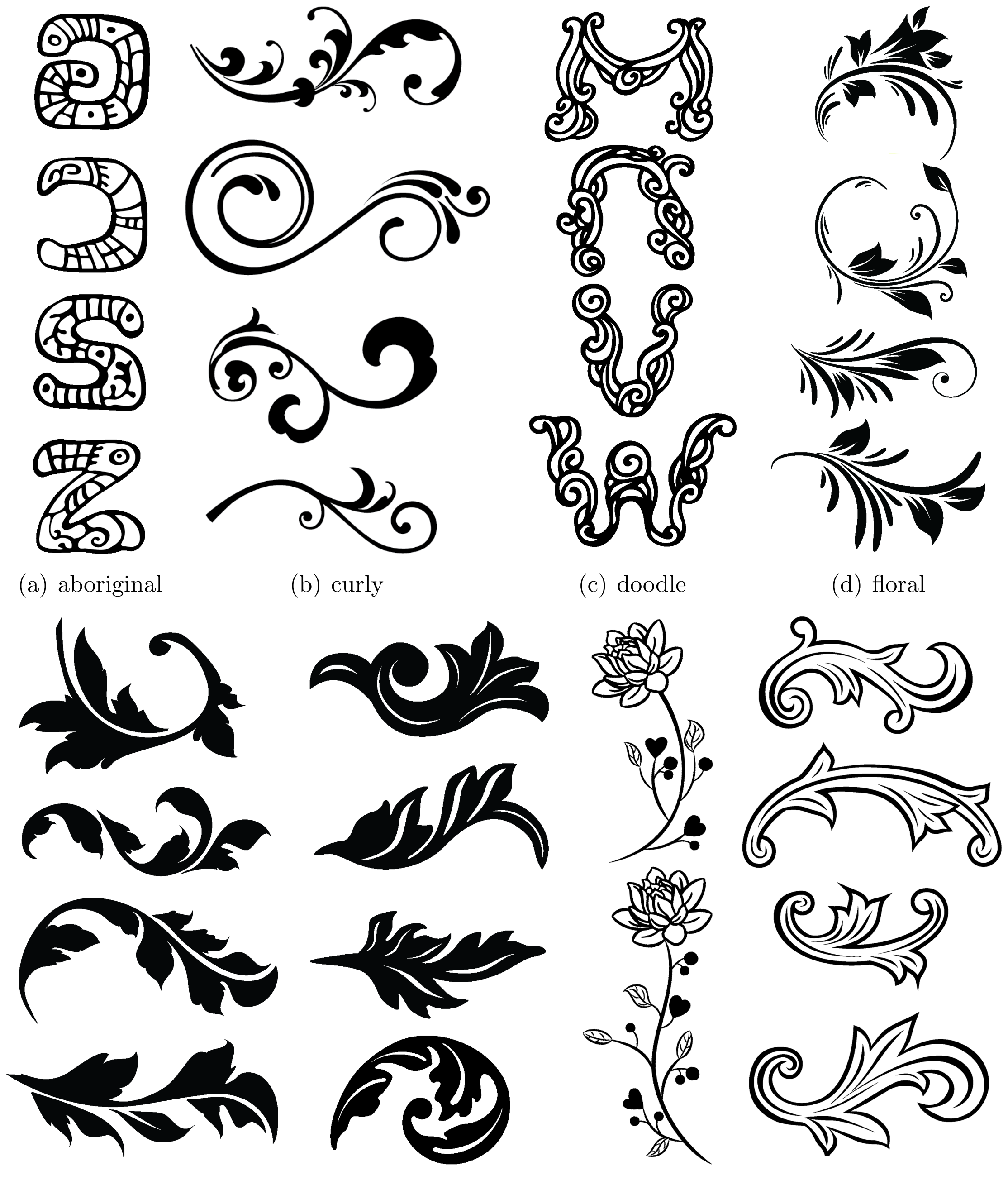


Figure 4.2: A selection of the structured stroke patterns used for synthesis.

strokes with brush or eraser tools form the input to a supplemental texture synthesis step that seamlessly incorporates the newly-drawn constraints with earlier results. Finally, we obtain a vector representation of the synthesis results (Figure 4.1e). The proposed synthesis pipeline is efficient, which facilitates applications such as user-guided pattern diversification and sketch-based decoration of confined regions.

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

4.2 Related Work

There have been **procedural approaches** for synthesizing decorative patterns. The early pioneer work by Wong et al. [145] use procedural rules to grow decorative patterns for the application of automatically filling a confined region. Other than reshaping and resizing the target region, they do not investigate other types of user interactions. Other procedural approaches allow growing structured patterns along a user-specified path, [2, 18], but the range of supported styles is heavily limited by the employed simplistic procedural rules. Měch and Miller [97] introduce an interactive procedural modeling framework, which generates complex decorative patterns. The procedural rules target plant-like structures and are hand-crafted by professional artists. Defining procedural rules for faithful reproduction of general structured patterns remains a challenging problem.

A variety of research has addressed **data-driven stroke synthesis**, where the general goal is to synthesize a 2D shape by example based on a 1D input query path. The *skeletal strokes* of Hsu et al. [56] warp pieces of the same structured texture to stylize lines. They focus on controlling the deformation of the selected features on the picture and do not introduce structural variations into the synthesis results. *Curve analogies* [51] introduces a statistical model for synthesizing new curves by example. However, they target 1D curves instead of the shape and texture of 2D strokes. Previous work use small patches of stroke exemplars to stylize users’ input lines, [3, 70]. Chapter 3 improves the synthesis quality of natural media strokes by warping and blending long exemplar segments. Other work, [93, 163], supports exemplars with repetitive small-scale local structures. Their techniques, however, do not differentiate the beginning and end of the strokes from the middle regions. We target multiple exemplars that contain high-level large-scale structures and characteristic appearance at both ends of the strokes. For this purpose, we use the piecewise matching idea introduced in Section 2.5 and 3.6 as the first step of our synthesis pipeline, which gives a good initial guess of the possible query stroke structure.

Another related line of research is **structured texture synthesis**. Markov Random Field-based texture synthesis approaches, stemming from the pioneering work of Efros and Leung [39], make the assumption that the appearance of a pixel is only dependent on a local spatial neighborhood regardless of the rest of the image. This model works well for homogeneous continuous-tone imagery. However, this assumption is weakened for highly structured textures where large-scale geometric features such as long connected lines are present, especially in binary or vector imagery where artifacts are difficult to hide. Previous work introduced hierarchical tile-based synthesis approaches, which better handle structured textures. Instead of synthesizing the image structure sequentially pixel by pixel, which often results in unrecognizable and spurious structures, previous work, [82, 83], explores the idea of tiling the exemplars and introducing structural variations by coordinate jitter and correction. They focus on synthesizing larger textures from a single small exemplar. Risser et al. [112] introduce structure-preserving jitter and show further evidence that the hierarchical tile-based parallel synthesis framework is able to preserve, modify and repair image structures. They also extended the synthesis pipeline to consider multiple source exemplars. However, their goal is to synthesize variations of the exemplars without any user constraints. We, on the other hand, target synthesizing new structured patterns following a user specified path. We borrow ideas from texture synthesis methods and follow the hierarchical upsampling and correction pipeline. The major difference is that we enforce user constraints by initializing the synthesis field using the matched exemplar segments that follow a similar path to the query. We also modify the upsampling and correction steps to suit our goals. The synthesized patterns contain meaningful structures and closely follow the input query path.

There has been work to use texture synthesis techniques to arrange **discrete elements**, [1, 7, 58, 62, 69, 80, 94]. The general idea is to define relationships between elements on a graph and synthesize new patterns by searching for input elements with graph-based neighborhood matching and pasting them to target locations. These approaches only support discrete primitives that do not intersect each other. However, the exemplars we have contain long, curly, intersecting lines and complex layout which pose additional challenges for defining inter-element relationships.

4.3 Algorithm Overview

Recent advances in sketch-based stroke synthesis [93] demonstrate promising results for synthesizing natural media strokes. The input strokes are not structured and can be combined using a simple blending algorithm. These approaches generate the synthesis results that closely follow the user's sketches, but cannot be directly applied to decorative patterns due to the difficulty of combining exemplar segments of very different structures. On the other hand, there has been remarkable progress in texture synthesis for structured exemplars, [83, 112]. However,

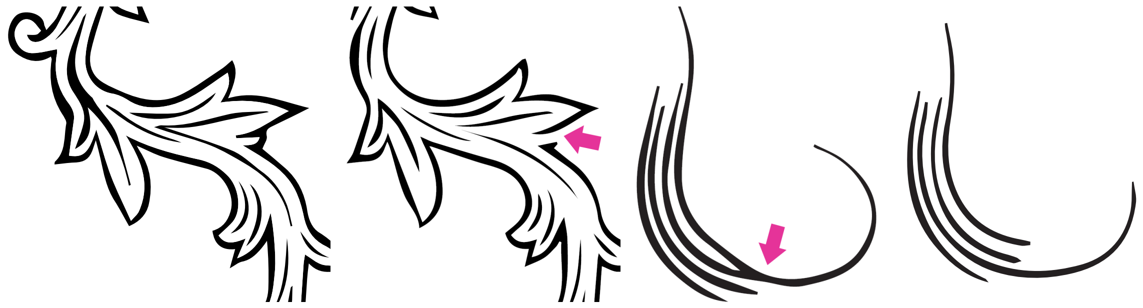


Figure 4.3: Trade-off of using Signed Distance Field. Lines are often joined better when using SDF (a,b), however this can also sometimes make spurious joins between unrelated lines (c,d).

the synthesis of high-level semantic contents following user constraints remains a challenging problem. We combine the advantages of both types of approaches for the new application of "painting" decorative patterns.

To prepare the exemplars for synthesizing new strokes, we need to extract a stroke-like semantic structure for each exemplar. We optionally convert the exemplars to a signed distance field representation (Section 4.4.1), which can preserve lines better for some exemplars. We then semi-automatically parameterize the exemplar stroke (Section 4.4.2). We also precompute the neighborhood information at every pixel so that runtime synthesis is efficient (Section 4.4.3).

At runtime, given a query stroke spine specified by the user, we look for long segments of exemplar strokes that have similar shape (Section 4.5.1) and apply as-rigid-as-possible deformation to align them with the query path (Section 4.5.2). We then apply graph cut to optimize the transition boundaries between the nearby exemplar segments (Section 4.5.3). The result of graph cut provides a good initial layout for the query stroke. We then use hierarchical texture synthesis approach to further reduce the warping artifacts and repair the broken image structures (Section 4.5.4). Finally, we vectorize the synthesized result using Adobe Illustrator.

4.4 Data Collection and Processing

We target synthesizing structured patterns such as the ones in Figure 4.2. The exemplars can have either a raster or vector representation. We have focused on synthesizing decorative florals, stylized fonts and other structured vector patterns. The exemplars share common characteristics: 1) They are composed of variable width curves and other simple solid-colored geometric shapes; 2) They have stroke-like structures with a clear directionality indicated by a center curve and/or the orientations of a group of curves; 3) They have unique appearance at the beginning and the end of the stroke.

We collect several libraries of structured patterns. Each library contains between 8 and 12 strokes all following a consistent design. The design is characterized by the

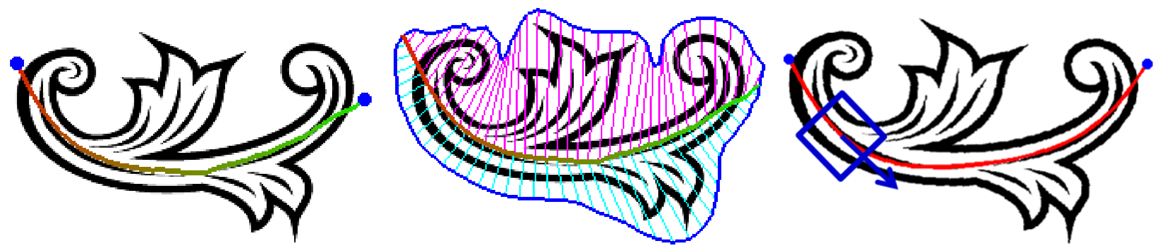

(a) (b) (c)

Figure 4.4: Exemplar parameterization. (a) The user specifies the blue end points and the spine. The transition from red to green indicates the directionality of the stroke from the start to the end. (b) The system then automatically parameterizes the stroke. (c) Given the end points, we can optionally optimize the red spine automatically. The blue square and arrow indicate the search neighborhoods are aligned with the stroking orientation.

choice of geometric primitives, the branching structures, the thickness and curviness of lines, etc. We collect strokes of different shapes, lengths and spine curvatures, which are then semi-automatically processed to create a library. Each library stroke is limited to lie within a square of resolution 512x512. We rasterize the exemplar strokes and optionally compute Signed Distance Field (SDF) representation (Section 4.4.1). Then we parameterize the exemplars (Section 4.4.2) in preparation for the runtime synthesis. Finally, we also pre-compute per-pixel neighborhood information on the raster exemplar (Section 4.4.3) for efficient runtime synthesis.

4.4.1 Signed Distance Field

Before further processing, we rasterize the input exemplars and optionally calculate the Signed Distance Field (SDF) [84]. In the SDF, black pixels have negative distances to the shape boundary and the white pixels have positive distances. The SDF effectively thickens the feature lines and introduces gradient information enriching the neighborhood information, which facilitates the piecewise matching process (Section 4.5.1) and the texture synthesis process (Section 4.5.4). Before the final vectorization, we simply threshold the SDF to extract a level set. However, as shown in Figure 4.3, the use of SDF can prove detrimental on some exemplars, where it can join unrelated lines. Thus we leave SDF usage as optional (we note in the supplementary where it has been used). The inset figure on the right shows an example of the SDF representation.

4.4.2 Exemplar Parameterization

Each decorative stroke has a clear directionality. The directionality is sometimes suggested by a continuous line that runs from the start to the end of the stroke. Other times, a number of lines collectively hint where the center branch is. To conduct

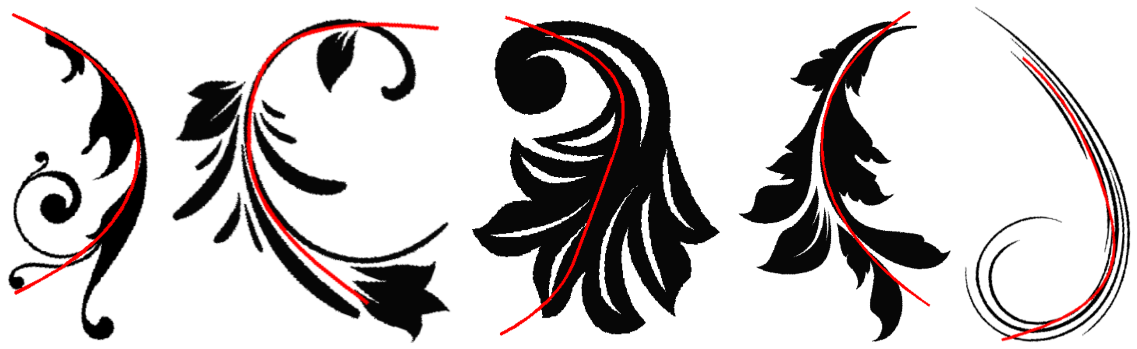
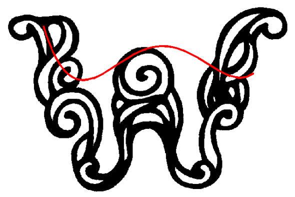


Figure 4.5: Automatic detection of stroke spine. The red curve indicates the detected spine.

synthesis, we need to extract the location of the main branch as a sequence of sample positions. To simplify the problem, we manually specify the start and end points of the main branch. Note that the start and end points do not have to be located on the tips of the structure or even on any of the feature lines (Figure 4.4a). The curls at the ends of the stroke serve for the purpose of decoration more than direction indication. They are part of the style that should be preserved in the synthesis results. By straightening the main branch towards the ends of the stroke, we can effectively simulate such curly appearance even when the query path is relatively straight.

With the two end points specified, we solve for the main branch by optimizing a path between them. Our goal is that the path does not wiggle too much, roughly follows the edge tangent flow (ETF) ([67]) of the exemplar, and is smooth. We find a polyline found by running Dijkstra's algorithm on a pixel grid containing the exemplar, where nodes of the graph are the pixels, and edges connect each pixel to neighbors distributed roughly uniformly in 32 directions. Each edge carries a weight w based on the vector v between the two adjacent pixels as $w = |v| + w_f(1 - f^p) + w_c(1 - c)$ where: f is the dot product of the v with the ETF sampled at its midpoint; c is the difference in curvature at the ends of this edge; and w_f, w_c and p are user-specified constants that control the behavior of these terms. (We found $w_f = 10$, $w_c = 3$ and $p = 4$ to work well in practice.) Finally, we fit a quartic polynomial to the vertices of the polyline to produce a smooth main branch that fits the artwork.



This typically produces a good match to the artwork and requires very little user effort (Figure 4.5). However, sometimes it fails to match the artwork well, especially where the curves of the exemplar do not present a consistent orientation (inset figure on the left). In these cases, it is straightforward for the user to sketch a preferred path for the main branch directly over the artwork. The optimized main branch (red to green indicates the stroke directionality in Figure 4.4a) coincides with the "center" of the pattern, slightly straightened up at both ends of the stroke. We then follow Section 3.5.1 to obtain the uv parameterization of the whole stroke (Figure 4.4b). Each stroke consists of between 30 and 100 samples, where a sample $\mathbf{t} = \{\hat{\mathbf{x}}, \hat{\mathbf{l}}, \hat{\mathbf{r}}\}$ consists of the 2D positions of the spine, left and right outline samples respectively.

4.4.3 Per-Pixel Processing

For efficient runtime synthesis, we further pre-process the exemplar image (or SDF if used) to extract hierarchical per-pixel information. We calculate three Gaussian pyramid levels for each exemplar. Note that since we do not perform the coordinate jitters, gaussian stack [82] which consumes more memory and computation is not needed. At each level, we only consider the pixels inside the automatically extracted stroke outline. We calculate a per-pixel orientation $\omega^e = (\omega^e_x, \omega^e_y)$, by interpolating the stroking orientations at the spine and the outline samples. The orientation of the spine is defined as $\bar{\mathbf{x}}_i = \hat{\mathbf{x}}_i - \hat{\mathbf{x}}_{i-1}$. The orientation of the outline is defined as $\perp \bar{\mathbf{l}}_\mathbf{i}$ and $\perp \bar{\mathbf{r}}_\mathbf{i}$, where $\bar{\mathbf{l}}_i = \hat{\mathbf{l}}_i - \hat{\mathbf{x}}_i$ and $\bar{\mathbf{r}}_i = \hat{\mathbf{r}}_i - \hat{\mathbf{x}}_i$. We then calculate an oriented 5x5 local neighborhood for each pixel with the upright direction (blue square and arrow in Figure 4.4c) of the neighborhood aligned with the per-pixel orientation ω_e.

4.5 Query Stroke Synthesis

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

4.5.1 Piecewise Matching

Given a query spine (blue curve in Figure 4.6a), we estimate a rough 2D shape and the *uv* parameterization (see Section 4.4.2). We then adapt the piecewise matching algorithm proposed in Section 3.6.1 to find a sequence of exemplar segments from the exemplar library and merge them together for the query stroke. To collect candidate nearest neighbors from the exemplars, for each query sample, we use the feature vector that includes the turning angle, the stroke width and the distance to the endpoints. In the dynamic programming step, we heavily penalize the ends of the query stroke being matched to the middle parts of the exemplars. For structured exemplars, it is especially critical to avoid cutting short the ends, since it might destroy important features and lead to broken lines that are hard to repair. We also avoid matching the middle parts of the query stroke to the end parts of the exemplar strokes and heavily penalize any jumps between library segments that have very different appearance or have very different stroke thickness at the segment boundaries (thickness is the rib length in the *uv* parameterization). After finding the appropriate library segments, we extend each segment in both directions to overlap the nearby matched segments. In Figure 4.6, the query stroke is matched to three exemplar segments.

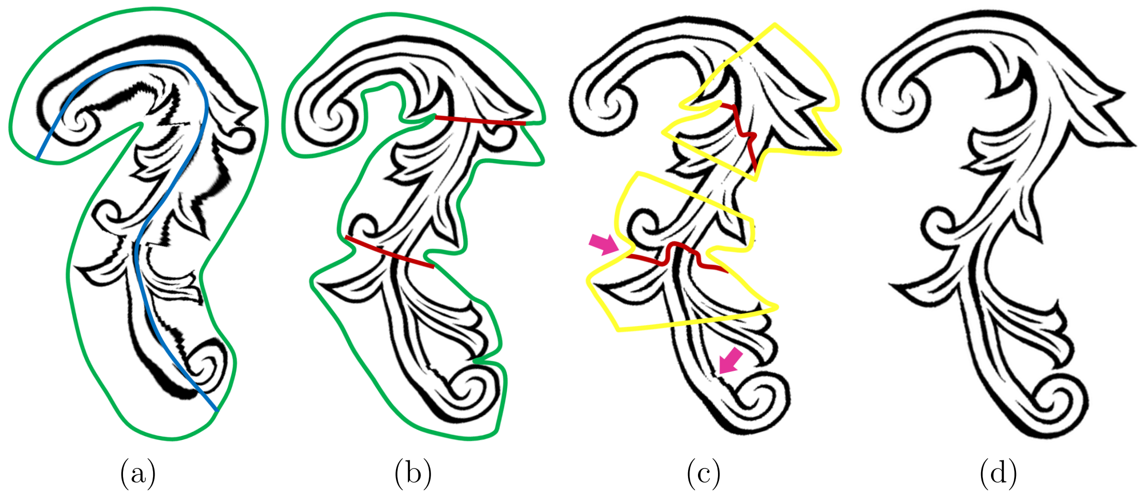


Figure 4.6: Synthesis pipeline. (a) The warping introduced in Section 3.6.2 results in noticeable distortions. (b) We apply As Rigid as Possible Deformation to recalculate the shape of the query stroke. (c) We find the optimal cuts (red curves) within the overlapped regions (outlined in yellow). (d) We apply hierarchical texture synthesis to reduce the residual distortion and fix broken line structures.

4.5.2 As Rigid As Possible Deformation

To synthesize decorative patterns closely following the user's intent, we need to align the spines of the exemplar segments with the input query spine. The warping method introduced by Section 3.6.2 leads to noticeable texture distortion for the structured exemplars (Figure 4.6a). We instead use As Rigid As Possible Deformation [61] to reduce the distortions. The query stroke's spine samples (blue curve in Figure 4.6a) are fixed at the input locations. The locations of the outline samples (green curve in Figure 4.6b) are optimized to reflect the original shape of the exemplar segments. The use of As Rigid As Possible Deformation minimizes the amount of structural distortion and therefore leaves an easier task for the texture synthesis step (Section 4.5.4) to remove the residual warping artifacts. Note that though it is later changed, the query stroke's input outline (green curve in Figure 4.6a) is utilized for the piecewise matching process in constructing the feature vector. Thus, if thin query strokes are drawn with little pressure, then thin exemplar segments are likely to be matched.

4.5.3 Graph cut

After the adjacent library segments are extended and deformed, we detect the region where they overlap (outlined by yellow curves in Figure 4.6c). We apply a 2-label planar graph cut algorithm [120] to find the least cost cut that smoothly transitions from one library segment to another. We define a graph on the overlap region where each pixel is a node and each node has four edges connecting the adjacent nodes. Each pixel (j, k) corresponds to two stroke textures, L and R. Then the cost of each node $\mathbf{E}_c$ and edge $\mathbf{E}_d$ is defined as:

$$\mathbf{E}_c = w_c(2 - L_{j,k} - R_{j,k}) + C \tag{4.1}$$

$$\mathbf{E}_d = \begin{cases} (L_{j,k} - R_{j,k+1})^2 & \text{going up} \\ (L_{j,k} - R_{j,k-1})^2 & \text{going down} \\ (L_{j,k} - R_{j+1,k})^2 & \text{going right} \\ (L_{j,k} - R_{j-1,k})^2 & \text{going left} \end{cases} \tag{4.2}$$

$\mathbf{E}_c$ is designed to penalize lengthy cuts and cuts that pass through black feature lines on the exemplar, which might result in undesirable changes in line thickness. $\mathbf{E}_d$ represents the difference in intensity of the two adjacent pixels along the edge direction from the two library segments respectively. We use $C = 0.02$ and $w_c = 0.01$. Figure 4.6c shows the result after applying graph cut (red curves in Figure 4.6c), in comparison to the result of applying a naive cut (red curves in Figure 4.6b) along the stroke rib in the middle of the overlap region.

The output of the piecewise matching and the graph cut steps include, for every query pixel, a local orientation $\omega^q = (\omega^q_x, \omega^q_y)$ (Section 4.4.3) and a 3D texture coordinate $\mu = (\mathbf{i}, \mathbf{x}, \mathbf{y})$, where $\mathbf{i}$ indicates the index of the exemplar from which this pixel originates.

4.5.4 Texture Synthesis

The graph cut step (Section 4.5.3) finds good transition boundaries most of the time, but at times the results might contain undesirable distortions and broken or jagged line structures (Figure 4.8a). We therefore apply a fast hierarchical texture synthesis step to fix the distortion and the broken structures, inspired by the approach proposed by Lefebvre et al. [82]. Their key idea is to initialize, synthesize and upsample the exemplar coordinates instead of the pixel values. During the upsample step, the finer level pixels inherit and offset the coordinates of the coarser level. Using coordinate inheritance better preserves the sharp image features compared to applying bilinear interpolation on the pixel values. We follow their synthesis pipeline, but modify the steps to suit our need. The synthesis pipeline is initialized using the exemplar coordinates produced by graph cut (Section 4.5.4). Then, we improve the image structures by iteratively replacing the initialized exemplar coordinates with the center coordinates of the appropriate exemplar neighborhoods (Section 4.5.4). The synthesized exemplar coordinates are then upsampled to the next finer level (Section 4.5.4). We alternate the correction and upsample steps until we reach the finest level and skip the upsample step at the finest level. We found three synthesis levels produce good results (Figure 4.6d).

Initialization

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

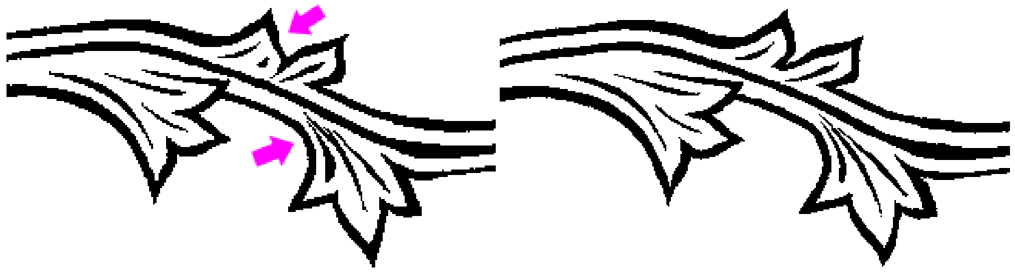

(a) 1 orientation (b) 11 orientations

Figure 4.7: Searching over more orientations slightly improves the synthesis results.

Correction

The essential step that repairs the broken image structures is inspired by the correction step proposed by Lefebvre et al. [82], which we briefly review. During correction, the exemplar coordinate of each query pixel is replaced by the center coordinate of an exemplar neighborhood that is most similar to the local query neighborhood measured by L^2 distance. The correction step also uses the idea of coherent synthesis [4], and considers the 3x3 immediate neighbors of the current query pixel. Correction can be applied to non-adjacent pixels independently, which allows fast parallel processing. Several iterations of this step for each pixel at each synthesis level ensure that the synthesis result is locally similar to the exemplars everywhere and therefore faithful to the style.

We adapt this step in several ways. We sample rotated query neighborhoods with the up axis aligned with the per-pixel orientation, ω^q (calculated in the same way as in Section 4.5.3). Rotating both the exemplar and the synthesis neighborhoods to align with the stroking direction increases the chances of success for neighborhood matching. In the overlap region of the adjacent exemplar segments (red curves in Figure 4.6c), the image structures are the most challenging to repair. We therefore apply approximate nearest neighbor search [101] to find the most similar 5x5 exemplar neighborhood among all exemplar strokes for each 5x5 query neighborhood. For each query pixel, we additionally gather eight coherent exemplar neighborhoods by applying an appropriate offset to the coordinate of each pixel in the 8-connected neighborhood. We favor choosing the closest coherent neighborhood unless the L^2 distance to this neighborhood is significantly larger (5 times or more) than the L^2 distance to the approximate nearest neighbor. We find the use of strong coherence very important for fixing image structures and maintaining clean feature lines. For the non-overlap region, we find it satisfactory to only select the best one among the eight coherent neighborhoods for faster synthesis performance. To better tolerate the distortion introduced by the warping step (Section 4.5.2), we can optionally sample a few query neighborhoods rotated at several different angles around and including the stroking direction, ω^e. We perform the search for each rotated query neighborhood and select the exemplar coordinate μ and the optimal query orientation ω^e that gives the lowest L^2 neighborhood distance. Figure 4.7 demonstrates slight quality improvement when orienting the query neighborhoods at 11 different orientations

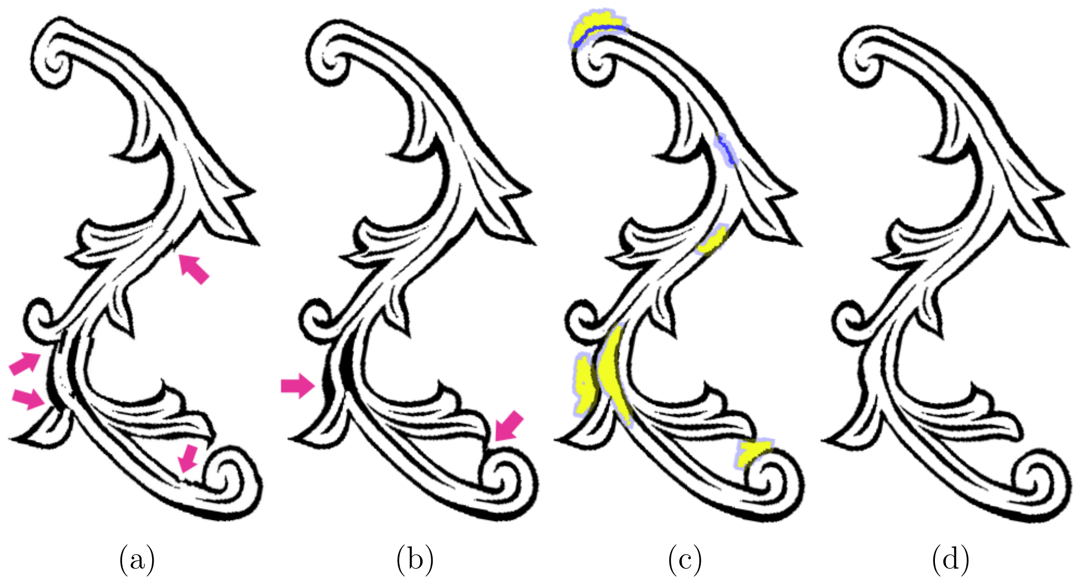


Figure 4.8: Synthesis refinement using secondary strokes. (a) The graph-cut step sometimes finds sub-optimal transition boundaries. (b) The texture synthesis step reduces but cannot completely remove the artifacts. (c) Users can apply refinement strokes (yellow indicates erasing and blue indicates adding). The light blue regions indicate the masks inside of which the synthesis pipeline modifies the exemplar coordinates. (d) The final synthesis result. Notice that though the user's scribbles are noisy, the synthesis pipeline produces smoother curves.

around the stroking angle. Due to the diminishing return of searching over different orientations, all results of this chapter are generated searching only one orientation exactly aligned with the stroking orientation.

Upsample

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

4.5.5 User-guided Refinement

Often the result of texture synthesis is satisfactory. However, in some cases, the graph cut step finds a sub-optimal cut due to the dissimilarity between the nearby segments (Figure 4.8a), which poses a challenging problem for texture synthesis. On top of the synthesized result (Figure 4.8b), users can optionally refine the design or fix the remaining artifacts by scribbling refinement strokes. The refinement strokes are drawn as black or white discrete pixels (visualized as blue or yellow pixels in Figure 4.8c) at the highest synthesis level, which facilitate adding or removing features respectively. To improve upon user's imprecise refinement strokes and obtain clean feature curves, we apply a slightly modified texture synthesis pipeline. We first dilate the refined pixels to obtain a binary mask $\mathbf{M}$ (visualized as light blue region in Figure 4.8c), which indicates the region of pixels to be changed. We downsample the refinement strokes and the binary mask for two levels until the coarsest synthesis level and seed the synthesis pipeline. At the coarsest level, we disable the coherence search for the user refined pixels, since there are no corresponding exemplar coordinates at these locations. We find the corresponding exemplar coordinates by searching for nearest exemplar neighborhood. We then upsample the updated exemplar coordinates to the next level and continue the usual synthesis pipeline (Section 4.5.4). At each synthesis level, the pixels outside the mask are locked and remain unchanged. At the coarsest level, the pixels inside the mask are updated to reflect the user's refinement constraints, and at other levels they are unconstrained. The hierarchical synthesis pipeline effectively smooths user's refinement strokes by matching to the exemplar. The synthesized curves are smoother than the initial refinement strokes, which is useful for novices who have difficulty drawing clean lines. All results of this chapter except the fonts and Figure 4.8 are generated without synthesis refinement.

4.6 Results and Discussion

The results in this chapter are all synthesized with exemplars of a maximum resolution of 512×512 for performance and memory considerations. Figure 4.9 shows that for datasets which contain very thin lines, using resolution higher than 256×256 is crucial for maintaining thin line structures. On the other hand, for the exemplars in Figure 4.2c, 4.2e and 4.2f, we find the resolution of 256×256 is enough for synthesizing reasonable structures. The whole synthesis pipeline takes about 1-2 seconds to synthesize a stroke similar to the one in Figure 4.6. At 512×512, the synthesis is about 5 times slower. On average, each stroke in this chapter takes about 8 seconds to finish. We did our performance measurement with the following hardware, Intel Core i7 2.3 GHz CPU. We believe the performance can be improved to real-time by implementing the synthesis correction step on GPU [82].

The synthesis starting level can be determined by the user based on the quality of the graph cut result. For minor texture distortion (uneven and jagged curves), starting the synthesis using the exemplar resolution of 256×256 can sufficiently smooth out the feature curves. With more severe texture distortion or broken lines, synthesizing three hierarchy levels starting from 128×128 improves the results. Starting from even lower resolution presents a trade-off in the result quality. With the exemplar resolution of 64×64, close-by features are sometimes merged together undesirably (magenta arrows in Figure 4.12a), while on the other hand, the

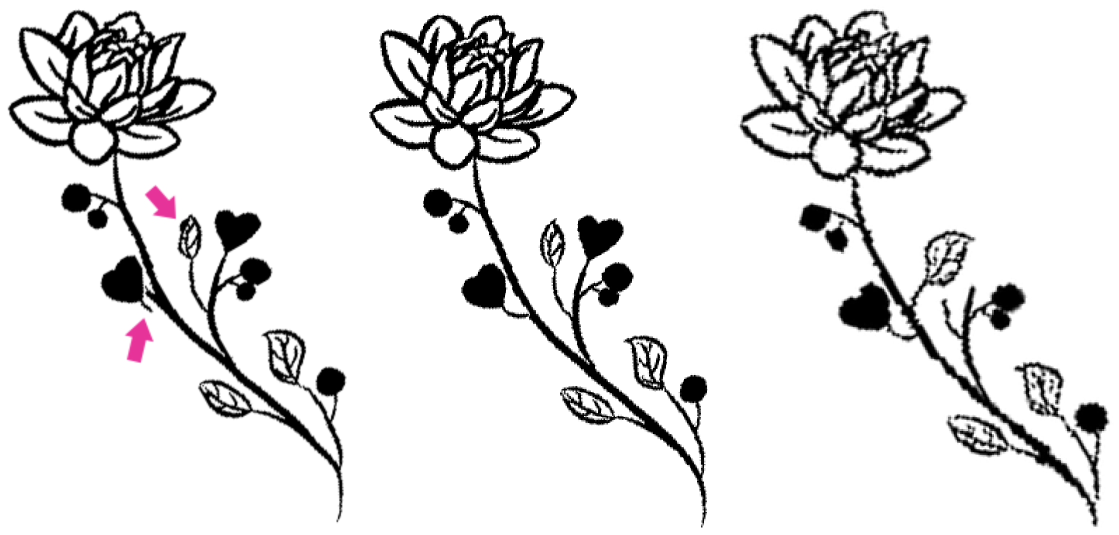

(a) Graph-cut (b) Final result (c) Lower-res result

Figure 4.9: Synthesis result comparison. (a) The graph-cut step gives good initialization at the cost of small artifacts. (b) The texture synthesis step removes the local artifacts and connects broken lines. (c) Using exemplars at lower resolution of 256×256 cannot preserve thin line structures in the synthesis results compared to using 512×512.

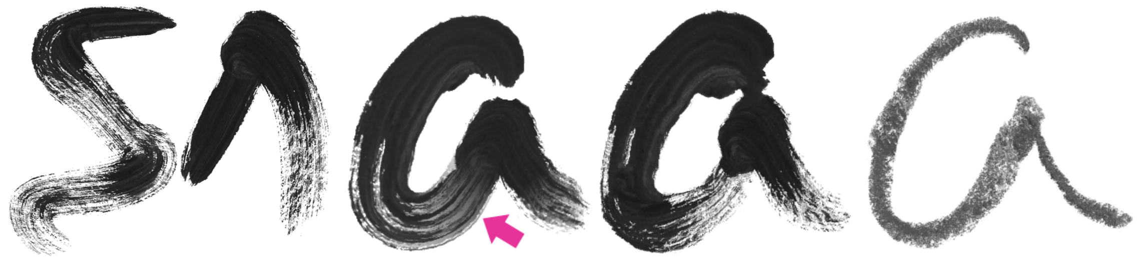

(a) (b) (c) (d)

Figure 4.10: Natural media synthesis results. (a) Sample dry watercolor exemplars. (b) RealBrush alpha blending result. (c) DecoBrush result on the same set of exemplars. (d) DecoBrush result using pastel exemplars.

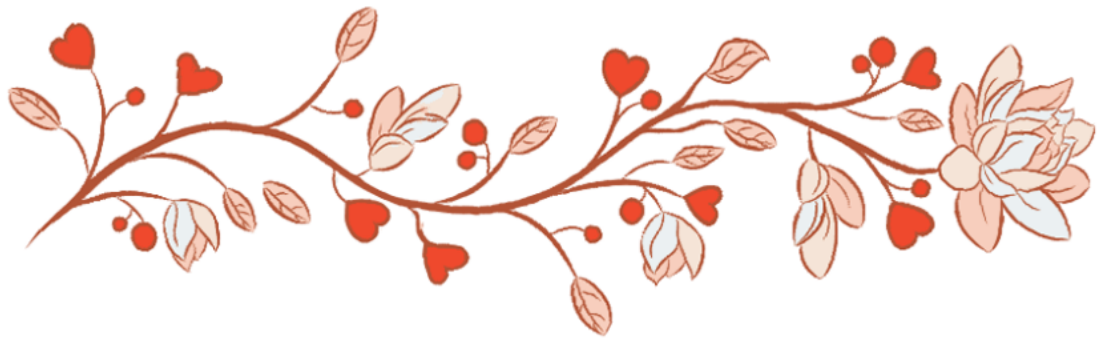
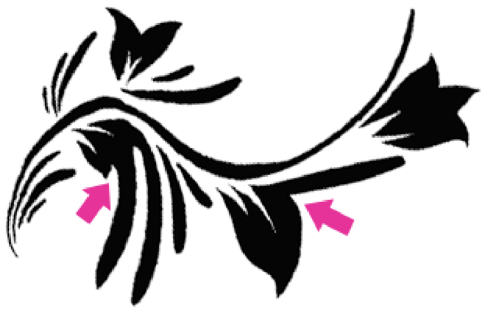


Figure 4.11: Colored synthesis result.


(a) 64 × 64

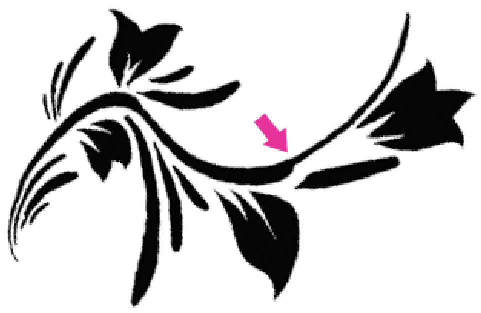

(b) 128 × 128

Figure 4.12: Trade-off of synthesizing from different resolutions. Starting the synthesis with exemplars at lower resolution improves the smoothness of the curves, but aggravates the problem of incorrectly connecting nearby sub-structures.

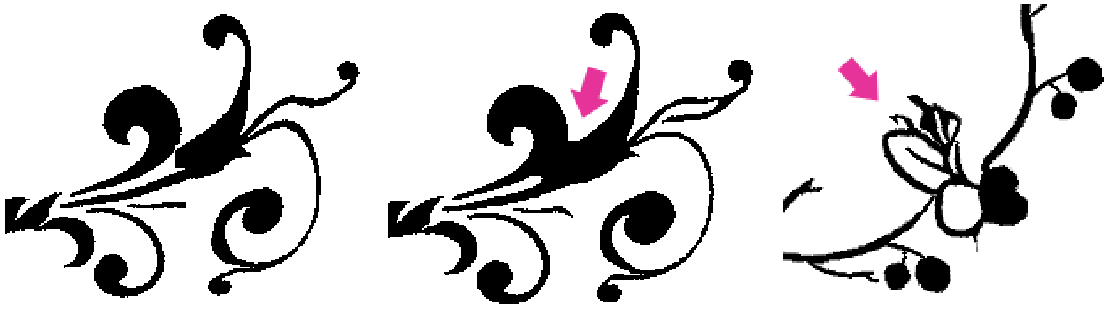

(a) (b) (c)

Figure 4.13: Synthesis limitations: Even though graph-cut results in separate structures (a), synthesis sometimes falsely connects them together (b). Details can appear in inappropriate places, like the partial flower in the middle of the stroke (c).

synthesized curves are usually smoother (magenta arrow in Figure 4.12b). This reveals a limitation (Figure 4.13a,b) of our approach. Curves that are near each other might be represented by only a few pixels at the coarse levels and are sometimes integrated into a single component through the synthesis correction steps.

We tested the robustness of our system by sketching strokes of different lengths and shapes synthesized with eight different exemplar libraries (see supplementary materials). We also made simple drawings and decorative designs shown in Figure 4.1e, 4.18, 4.19, 4.20, 4.21. Each of the designs was created by novice with less than 20 strokes in total. The resulting drawings contain complex structures in the styles of the exemplars. We vectorized our drawing results using Adobe Illustrator.

If gray-scale image exemplars are used, our approach is also able to synthesize structured natural media exemplars with improved quality. Figure 4.10 demonstrates that compared to RealBrush, our synthesis results are free of blending artifacts and contain realistic variations in the stroke thickness. Figure 4.11 shows that our system also addresses colored exemplars. We convert the colored exemplars to grayscale and synthesize them as usual. Since the main synthesis component works by referencing texture coordinates, we synthesize the colors by directly reading them from the exemplars.

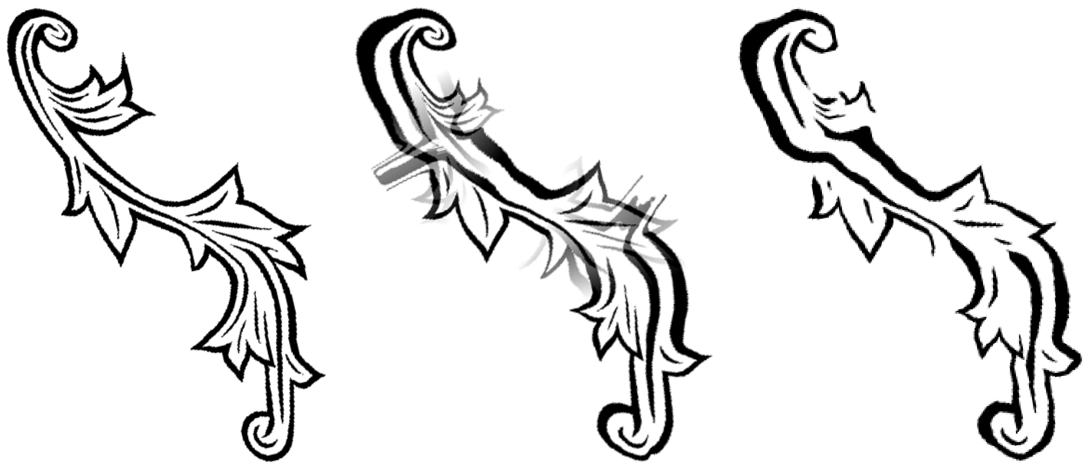

(a) DecoBrush result (b) RealBrush alpha blending (c) texture synthesis

Figure 4.14: Comparison with RealBrush (Chapter 3). All three results are synthesized using the exemplar library in Figure 4.2h.

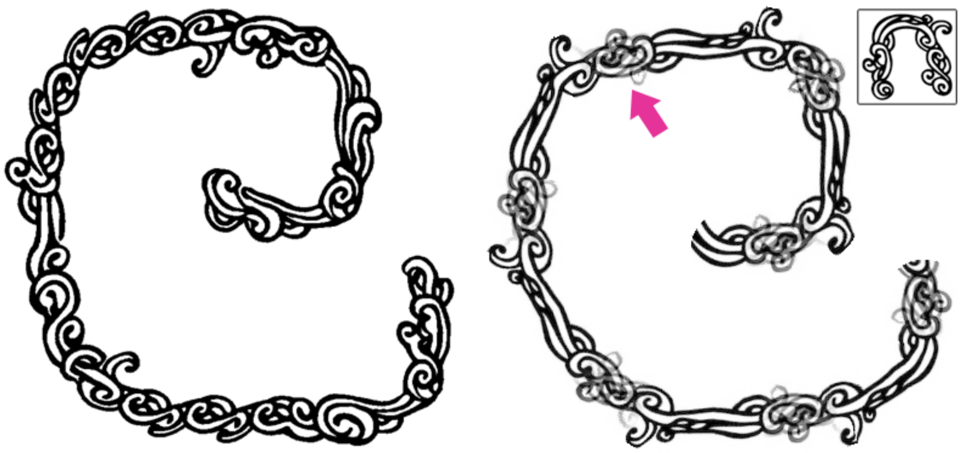

(a) DecoBrush result (b) Painting-By-Feature result

Figure 4.15: Comparison with Painting-By-Feature [93]. The result of Lukáč et al.[93] uses the single exemplar shown in the upper right corner.

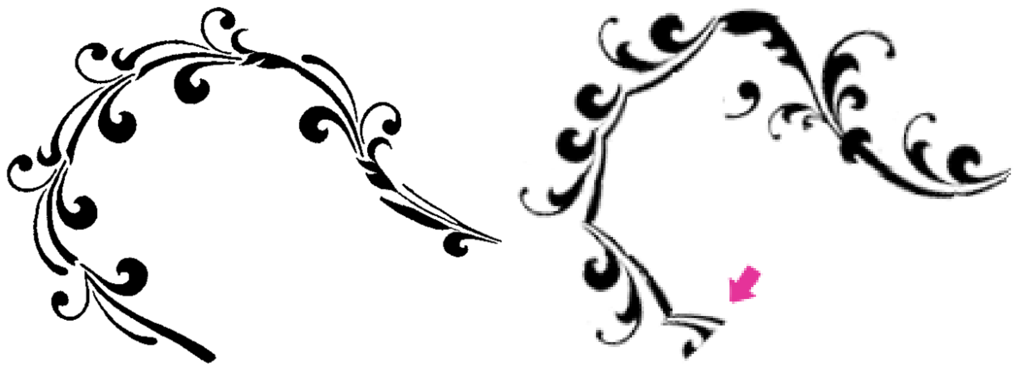

(a) DecoBrush (b) The work of [163]

Figure 4.16: Comparison with the work of Zhou et al. [163]. The results are synthesized with the "curly" exemplars in Figure 4.2b.

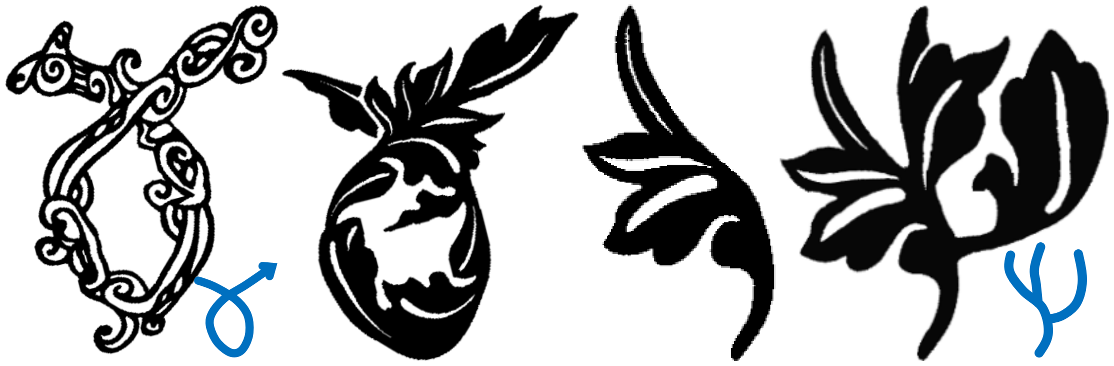
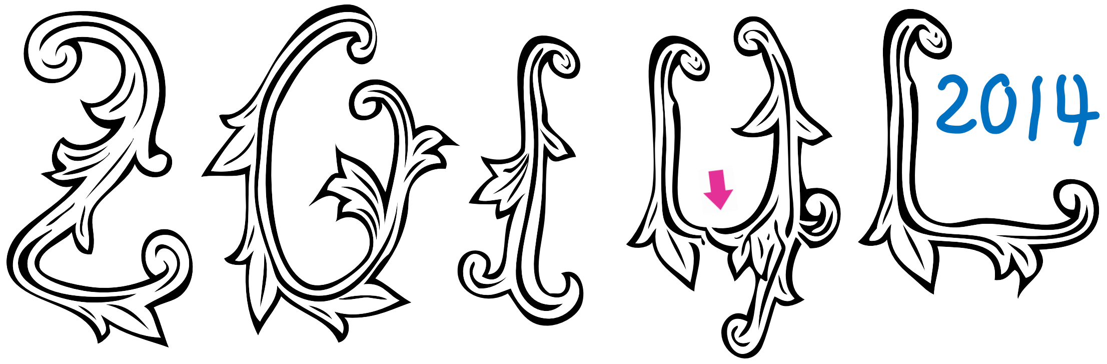

(a) (b) (c) (d)

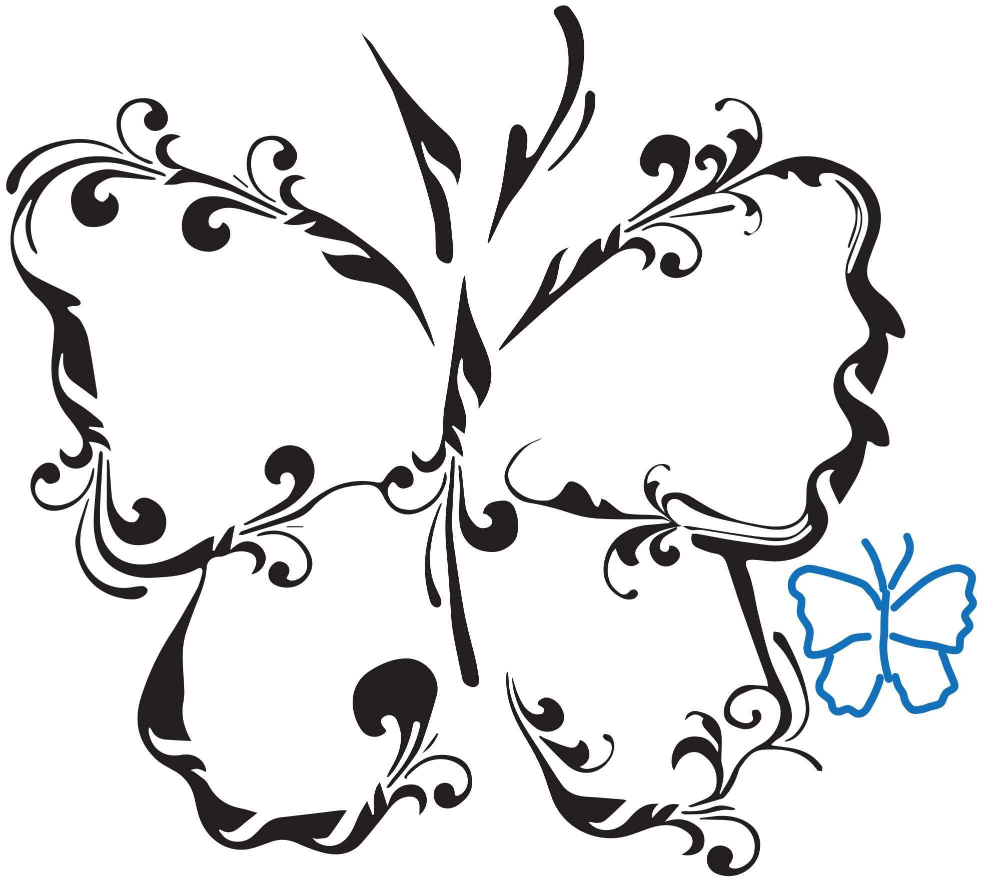
Figure 4.17: Stroke intersection and branching results. (a-b) demonstrates the self-intersecting strokes. (d) shows the result of branching two strokes from the main spine in (c). (a) is synthesized with "doodle" in Figure 4.2c. (b-d) are synthesized with "palm" in Figure 4.2f.



(a) the number "2014" (b) partial "4"

Figure 4.18: Synthesis results and stroke crossing limitation. (a) "2014" is synthesized with the "wings" exemplars in Figure 4.2h. The digit "4" is drawn with two crossing strokes. Our current pipeline cannot synthesize proper crossing appearance due to the lack of crossing structures in the exemplars. (b) shows the first stroke of "4" before the crossing.

In Figures 4.14, 4.15 and 4.16, we compare with RealBrush (introduced in Chapter 3), Painting by Feature [93] and the work of Zhou et al.[163]. In Figure 4.14, the RealBrush alpha blending result introduces noticeable distortions, and the texture synthesis module removes delicate line structures, since it was designed for highly textured media such as sponge or glitter. In Figure 4.15, Painting by Feature introduces ghosting artifacts and does not handle the stroke ends differently from the middle. Figure 4.16 shows that the method of Zhou et al. cuts off the exemplar features at the end of the query stroke and generates jagged spines that do not closely follow users' input paths.


(a) Butterfly (synthesized using Figure 4.2b)

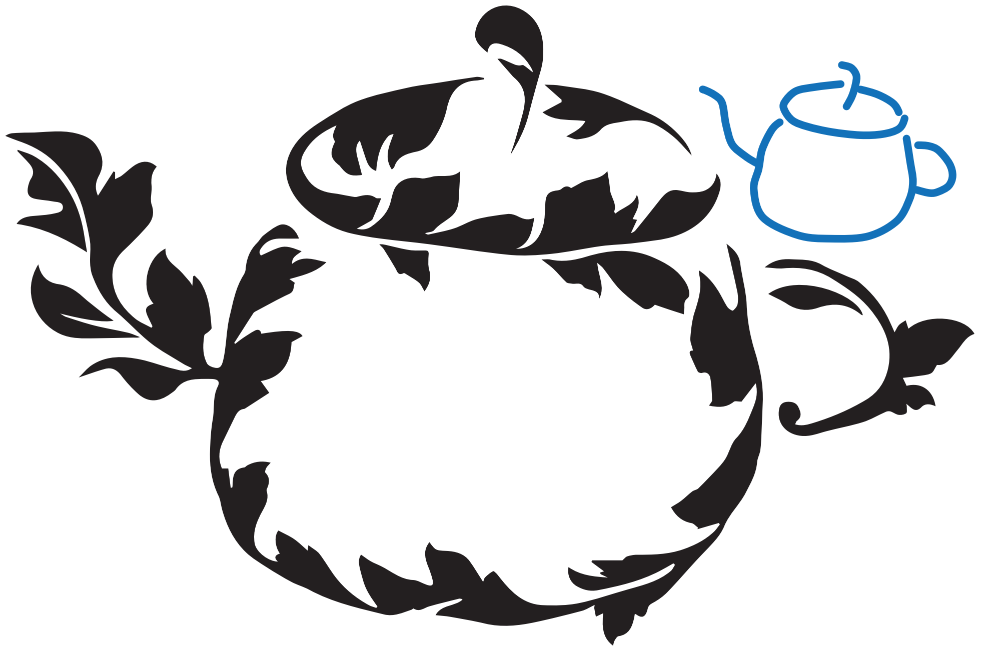
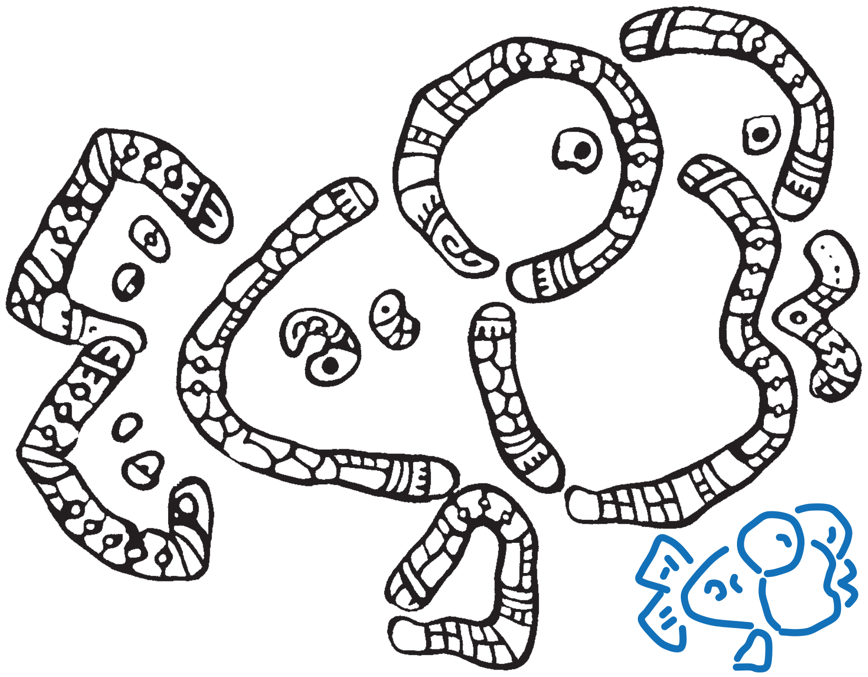
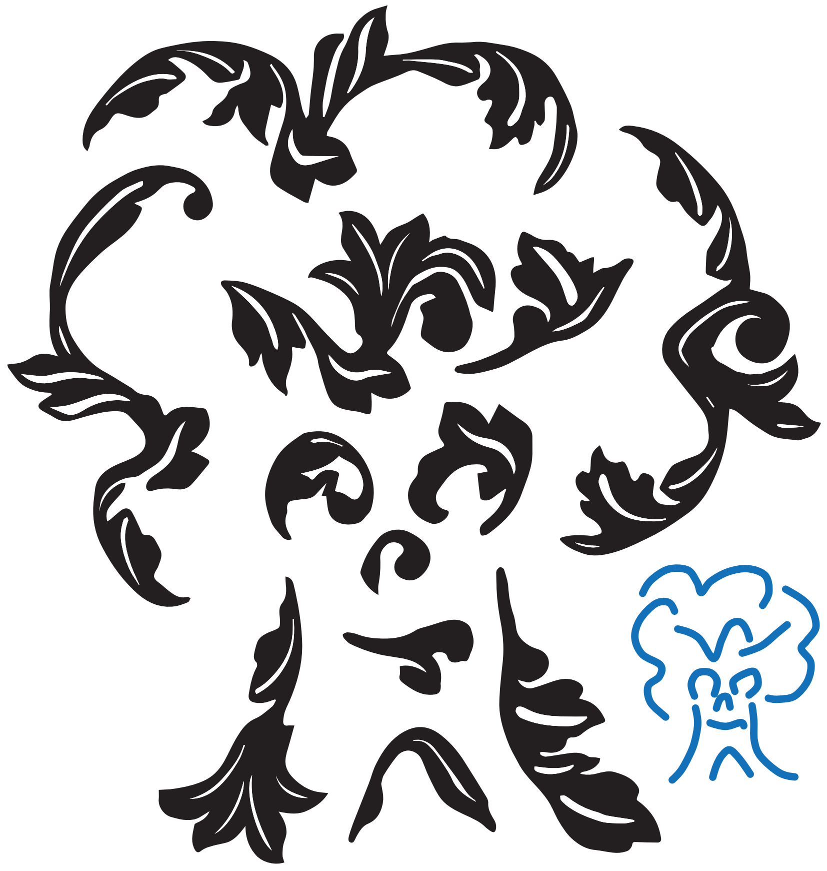

(b) Teapot (synthesized using Figure 4.2e)

Figure 4.19: Synthesized drawings.



(a) Fish (synthesized using Figure 4.2a)



(b) Tree lady (synthesized using Figure 4.2f)

Figure 4.20: Synthesized drawings.

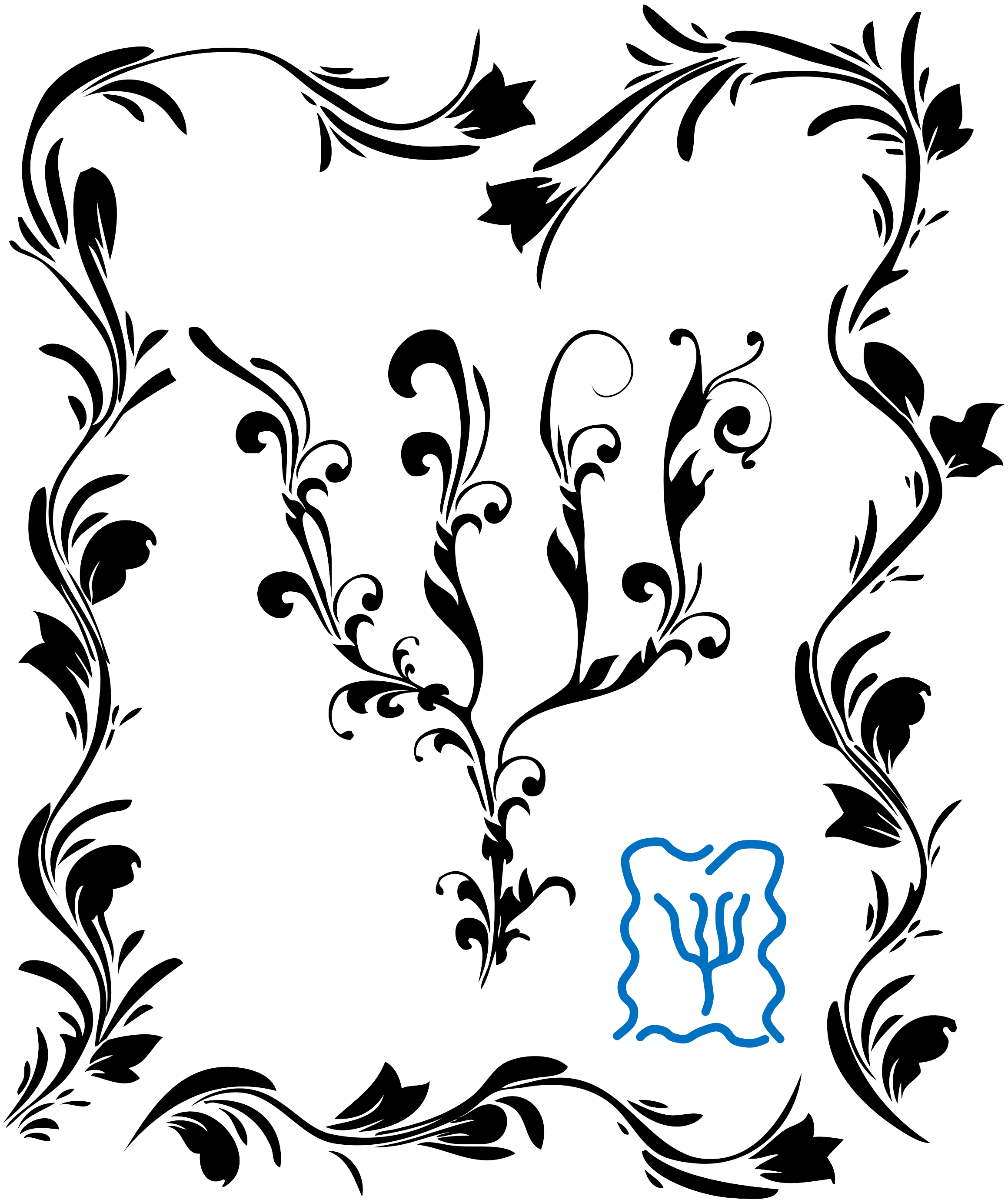


Figure 4.21: Synthesized drawings. The outer result is syntheszied using Figure 4.2d. The inner result is synthesized using Figure 4.2b.

4.7 Limitations and Future Work

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

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

Variation in output. Because the initialization to our texture synthesis process comes from piecewise-matching portions of exemplars to the query path, the statistics of the resulting texture can be skewed by the shape of the query. For example, a near perfect circular query may cause a single, similar-curvature portion of an exemplar to match repeatedly along the query, leading to repetition. Lukáč et al. [93] introduced an efficient algorithm for matching the statistics of an exemplar set when selecting them for a linear sequence. It may be possible to adapt their algorithm for our setting where we need to take special care at the endpoints of curves and also match the rough shape of the query. More broadly, our system currently has no random component, so the same input always produces the same output. It would be nice to offer a mode suitable for *ideation* where several different variations are presented and the designer chooses among them.

Parameterization challenges. First of all, there are other general-purpose parameterization techniques available, [121, 130], that are better than the one introduced in Section 4.4.2. Additional parameterization challenges exist for our particular problem. For example, when drawing a curve around a sharp corner, branches that extend towards the curve can intersect in the corner. We believe that a more sophisticated parameterization of the path could ameliorate these artifacts. Moreover, our current framework cannot handle some forms of decorative art – those with very large features relative to the size of the strokes, or which do not have an obvious directionality. To extend the kinds of artwork that could be handled well, it would be desirable to be able to express secondary branches off the main spine. During sketching this could provide additional control over where such features appear. In addition it would be ideal if there is some mechanism whereby a user could indicate regions of the exemplars that should remain intact (such as the head of the flower in the “rose” dataset), so that artifacts shown in Figure 4.13c can be avoided.

Vectorization from SDF. As discussed in Section 4.4, many of our example styles work best when we synthesize a signed distance field (SDF) rather than binary image as output. In such cases, our current approach is to use Adobe Illustrator to vectorize from a level set in the SDF. However, the synthesized SDF contains much more data (highly redundant) than simply the values neighboring the level set, and this information might be used to address small topological problems that are difficult to repair directly from the level set. One strategy might be to use the gradient and SDF values sampled at many patches in the neighborhood of the level set to generate point and normal samples at the boundary, and then reconstruct a boundary from this (noisy) data using Poisson reconstruction [68].

Stroke Intersections and Branching. With the piecewise matching pipeline, when a long stroke intersects itself, the new drawn part overwrites the previously drawn parts. Then the texture synthesis pipeline refines all pixels in the query stroke in the same way, including the overlap region. We can synthesize meaningful intersections for some datasets that contain intersection-like structures, like “doodle” and “palm” (Figure 4.17a,b). In addition, we can emulate branching by starting a new stroke from the main branch of a previously synthesized stroke. The texture synthesis step merges the new structures with the existing structures, creating a branching appearance. Figure 4.17c shows a synthesized stroke before branching. Figure 4.17d demonstrates the effect of adding two branches. The decorative capital letters also demonstrate branching: the T and D are generated with two strokes, where the beginnings of the second strokes overlap the first strokes. To better handle intersections and branching for more complex datasets such as “floral” and “wings”, one should design explicit synthesis procedures for the overlap regions based on exemplars that contain proper crossing and branching structures. Figure 4.18a shows that the current synthesis pipeline does not synthesize optimal structure in the overlap region between the two strokes of the digit “4”.

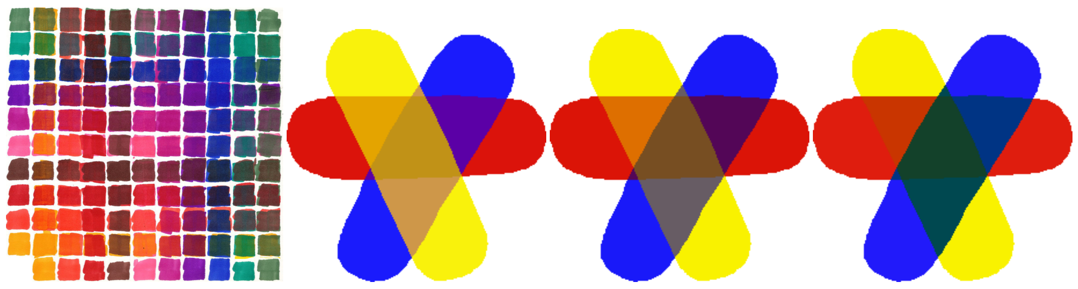
4.8 Conclusion

We introduce DecoBrush [89], a novel sketch-based design interface that makes the vector design task easier and accessible for novices. We propose a novel combination of two texture synthesis paradigms to synthesize, for the first time, complex strongly structured exemplars, using a quilting-like approach for initialization followed by pixel-based synthesis for optimization. Our system design balances the interactive performance and the synthesis quality. The major insight is that raster-based texture synthesis, with strong preference for coherence, can be used to synthesize structured vector patterns, but only given good initial structures. We publish our exemplar database to facilitate future research in interactive vector design.

Chapter 5

RealPigment: Paint Compositing by Example

The color of composited pigments in digital painting is generally computed one of two ways: either alpha blending in RGB, or the Kubelka-Munk equation (KM). The former fails to reproduce paint like appearances, while the latter is difficult to use. Chapters 3 and 4 deal with color layering using alpha blending. This chapter presents data-driven pigment models that reproduce arbitrary compositing behavior by interpolating sparse samples in a high dimensional space. The input is a color chart, which provides the composition samples. We propose two different prediction algorithms, one doing simple interpolation using radial basis functions (RBF), and another that trains a parametric model based on the KM equation to compute novel values. We show that RBF is able to reproduce arbitrary compositing behaviors, even non-paint-like such as additive blending, while KM compositing is more robust to acquisition noise and can generalize results over a broader range of values. The proposed color models can be integrated with the painting systems described in Chapters 3 and 4 for enhanced stroke interaction appearance.



(a) marker color chart (b) alpha blending (c) radial basis (d) Kubelka-Munk

Figure 5.1: Color compositing: (a) a color chart (made with markers) crafted by an artist in order to observe how different kinds of pigments appear when layered; (b) alpha blending, though standard in computer graphics, fails to capture known effects like yellow over blue gives green; (c) RBF interpolate and extrapolate from the data in the chart to produce more realistic effects; (d) fitting parameters of a KM model to the data in the chart improves on RBF for paint-like behaviors.

5.1 Background

Digital drawing and painting tools mimic many aspects of traditional media with increasing fidelity. Much of the research has addressed the shape and texture of individual strokes, as well as secondary effects like bleeding and smearing. However, the interaction of multiple pigments has received relatively less attention, yet remains crucial for plausibly simulating the effects of many media including watercolor and oil paint. The major strategies for layering digital pigment have been either to rely on simple ad-hoc methods like alpha-blending using various blend modes, or to resort to complex physical models such as that of Kubelka and Munk (KM) [77]. Simple alpha blending maps trivially onto the traditional graphics pipeline, but spectacularly fails to capture straightforward paint mixing principles such as "yellow and blue makes green" (Figure 5.1b). On the other hand, physically accurate models like KM (Figure 5.1d) require many parameters to capture each individual pigment, and even more parameters when they are mixed. Of course it is possible to measure such KM parameters for a set of paints and use them in a digital palette, but this requires sophisticated hardware and restricts the artist to working within the fixed set of paints, and when drawing still requires specifying more parameters such as density and thickness – this extra complexity leads to cumbersome interfaces in a digital painting tool.

Instead this chapter proposes a data-driven approach toward layering color in digital painting tools. In our process, the way that colors blend is specified by leveraging a physical *color chart* like the one shown in Figure 5.1a. Artists use such instruments to understand how the range of colors expressible in a particular palette of paints will combine, before applying paint to canvas. The artist first applies a blob of a particular color across each cell in a row, and moves onto the next row with a different color, and so on. Next, the artist applies a blob of color across each cell in a column, with a different upper color for each column. The resulting grid depicts visually how all pairs of color will appear when layered, and informs their application in a subsequent painting.

In our proposed digital painting interface, the user simply selects a color chart (the *exemplar*) in order to specify how colors will be composited. The color chart may be selected from a set of pre-existing charts, or may be created anew in order to specify a novel mixing behavior. Next, the user simply selects (RGB) colors using a standard color picker and paints strokes, and the system plausibly composites colors in a way that mimics the exemplar chart.

This chapter makes a series of contributions to enable this interface. The first contribution is a novel formulation of the pigment compositing problem as interpolation among a sparse set of high dimensional samples. Our second contribution is, based on this novel formulation, detailing how to acquire such a set of samples via a color chart, and interpolate them using radial basis functions (RBF) [110] (Figure 5.1c). The third contribution is to improve upon RBF by using a priori knowledge about the shape of the function by using the KM equation, effectively interpolating in KM space (Figure 5.1d). Finally, we explore the performance of these methods compared against a baseline of optimized alpha blending. We show that for pigment models that can be reproduced by the KM equation, the optimized KM interpolation is able to produce better results that smooth over acquisition errors and can be extrapolated to a broader range of compositions. Conversely, RBF interpolation better approximates the exact behaviors in the color chart, whether or not those behaviors are realizable by pigments (e.g. additive blending).

5.2 Related Work

There are extensive research on modeling physical pigments to accurately predict the color when they are combined to improve printing processes and make realistic natural media painting systems.

Kubelka-Munk and Other Physically-Accurate Models.
The seminal work by Kubelka and Munk [77] (KM) defines a per-wavelength equation for the reflectance of a homogeneous, isotropic pigment layer atop a substrate in a continuous setting (see Section 5.3). Alternate formulations have been developed in the same domain [42, 49, 140], but KM remains the most commonly used.

Much work has been done to improve KM. Saunderson [119] provides a correction based on surface reflection of the pigment layer, and Berns and De la Rie [17] extend this result by considering reflective varnishes for oil paint. Granberg and Edström [47] show the KM model mis-estimates strongly absorbing pigments, and Yang and Kruse [157] propose a revised KM model which corrects for path-length errors in highly scattering media. However, Edström [35] later show this revision had other errors.

Extensions have been proposed to relax the assumptions made by KM. Shakespeare and Shakespeare [124] propose a method to incorporate fluorescence, while Yang etal [158] add inhomogeneous materials. Edström extends these to also include anisotropic reflections [36, 37]. Finally, Sandoval and Kim [118] re-derive the KM equation from the radiant transport equation, thereby generalizing it to handle inhomogeneous materials with boundaries, and with greater accuracy.

Approximations to Kubelka-Munk.
Many natural media painting systems have used the KM model in various forms to improve the quality of paint rendering. The Stokes Paint program [11] uses linear combinations of four or eight acquired real pigments with KM coefficients to generate a wide gamut of paints. Impasto [13] improves on this with a numerical approximation of the same pigments for reduced memory and computation.

Curtis et al. [25] uses a reduced KM model with coefficients for only three wavelengths, which are computed from user-specified RGB colors. A further approximation is presented by Wang and Wang [142], which uses a single coefficient variant of KM, for each of three wavelengths. The simplest approximation is found in the work of Rudolf et al. [115], which uses one absorption and scattering coefficient pair plus an RGB color, thereby entirely obviating multispectral rendering.

Gossett and Chen [46] create a paint-inspired color model they label RYB (for Red Yellow Blue), which defines three paint pigment primaries and linearly interpolates among them to define a paint-like gamut. This is most similar to our approach, but we generalize the concept to support arbitrary, non-orthogonal, sparse sets of paint samples with data-defined compositing behavior.

Finally, instead of explicitly defining an equation to model the pigment composition effect, Xu et al. [152] and Westland et al. [144] both propose using a neural network to learn how pigments combine to produce the final reflectance spectra. These algorithms must be trained on acquired pigment composition samples with KM coefficients, and do not provide clear mechanisms for exerting artistic control over the resulting model.

Pigment Acquisition.

Acquiring the parameters of individual pigments to use in these various models is an arduous task, involving multispectral measurements of controlled samples in controlled environments [106]. Mohammadi and Berns [100] show that it is crucial to capture the full KM coefficients for a pigment in order to accurately reproduce the range of generated colors. Kokla et al. [71] propose a way to use a set of captured pigments to determine from a simple image of a painting or manuscript the mixture of pigments used for each local color, via an optimization.

5.3 Pigment Model

he commonly accepted method for predicting the reflectance of a layer of pigment is to use the Kubelka-Munk (KM) equation [43, 77]:

$$\begin{gathered} R = \frac{1 - \xi(a - b \coth bSh)}{a - \xi + b \coth bSh} \\ a = 1 + \frac{A}{S},\ b = \sqrt{a^2 - 1} \end{gathered} \tag{5.1}$$

where h is the thickness of the pigment layer, S and A are the pigment's scattering and absorption coefficients, ξ is the substrate reflectance, and R is the resulting reflectance of the pigment layer. All of these quantities are per-wavelength, so a pigment p is modeled as a set of scattering and absorption coefficients over a set of wavelengths L as $p = \{S_l, A_l | l \in L\}$.

Realizing the RGB to display in an image corresponding to a stroke of pigment on a canvas is a standard multispectral rendering problem. Let us denote this transform as $c = \{r, g, b\} = f(p)$, for a pigment p. $f(p)$ is the perceptual response of the pigment, which could be regarded as the space that artists think in when selecting pigments for painting. That is, rather than understanding pigments in terms of a set of coefficients or a reflectance spectrum, an artist may associate pigment p with a particular color c, for some nominal conditions (illuminant, thickness, substrate).

Now consider two pigments, p_1 and p_2, painted on a white canvas such that p_2 is on top of p_1. The perceived color of this composition is $f(p_2 \circ p_1)$, which can be computed by the recursive application of the KM equation. However, from the artist's perspective, the behavior is better modeled as $g(f(p_2), f(p_1))$, where g is a composition function that operates on RGB values. Indeed, a digital artist selects an RGB to paint with, and needs a predictable and plausible outcome when painting over the RGBs on the canvas.

The function $g : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3$ is an approximate model of the pigment composition process in which foreground and background RGB colors combine to form a new RGB color. Our data driven color model sparsely represents g by a set of a-priori known composition results that are interpolated to generate the result for arbitrary foreground and background colors. These sparse samples come from the color chart provided by the user; a different color chart will result in a different function g and therefore a different color model. For example, an oil paint chart would produce a different g than a watercolor chart.

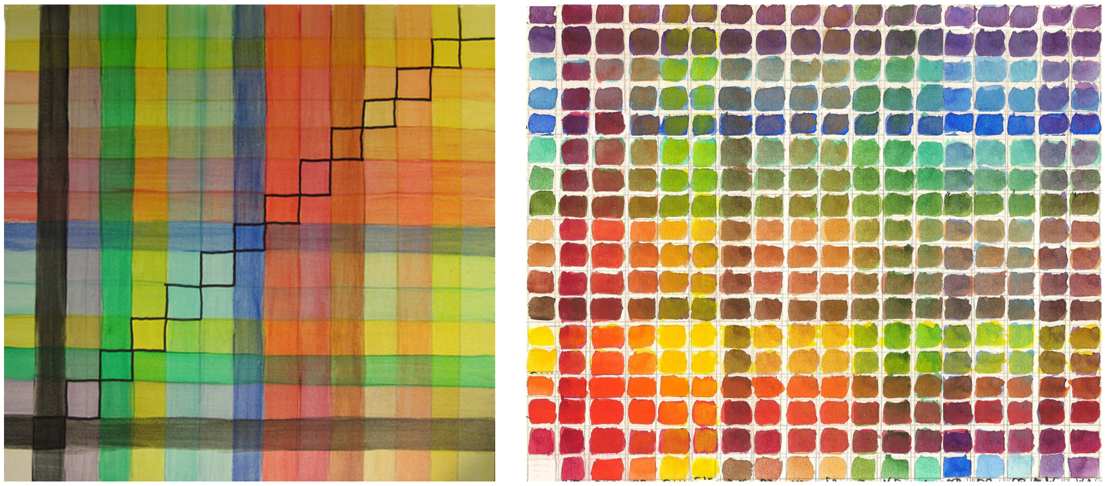
The nature of g's approximation is that it is not possible in general to find an analytical form of g such that $g(f(p_2), f(p_1)) = f(p_2 \circ p_1)$ for all p_1 and p_2. One reason is that g cannot express the behavior of *metamers* – different pigments $p_a \neq p_b$ that share the same perceived color: $f(p_a) = f(p_b)$. In general, these metamers will yield different results when painted over some other color p_c: $f(p_a \circ p_c) \neq f(p_b \circ p_c)$. This phenomenon cannot be expressed by g, which treats pigments only in terms of perceived color, i.e. $f(p_a) = f(p_b) \Leftrightarrow p_a = p_b$.

While g has these inherent ambiguities regarding metamers, we regard this limitation as a benefit of our approach. While metamerism is important for simulation of physical reality, it is a highly non-intuitive effect. Understanding the expected behavior of each combination of real pigments is a difficult part of learning to paint, which is why even expert painters often create color charts to help them predict these behaviors. Thus, our pigment model can improve the predictability of paint compositing by eliminating this difficult phenomenon.

While this discussion has focused on paint composition, in which one layer of pigment is applied atop another, it can be trivially extended to paint mixing, in which two pigments are blended into a single layer. In the KM model, pigment mixing is accomplished by interpolating between the two pigments' scattering and absorption coefficients, with resulting color $f(p_2 \otimes p_1)$. This can be modeled in the same manner, by a function $m(f(p_2), f(p_1)) : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3$, which is defined by a sparse set of color mixing samples. The primary difference between m and g is that m is symmetric, $m(b, a) = m(a, b)$, while g is not, $g(b, a) \neq g(a, b)$.

5.4 Data Collection and Processing

Artists create color charts to familiarize themselves with properties of different pigments. They experiment with glazing a layer of pigment on top of a dried layer to study the effect of compositing. In order to choose the right pigments to paint with, they observe the color chart and speculate on behaviors of pigments outside the chart. This process can be automated by prediction models, which learn from the known compositing examples in the chart and predict



(a) thin acrylic (b) watercolor

Figure 5.2: Physical color compositing charts available on-line.

the result given an arbitrary pair of new colors. To train our prediction models (see Section 5.5), we collect a set of physical and synthetic color charts.

Physical charts. As a common practice, a compositing chart is created as follows: Given n base pigments, divide the empty canvas into a grid of n^2 cells. As the background layer, paint each row i with base pigment i, keeping uniform distribution of the pigment for all cells of the row. After the background layer is completely dried, as the foreground layer, paint each column j with base pigment j. We collect compositing charts online, which are hand-painted by anonymous Internet users using the previously described procedure in several different natural media, including acrylic (Figure 5.2a), watercolor (Figure 5.2b) and marker pen (Figure 5.1a). In addition to the online charts, we also captured a watercolor chart ourselves in a controlled setting. Figure 5.3a shows the original capture. To better control paint thickness, we made uniform glazings of watercolor on transparent plastic, cut the plastic pieces to obtain background and foreground squares, and then captured every layered combination of these squares.

To process these physical charts, we undo the gamma correction and extract a single color from each grid cell by averaging pixels around the cell center. The cell borders are avoided to prevent the boundary effects, such as watercolor edge-darkening, to adversely influence the measured color. After processing, each grid cell contains a *composite color* – the result of glazing a particular foreground pigment over a background pigment or a *base color* – the result of applying a base pigment on the empty canvas (the leftmost column and the bottommost row in Figure 5.2 and 5.3). Figure 5.3b shows the processed chart. Though subtle, the compositing charts are not symmetric. The result of applying pigment A on top of B is slightly different from the result of the other way around. For example, in Figure 5.3b, column 4 within the composite matrix shows clear asymmetry from row 4, because the dark blue paint is very absorbent of light whereas other colors are more reflective.

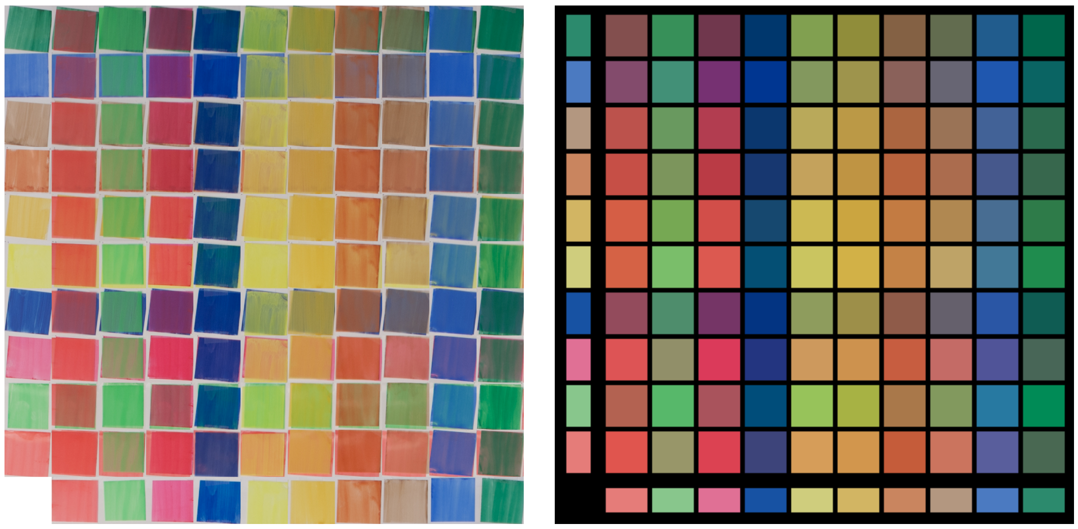

(a) physical chart (b) processed chart

Figure 5.3: We capture a gouache chart using uniform glazes on transparent plastic.

To simplify our inference problem and reduce the training parameters, we make some **assumptions** about the physical color charts. We assume that the thickness of the pigment layer is more or less constant for all rows and columns and that the charts are captured under roughly uniform lighting. The assumptions are reasonable, since they are the guiding principles for creating the charts. But they are hard to be enforced precisely and therefore small variations in brightness and layer thickness can be observed. For example, Figure 5.2b shows luminance variation within many cells due to the difficulty of depositing watercolor pigment uniformly.

Synthetic charts. Uniform lighting and constant layer thickness is desirable, but not guaranteed, due to the hand-painted nature of the charts. To understand how violations of these assumptions influence the prediction and to provide a baseline evaluation of our models, we algorithmically construct some synthetic charts. Specifically, we apply KM model (see Section 5.3) with the physically measured coefficients [106] of five acrylic pigments, including Phthalo Blue, Pyrrole Red, Hansa Yellow, Carbon Black and Titanium White. Each of the n base pigments are generated as a random mixture of the five primaries. The grid cells of the chart are calculated by two evaluations of the KM model. One evaluation of KM applies the background pigment onto the substrate and another applies the foreground pigment onto the rendered background. For KM, we use constant layer thickness for all pigment layers, standard illuminant D65 [102] and ideal white substrate (completely reflective at all wavelengths). For some natural media, the compositing behaviors might vary with different layer thickness. For example, for oil or acrylic, a thin layer of foreground pigment allows the background pigment to show through (Figure 5.4a), while a thick layer heavily obscures the background (fifth column from the right in Figure 5.4b) with the exception of using light-colored pigments as the foreground (last column from the right). Additionally, high light absorbance can cause certain

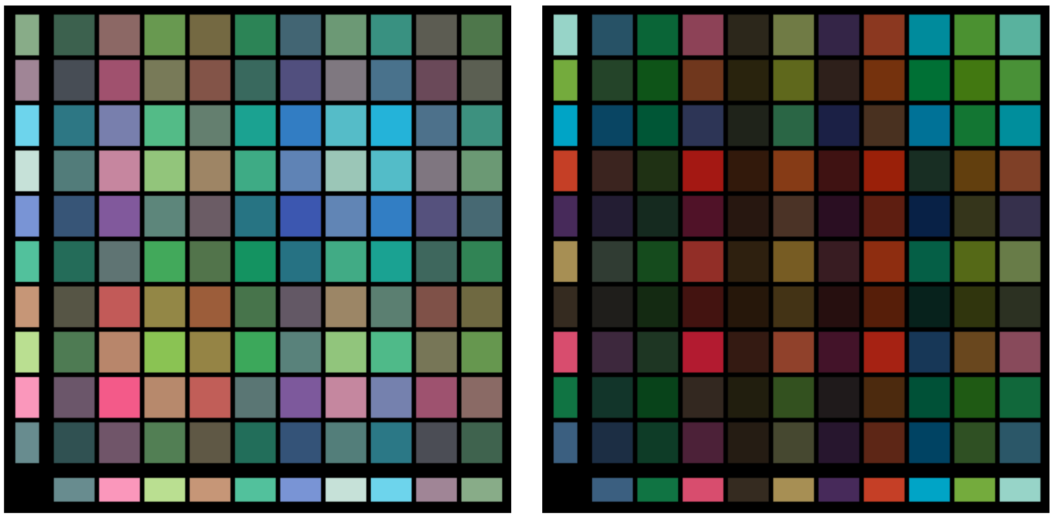

(a) thin pigment layer (b) thick pigment layer

Figure 5.4: Synthetic color compositing charts with different pigment layer thickness, created using acquired Kubelka-Munk coefficients and multispectral rendering.

colors to result in dark composites if used in either background or foreground (seventh column and row from the top and the right).

5.5 Prediction Models

Here, suppose a color chart is painted with N base pigments, then we have N base color samples, represented as B_i. And we have N^2 composite color samples, represented as C_{ij}, produced by layering base pigment j (foreground) on top of base pigment i (background). In the empty region of the chart, we can sample the substrate color $\mathcal{S}$. We convert all color values into the CIELAB color space for its perceptual linearity. Each color chart can be seen as a 6D point cloud of N^2 points, $\{(B_i, B_j) \mid i, j \in \{1 \dots N\}\}$. We use the standard quick hull algorithm [6] to calculate the convex hull, $\mathcal{H}$, of the point cloud, which is utilized in the following prediction models.

When a user selects a color chart to specify compositing behavior and then paints a given foreground color Q_f over a different background color Q_b, our goal is to predict the composite color Q_c. To address this goal, we propose three prediction models, *optimized alpha compositing*, *radial basis interpolation* and *optimized KM compositing*, with different pros and cons. Optimized alpha compositing (Section 5.5.1) is included as a baseline, which demonstrates the best possible performance of alpha blending to approximate the real paint compositing in the chart. Radial basis interpolation (Section 5.5.2) is a straightforward implementation of our model from Section 5.3 that interpolates and extrapolates the matrix g. Alternately Section 5.5.3 presents an optimized KM compositing scheme that refines the model for color charts that contain paint-like pigments. Each model contains a training phase conducted per color chart, and a prediction phase performed per query color.

5.5.1 Optimized Alpha Compositing

Alpha compositing is often used in digital painting, where an alpha value is specified by the user together with a RGB color. Choosing the alpha manually to reproduce the appearance of a target medium is challenging. For example, there is no alpha that can perfectly simulate the darkening effect of watercolor. Given a color chart, we can solve a constrained least squares problem to optimize the alpha that best explains the chart, which can then be used to predict the compositing result given new query colors. We include this method to provide a reference point for evaluating the performance of our other methods.

Training. Given an intrinsic color $\beta_i = (L_i, a_i, b_i)$ and a chosen alpha value α, alpha compositing on the substrate gives $b_i = \alpha\beta_i + (1-\alpha)\mathcal{S}$. With the same alpha and another intrinsic color β_j, recursive alpha compositing over the previous result gives $c_{ij} = \alpha\beta_j + (1-\alpha)b_i = \alpha\beta_j + (1-\alpha)\alpha\beta_i + (1-\alpha)^2\mathcal{S}$. Given a color chart, we minimize the difference between b_i and the measured base color B_i, as well as the difference between c_{ij} and C_{ij}, solving for α and $\beta = \{\beta_i, i \in \{1 \dots N\}\}$:

$$\alpha^*, \beta^* = \operatorname*{argmin}_{\alpha,\beta} E_a \tag{5.2}$$
$$\text{such that } \alpha \in [0,1], \text{ and } L_i \in [0,100], \forall i \in \{1 \dots N\}$$
$$E_a = \sum_i^N \|B_i - b_i\|^2 + w_c \frac{1}{N} \sum_i^N \sum_j^N \|C_{ij} - c_{ij}\|^2 \tag{5.3}$$

We use $w_c = 1$ to give equal importance to fitting both the base and the composite colors. We initialize α to be 0.5 and β to be random. The optimized alphas α^*, in the order of Table 5.1, are 0.58, 0.42, 0.49, 0.63, 0.31, 0.45, plausibly reflecting the thinner or thicker paint behaviors in the corresponding charts.

Prediction. From training, we obtain the optimized α^* that best describes the chosen color chart. Suppose the unknown intrinsic color of the foreground is $q_f = (L_f, a_f, b_f)$, then the following holds, $Q_f = \alpha^* q_f + (1-\alpha^*)\mathcal{S}$. The composite color $Q_c = (L_c, a_c, b_c)$ can be estimated as:

$$Q_c = Q_b(1-\alpha^*) + \alpha^* q_f = Q_b(1-\alpha^*) + Q_f - (1-\alpha^*)\mathcal{S} \tag{5.4}$$

However, the resulting luminance L_c might be outside the valid range $[0,100]$, when the 6D query is outside the convex hull of the chart. To address this, we perform a fast runtime optimization to make small adjustment to the alpha estimation. We use α^* and β^* to initialize the following optimization, solving for $\hat{\alpha}$, $\hat{\beta}$ and $\hat{q}_f$:

$$\hat{\alpha}, \hat{\beta}, \hat{q}_f = \operatorname*{argmin}_{\alpha,\beta,q_f} E_a + w_a \|Q_f - \alpha q_f - (1-\alpha)\mathcal{S})\|^2 \tag{5.5}$$
$$\text{such that } \alpha \in [0,1], \text{ and } L_f \in [0,100]$$

We set $w_a = 3$ to encourage the query color Q_f to be well fitted. The composite color can then be predicted by substituting $\hat{\alpha}$ and $\hat{q}_f$ into the equation (5.4).

5.5.2 Radial Basis Function Interpolation

The Radial Basis Functions (RBF) is one of the primary tools for interpolating multidimensional scattered data. RBF is suitable for our problem for the following reasons: it does not require the data points to lie on a regular grid, and gives good interpolation accuracy even when the data is sparse. RBF is defined as:

$$g(\hat{x}) = \sum_{t}^{n} \lambda_t \phi(\|\hat{x} - \hat{x}_t\|), \hat{x} \in \mathcal{R}^D \tag{5.6}$$

Training. A color chart can be interpreted as a 6D point cloud, with each data point corresponding to a 3D value. In our case, $\{\hat{x}_t\}$ in equation (5.6) corresponds to $\{(B_i, B_j) \mid i, j \in \{1 \dots N\}\}$. We have $n = N^2$ and $D = 6$. Given a sparse point cloud of several hundreds points, an arbitrary query is often near or outside its convex hull, $\mathcal{H}$. In these situations, the basic RBF interpolation is inaccurate. We use augmented RBF, which adds a polynomial term to the equation (5.6):

$$f(\hat{x}) = g(\hat{x}) + \sum_{m}^{M} \gamma_m p_m(\hat{x}) \tag{5.7}$$

We use the parameter-free linear kernel, $\phi(r) = r$ and the following polynomial basis, $\{p_m(\hat{x}) \mid m \in \{1 \dots M\}\} = \{1, \hat{x}, \hat{x}^2, ...\}$, $M = 2$ (that is, $\{1, \hat{x}\}$). The linear polynomial term can improve the interpolation accuracy, especially at and outside the boundary (second and third rows in Table 5.1), but little benefits can be obtained going beyond linear [146]. The polynomial term also allows simple extrapolation beyond the convex hull of the chart. Note that, our data values $\{C_{ij}\} = \{C_t \mid t \in \{1 \dots N^2\}\}$ are 3 dimensional, therefore we train three separate RBFs for each color dimension $\hat{f}(\hat{x}) = (f_l(\hat{x}), f_a(\hat{x}), f_b(\hat{x}))$. We write the following constraints in a matrix form and solve for the expansion parameters, λ and γ.

$$\hat{f}(\hat{x}_t) = C_t, \quad \forall t \in \{1 \dots N^2\} \tag{5.8}$$

$$\sum_{t}^{N^2} \lambda_t p_m(\hat{x}_t) = 0, \quad \forall m \in \{1 \dots M\} \tag{5.9}$$

Prediction. With the trained λ and γ, we predict the composite color by substituting the query point (Q_b, Q_f) into equation (5.7).

5.5.3 Optimized KM Compositing

Similar to Section 5.5.1, we attempt to train a parametric model to explain the color samples from the compositing chart. The difference is that this section assumes the underlying model to be Kubelka-Munk (KM), which is a powerful multi-spectrum model that more accurately simulates the appearance of physical pigments.

Training. We use $L = 8$ uniformly spaced wavelength samples to approximate the full visible light spectrum. We assume each base pigment of the chart is created by mixing different proportions of $K = 3$ distinct primary pigments, which corresponds to the common art practice that a color on the palette is usually created by mixing several different paints. This prediction model has the following parameters:

- $\sigma = \{s_l^k, a_l^k \mid l \in \{1 \dots L\}, k \in \{1 \dots K\}\}$: the scattering and absorption coefficients per wavelength of each primary pigment, in total $2LK$ parameters.
- $m = \{m_i \mid i \in \{1 \dots N\}$, where $m_i = \{m_i^k \mid k \in \{1 \dots K\}\}$: the proportions for combining the K primaries, for each of the N base pigments, in total NK parameters.
- $\xi = \{\xi_l \mid l \in \{1 \dots L\}\}$: the reflectance coefficients of the substrate at each sample wavelength, in total L parameters.

Given a mixing proportion m_i^k, the absorption and scattering coefficients of the mixed pigment i at each wavelength l is given by $A = \sum_k^K m_i^k a_l^k$ and $S = \sum_k^K m_i^k s_l^k$. We concisely represent the pigment mixing and the KM equation (Section 5.3), using the following notation $\mathbf{km}(\sigma, m_i, \xi)$, where the input are σ and ξ, the output is the reflectance spectrum, and we use constant layer thickness, $h = 1$. We represent the standard transform function that converts a reflectance spectrum to tristimulus CIELAB values as $\mathbf{lab}(\xi)$, where we use the standard illuminant D65. Then, each base color b_i and composite color c_{ij} can be evaluated by KM:

$$r_i = \mathbf{km}(\sigma, m_i, \xi) \tag{5.10}$$
$$b_i = \mathbf{lab}(r_i) \tag{5.11}$$
$$c_{ij} = \mathbf{lab}(\mathbf{km}(\sigma, m_j, r_i)) \tag{5.12}$$

We train the model parameters as follows:

$$\sigma^*, m^*, \xi^* = \underset{\sigma, m, \xi}{\operatorname{argmin}} E_k + w_c E_c + w_p E_p + w_s E_s \tag{5.13}$$

$$\text{such that } s_l^k \in [0, \infty), a_l^k \in [0, \infty), m_i^k \in [0, 1],$$

$$E_k = E_a + w_\xi \|\mathcal{S} - \mathbf{lab}(\xi)\|^2 \tag{5.14}$$

$$E_c = \frac{1}{K} \sum_i^N \left(1 - \sum_k^K m_i^k\right)^2 \tag{5.15}$$

$$E_p = \frac{N}{(L-1)K} \sum_k^K \left(\sum_l^{L-1} (s_l^k - s_{l+1}^k)^2 + (a_l^k - a_{l+1}^k)^2\right) \tag{5.16}$$

$$E_s = \frac{N}{L-1} \sum_l^{L-1} (\xi_l - \xi_{l+1})^2 \tag{5.17}$$

where E_k replaces the b_i and c_{ij} in equation (5.3) with the new definitions in equation (5.11) and (5.12), and adds another term to fit the measured substrate color $\mathcal{S}$. E_c, E_p and E_s serve as the regularization. E_c constrains that for each base pigment, the mixing proportions of the primaries sum up to 1. E_p encourages the scattering and absorption coefficient curves to be smooth for each primary. E_s encourages the reflectance curve of the substrate to be smooth. We initialize σ, m and ξ randomly within their respective valid range. We set $w_c = 1.5$, $w_p = w_s = 60$, $w_\xi = 1$. Rendering the optimized primaries on white substrate with KM equations results in three distinct colors that well represent the base colors in the marker chart, for example (inset figure).

Prediction. Given the query foreground color Q_f and the optimized parameters σ^*, ξ^*, we solve a least squares problem to predict the proportion, $m_q = \{m_q^k \mid k \in \{1 \dots K\}\}$, by which the primary pigments are mixed to generate the query foreground color. We could use the same least squares problem to solve for the mixture of primary pigments for the background color Q_b, but this would take twice as long and throw away previously computed data needlessly. Instead we cache and re-use the reflectance spectrum of the canvas due to previous strokes, $\mathcal{R} = \{r_l \mid l \in \{1 \dots L\}\}$. Then the composite color Q_c can be predicted as follows:

$$m_q^* = \underset{m_q}{\operatorname{argmin}} \|\mathbf{lab}(\mathbf{km}(\sigma^*, m_q, \xi^*)) - Q_f\|^2 \tag{5.18}$$

$$\text{such that } m_q^k \in [0,1], \forall k \in \{1 \dots K\},$$

$$Q_c = \mathbf{lab}(\mathbf{km}(\sigma^*, m_q^*, \mathcal{R})) \tag{5.19}$$

Prediction in this way is efficient, but can be problematic at times. The trained primary pigments σ^* define a color gamut (the range of colors that can be represented by mixing the primaries). When the query color Q_f is outside the gamut, the least squares solution might be far from the true answer. We use the convex hull of the color chart, $\mathcal{H}$, to approximate the color gamut.

In this case, we propose the following remedy. Similar to the idea in Section 5.5.1, we perform a runtime optimization taking into account the query color Q_f using the trained parameters, σ^* and m^*, as initialization and having the substrate spectrum fixed at ξ^*. We solve the following least squares problem, slightly modified from equations (5.13) and (5.14):

$$\hat{\sigma}, \hat{m}, \hat{m}_q = \underset{\sigma, m, m_q}{\operatorname{argmin}} E_q + w_c E_c + w_p E_p + w_s E_s \tag{5.20}$$

$$\text{such that } s_l^k \in [0,\infty), a_l^k \in [0,\infty), m_i^k \in [0,1], m_q^k \in [0,1],$$

$$E_q = E_k + w_k \|\mathbf{lab}(\mathbf{km}(\sigma, m_q, \xi^*)) - Q_f\|^2 \tag{5.21}$$

The term E_q adjusts the primary coefficients and the query mixing proportion to constrain the query color to lie inside the gamut. After the optimization, we predict the composite color by substituting $\hat{\sigma}$ and $\hat{m}_q$ into equation (5.19). We set $w_k = 3$ and initialize m_q using the mixing proportion of the base pigment that has the most similar color to the query.

	Average prediction error								
methods	marker			acrylic			watercolor		
alpha	30.75	29.20	29.99	10.92	9.02	9.88	21.23	18.37	19.80
rbf	16.78	9.01	12.90	9.90	4.85	7.14	9.19	7.05	8.12
rbf-nopoly	19.02	9.58	14.30	14.39	5.19	9.38	9.69	7.14	8.41
km	12.32	10.72	11.52	8.04	8.26	8.16	8.83	8.98	8.90
methods	gouache			thin synthetic			thick synthetic		
alpha	10.89	8.19	9.54	3.02	2.40	2.71	11.90	8.74	10.32
rbf	8.45	4.80	6.63	1.30	0.87	1.09	5.53	4.49	5.01
rbf-nopoly	9.60	5.08	7.34	6.55	2.89	4.72	7.54	5.33	6.43
km	9.17	6.68	7.93	0.41	0.34	0.38	2.51	2.04	2.28

Table 5.1: Hold-one-out evaluation results, broken up to show error for outside gamut samples, near gamut samples, and both sets combined.

5.6 Results and Discussion

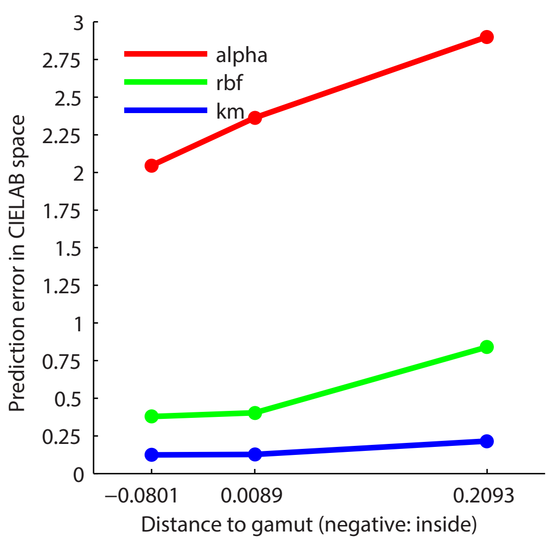
This section describes a number of quantitative and qualitative means to evaluate the quality of our pigment model, and discuss the impact of various design considerations on our results.

5.6.1 Quantitative Results

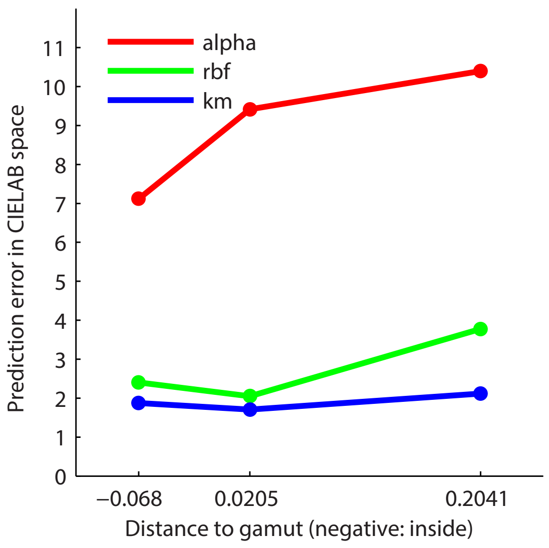
We report errors in terms of L2 difference in CIELAB colorspace, which represents the perceptual error in the predicted color. The just noticeable difference (JND) in LAB is between 1.0 and 2.3 [114] – differences less than this value are not generally perceivable to casual viewers.

Hold-one-out Evaluation. To evaluate the quality of our prediction, we would ideally compare the predicted color of a composition with the ground truth color for a given medium. However, ground truth data is difficult to come by particularly for color charts from the Internet. To work around this we use a hold-one-out methodology. If we have N base colors in a mixing chart, for color i we create a new pigment model on the $N-1$ other pigments (removing column and row i from the chart), and then use this reduced model to predict the composition results of color i with the other $N-1$ colors, which we have ground truth for (from the original chart of N base colors). Data from this evaluation is presented in Table 5.1. We find that for media of lower scattering properties (i.e. less opacity), namely marker and watercolor, RBF and KM both perform dramatically better than alpha blending. More highly scattering pigments such as acrylic and gouache are closer in performance, though still an improvement over alpha blending. Table 5.1 also shows that RBF with the polynomial term outperforms the basic RBF in all test cases, especially for the outside gamut samples.

Influence of Gamut. Because the pigment model is a 6D function and there are relatively few samples, in general all the samples are on the convex hull which defines the gamut of the model, or the volume within samples can be interpolated. This



(a) Thin paint in Figure 5.4a



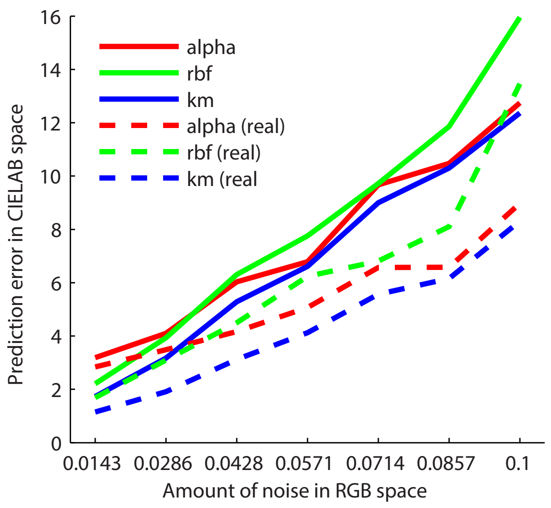
(b) Thick paint in Figure 5.4b

Figure 5.5: Gamut influence on prediction accuracy.

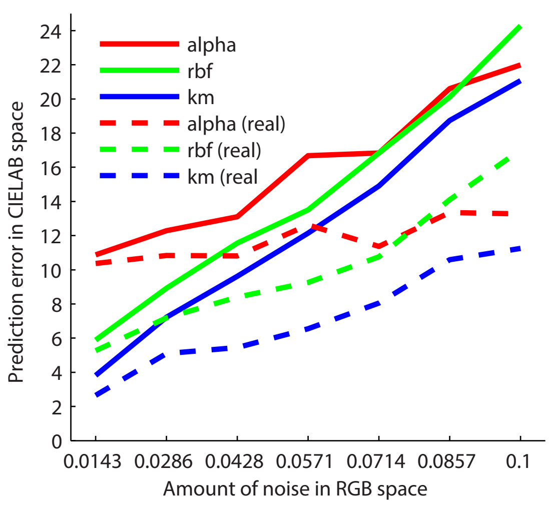
means that when performing the hold-one-out evaluation, the held out sample i is usually outside the gamut of the remaining $N-1$ samples. However, depending on the shape of the gamut, the sample i may be near the gamut (e.g. if i is an orange color and both red and yellow remain) or far outside the gamut (e.g. if i is blue and the remaining $N-1$ samples are all reds, oranges, and yellows). The prediction of near gamut samples is generally better than outside gamut as the prediction is closer to interpolation than extrapolation. For that reason, Table 5.1 splits the aggregate errors into near and outside gamut values, to see the difference. The important detail is that RBF tends to have a bigger discrepancy between near and outside gamut samples than KM, because RBF relies on extrapolation while KM fits a more restricted model that generalizes better.

To evaluate the error on inside gamut samples requires many more samples than we can reasonably acquire so we rely on synthetic charts made from measured KM coefficients of real paint pigments [106] and multispectral rendering. Figure 5.5 shows the relationship between distance from the gamut boundary for the thin and thick synthetic charts from Figure 5.4. We approximate the query distance from the gamut boundary (x axis value in Figure 5.5, negative means inside) by the Euclidean distance to the training convex hull normalized by the average distance of pairs of training samples. The performance of KM remains roughly the same, while RBF and alpha blending are both correlated with the signed distance from the gamut boundary.

Influence of Noise. The acquisition process for a physical chart involves a fair amount of noise. Most significantly, the swatches may not all be painted with the same pigment thickness and lighting may not be constant across the chart. We again rely on synthetic color charts to gain some understanding of the influence of noise on our results. Figure 5.6 shows prediction error for our thin and thick synthetic paint charts with increasing amounts of Gaussian noise (maximum [-0.1, 0.1]) added to each RGB channel. The graphs show error in two ways: the solid lines represent the same



(a) Thin paint in Figure 5.4a



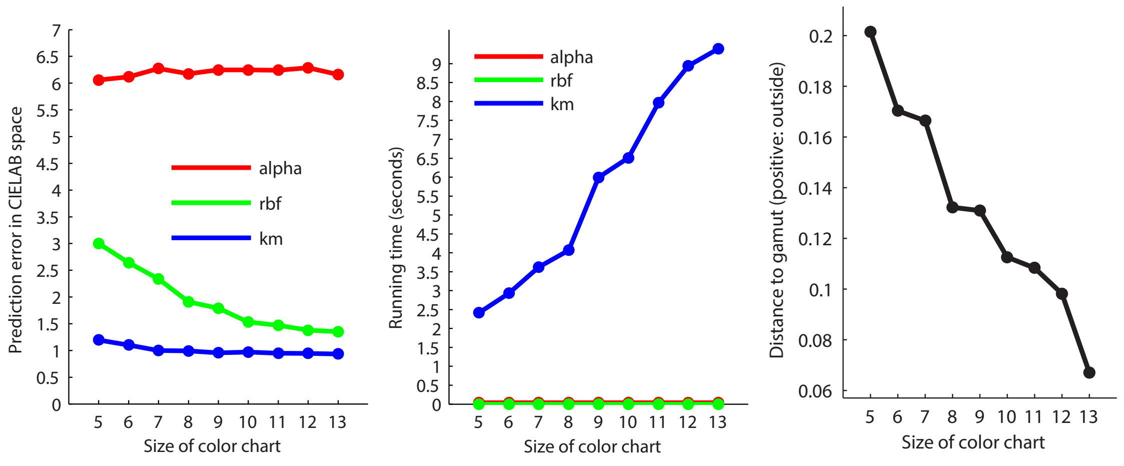
(b) Thick paint in Figure 5.4b

Figure 5.6: Noise influence on prediction accuracy.

hold-one-out error calculation as explained before. However, in this experiment, the ground truth values have noise added to them, which may increase the error over the "real" ground truth values which are the actual composition colors without noise. This "real" error is shown with dashed lines, and demonstrates that our hold-one-out methodology may overestimate the error between our models and actual pigments.

KM and RBF handle noise differently. RBF will simply try to reproduce whatever noise may be present in the color chart – systematic errors such as luminance changes across the chart can be reproduced well. KM on the other hand fits a single pigment model to all the data, effectively finding a best fit that minimizes the errors due to noise. While this may result in a higher apparent error for physical pigments due to our hold-one-out methodology, Figure 5.6 suggests that the actual error between the predictions and the real paints may be overall less.

Influence of Chart Size. It is interesting to examine the impact of chart size on performance, to determine how many pigments should be acquired (color charts online tend to have between 6 and 10 base colors). Figure 5.7 shows these results, where for each $N = \{5 \ldots 13\}$, we created 5 random synthetic charts with base colors sufficiently different from each other and evaluate the error for a persistent set of 10 random samples. What we find is that KM is quickly able to generalize a model and does not change significantly as the chart size increases, whereas RBF is initially worse than KM and slowly approaches its performance as the chart size improves. On the other hand, performance for alpha blending is roughly constant and very fast regardless of chart size, while KM's time to compute a prediction increases linearly with the number of pigments, as the size of the optimization problem becomes harder. We measure the running time using a desktop computer with Intel Core i7-3770, 3.40GHz. Finally, the average sample distance from the model gamut decreases also roughly linearly as the chart size increases. With a practical chart size, random queries are more likely to be outside the gamut, which underscores the importance of handling extrapolation.

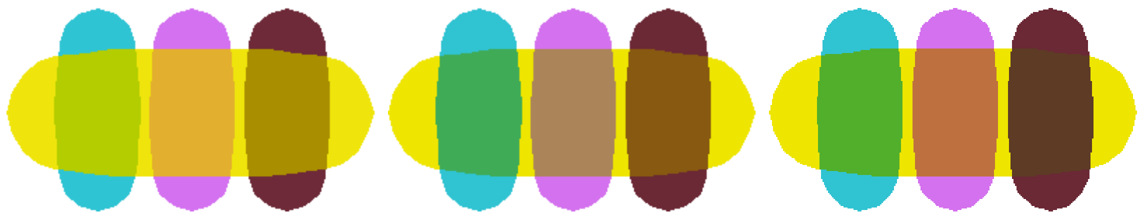


(a) Prediction (b) Performance (c) Gamut Distance

Figure 5.7: Experiments with synthetic charts.

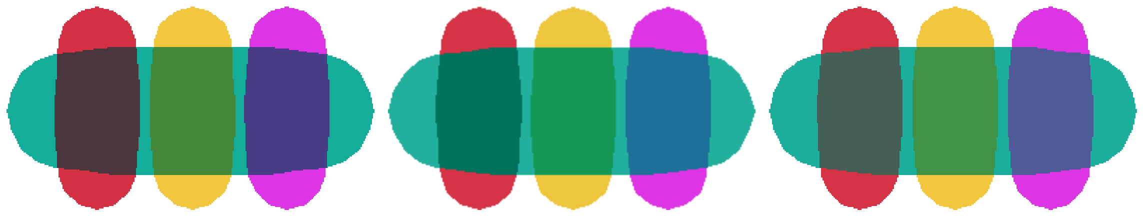
For KM it is also important to note that as the chart size grows, the amount of data increases faster than the number of unknowns. For N base colors, $L = 8$ wavelengths and $K = 3$ primary pigments (disregarding substrate optimization), we have $2LK + NK$ unknowns and $N^2 + 2N + 4K$ equations. Therefore, we want $48 + 3N < N^2 + 2N + 12 \Rightarrow N^2 - N - 36 > 0$, which means that when $N \geq 7$, we have more equations than unknowns. For $N < 7$, the system is underconstrained.

Influence of Parameters. The alpha and KM prediction methods both have a number of parameters to tailor their optimizations. However, we found that the results were not particularly sensitive to the specific values of the parameters. Similarly, optimizing the illuminant spectrum and the substrate reflectance spectrum, versus using standard values, does not make a large difference on the results, with the exception of the acrylic chart where the substrate is far from white. The impact of the regularization error terms is to reduce the number of optimization steps, which for KM is between 200 and 1000 iterations, and for alpha is between 20 to 60. A more significant factor in the quality of the prediction and speed of convergence is the amount of acquisition noise in the color charts – noisier charts require more optimization steps. Note that KM uses non-linear optimizations. To avoid getting trapped in a local minimum, off-line we perform the training several times with random initialization and pick the parameters giving the lowest training error. Then, initializing the runtime per-query optimization with the trained parameters improves the prediction performance and reduces the influence of local minima. Though the global minimum is hard to reach, we find that the prediction results of the synthetic charts are very close to the groundtruth suggesting that adequate local minimums are usually obtained. We solve the non-linear least squares problem using the standard Levenberg-Marquardt algorithm provided by Alglib [125]. To reduce the complexity of the runtime optimization, we can fix the mixing proportions m^* for the N base colors and only optimize the pigment coefficients, σ, for the K primaries and the mixing proportion, m_q, for the query .



(a) Alpha (b) RBF (c) KM

Figure 5.8: Prediction methods comparison. All strokes are synthesized using the marker chart (Figure 5.1a). Foreground: horizontal.



(a) Marker (b) Acrylic (c) Watercolor

Figure 5.9: KM prediction results using different color charts from Figures 5.1a, 5.2a, and 5.2b, respectively. Foreground: horizontal.

Finally, we have noticed that charts with more bold or vivid colors (e.g. marker, thick synthetic) have higher prediction errors than lighter, more pastel charts (e.g. watercolor, thin synthetic). We believe this is due to our hold-one-out methodology and the larger gamuts of these charts. When holding out pigment i, the absolute distance to the gamut of the $N-1$ pigments is going to be larger for charts that have a wider gamut. Reporting relative error, or normalizing the errors with respect to the gamut size, may facilitate direct comparison of the accuracy of different charts, but we have not found this to be necessary.

5.6.2 Qualitative Results

We present a qualitative comparison of our model's prediction quality in Figures 5.1 and 5.8. We choose saturated colors, the behaviors of which are familiar to casual viewers, to demonstrate our results using different prediction models. Figure 5.1 shows that for queries that are outside the training gamut (similar to, but different from the training base colors), the "best effort" alpha blending fails to reproduce the paint-like appearance. RBF improves the prediction of blue and yellow over red, but fails to produce green when applying yellow over blue, due to the fact that bright yellow is not represented in the chart, Figure 5.1a. KM, on the other hand, provides more robust extrapolation producing dark green as the result. Figure 5.8 shows more results of using the marker chart. Note that when layering bright yellow over cyan and pink, KM produces light green and orange respectively, which are closer to the behaviors in the chart. When using a darker brownish red as the background color, layering bright yellow results in dark brown making the yellow appear more transparent. We also compare our KM optimization results on the same

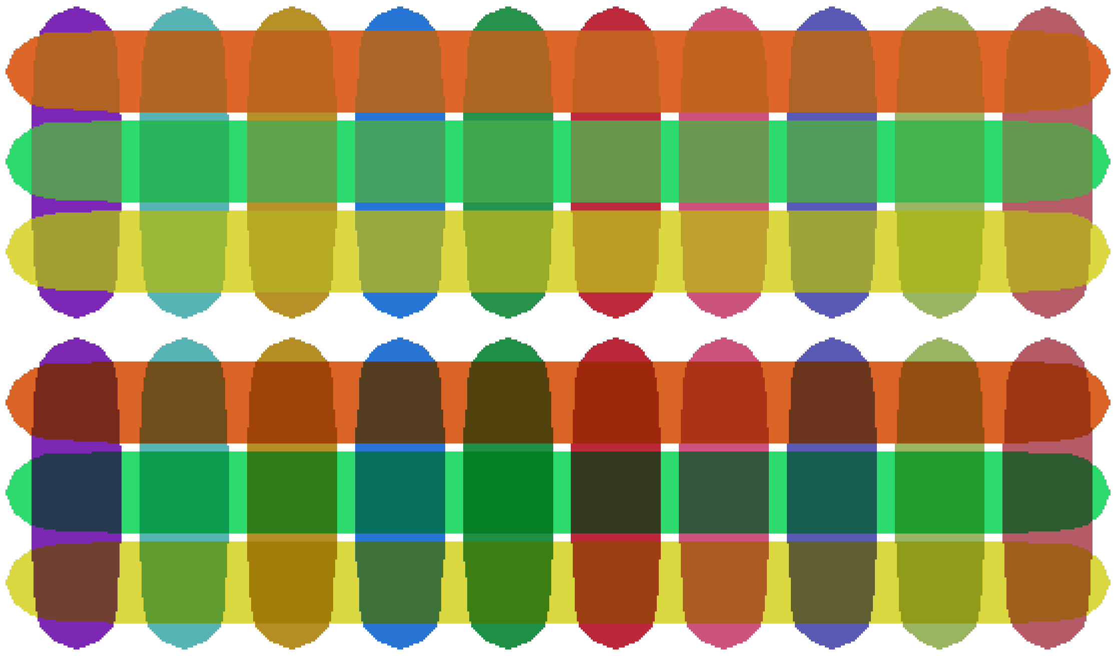


Figure 5.10: KM prediction results. Top matrix: synthesized with the gouache chart in Figure 5.3a. Bottom matrix: synthesized with the thick synthetic chart in Figure 5.4b. Foreground: horizontal.

set of queries, based on different charts. Figure 5.9 shows that different color charts produce perceptually distinct results. For example, cyan over red produces the light brownish color when using the watercolor chart (similar to $C_{2,8}$ from the top and right in Figure 5.2b) and darker brownish red when using the marker chart (similar to $C_{6,16}$ in Figure 5.1a). Note that though the saturated magenta is outside the gamut of the acrylic chart, paint-like composition is still obtained. Figure 5.10 demonstrates more results of KM based on two different training charts. We experiment with less saturated colors that are different from the training base colors and show that the prediction results faithfully simulate the high opacity and darkening effects in Figure 5.3a and 5.4b respectively.

We also show a non-paint-like model in Figure 5.11. We use Adobe Photoshop to create a color chart with RGB colors and the additive blend mode, which simulates the behavior of light (e.g. red and green makes yellow). This chart is not well modeled by alpha blending or KM, as neither has the ability to represent this type of behavior, but RBF handles it well, as can be seen in the simple painting in Figure 5.11, where colors not in the chart are repeatedly laid atop one another until the color is completely saturated. The hold-one-out evaluation on this chart produces combined error values of 50.97, 19.92, and 51.74 for Alpha, RBF, and KM respectively.

5.6.3 Controlling Opacity

Our pigment model does not include an explicit parameter for "alpha" in the regular digital painting sense, in which a user can select an RGB and set the alpha

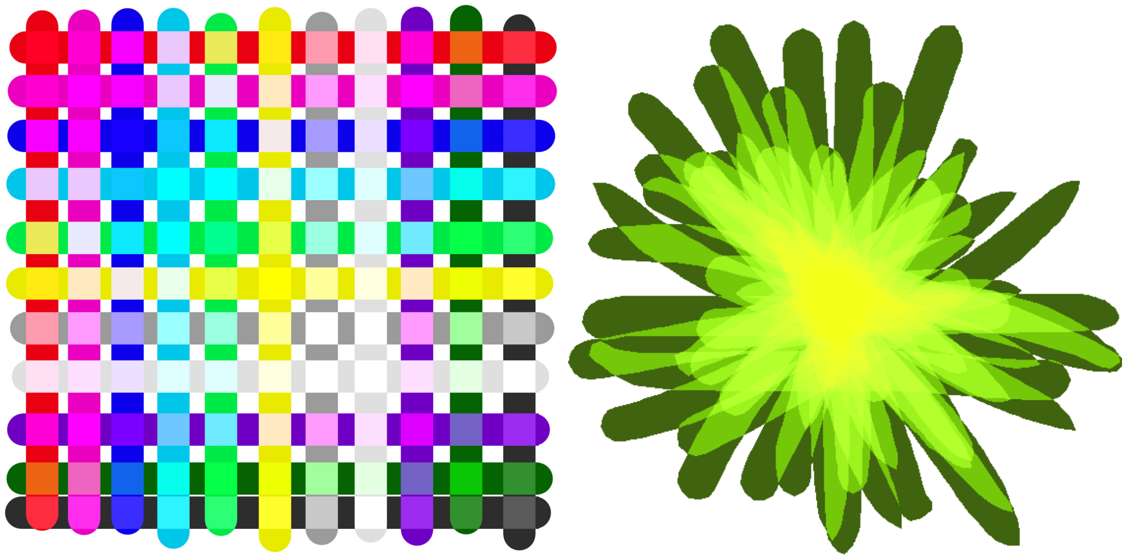


Figure 5.11: A non-paint-like color chart that exhibits additive blending, created in Adobe Photoshop. RBF is used to create the strokes on the right that reproduce the additive effect.

value to determine how much of the background shows through. This information is however implicitly encoded in the color chart, as any set of paint composition behaviors can be represented. For example, one color chart could use thinned paint while another could use thick paint, and the different models would then include the corresponding transparent or opaque composition behavior. Furthermore, a single color chart could include samples from a dark red pigment applied thickly, appearing dark red, and thinly, appearing light pink – the model would then produce opaque results for dark red colors, and translucent results for light pink colors. See Figure 5.12 for an example.

The main limitation of this approach is that, because the pigment model cannot represent metamers, some flexibility is lost. For example, it is not possible to have both a thin, translucent stroke of dark red pigment that appears light pink, as well as a thick opaque stroke of light pink pigment, as these strokes would have the same RGB value in the color chart and so would conflict with one another.

In the implementation of our pigment model in a painting application, it may still be desirable to support some type of alpha control that operates in a way that users are accustomed. It would be straightforward to provide such a control which would interpolate between the background color and the composited result. That is, at $\alpha = 1$, the resulting color is the pigment model's prediction, at $\alpha = 0$, the result is the background color, and at $\alpha = 0.5$, it is the average of the two, etc. Another approach would be to use α to interpolate between two charts, one with a thin application of the foreground pigment and the other with thicker paint applied.

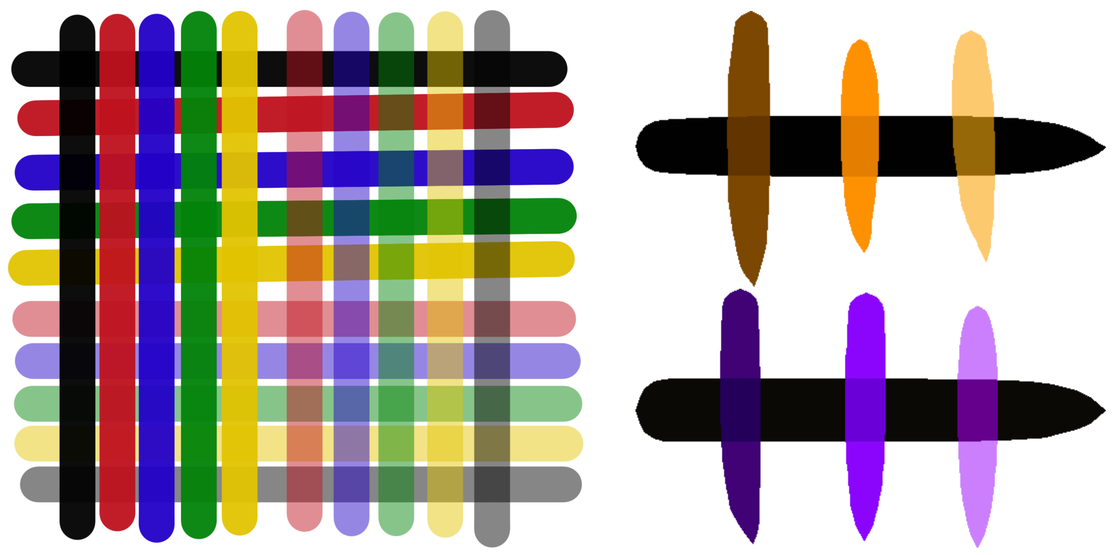


Figure 5.12: A color chart (left) with both opaque and translucent compositions, and some strokes (right) painted with the resulting pigment model (foreground: vertical). Note that lighter colored strokes are more transparent, while darker colored strokes are more opaque. Also all colors do not appear in the chart.

5.7 Limitations and Future Work

This chapter raises a number of possible areas for future work.

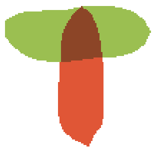
Modeling noise. We do not directly model noise in the acquisition process due to variation in the paints, thickness, opacity, lighting etc. A more sophisticated fitting scheme might be able to model these components and therefore respond to them more robustly.

Support for paint mixing charts. Section 5.3 discusses briefly how the chart-based approach we use for compositing could easily be extended to paint mixing. Nevertheless, any digital painting application that allows mixing needs controls for *both* operations. The cross product of all mixed and composited colors is obviously a substantially larger space; it may necessitate acquisition of considerably more data and/or require more care from the UI perspective.

Online painting application. Our prototype implementation is for offline experiments, and we have not yet integrated our approach into an online painting system. Typical digital painting systems have support for brush and paper textures, and these effects interact with the compositing rules – which may require at least engineering if not further research to implement convincingly.

Simpler model outside convex hull. We have found that the compositing behavior is high-quality when pigment combinations are selected that are within the convex hull of the data in the chart. As colors move further away the ability for our methods to extrapolate well breaks down. Of course it is always possible to add more exemplars in the region one wishes to paint, but we also speculate that it might be possible to smoothly transition to some simpler, plausible model far away from the exemplar data.

5.8 Conclusion


his chapter presents RealPigment [90], a data-driven color model that can be adopted by existing painting systems for more realistic color compositing effects. We propose and compare a class of compositing methods and show their respective advantages. Using our framework, novice users can take a picture of a color chart and then paint strokes with it. The query strokes allow arbitrary color and emulate the compositing behaviors of the chart. We conduct quantitative analysis to evaluate our color model and develop a simple painting program that demonstrates the applications.

Chapter 6

Conclusion

6.1 Summary

This thesis aims at designing efficient algorithms and intuitive user interfaces for novices to more easily create visually-compelling drawings and paintings exploiting the power of example. In particular, we introduce a data-driven painting paradigm, where the painting process is factorized into a set of orthogonal components: 1) Input trajectory; 2) Input hand-pose; 3) Shape; 4) Inter-stroke interaction texture; 5) Inter-stroke interaction color. We describe four prototype systems to demonstrate that each of the components can be synthesized efficiently and independently based on small sets of decoupled exemplars.

HelpingHand (Chapter 2) shows that the input trajectory from the user can be stylized and the hand-pose, if missing, can be synthesized by example. It eliminates the need of high-end input device and allows novices to make pretty handwritings and line drawings without substantive trainings. RealBrush (Chapter 3) implements a concrete factorized synthesis pipeline to support painting in a wide variety of physical media. We show that "shape" and "inter-stroke interaction" can be faithfully synthesized by separate sets of exemplars in an efficient manner. DecoBrush (Chapter 4) generalizes the "shape" to include strongly structured vector patterns. We show that the sketch-based user interface can be used for efficient design of vector art. Finally, we present RealPigment (Chapter 5), a data-driven color model that makes use of color compositing charts to realistically predict the resulting color of "inter-stroke interaction".

6.2 Future Work

Previous chapters have summarized the possible extensions to the respective systems (Section 2.9, 3.10, 4.7 and 5.7). This section outlines a few long term research directions with related to digital art.

Data-driven art creation. This thesis only scratches the surface of data-driven art creation. There are many remaining challenges to be addressed in the future. For example, the synthesis of stroke interactions can take into account local context and canvas history. For trajectory stylization, the shape of the new trajectory should depend on the existing curves in the local proximity. For structured decorative art, the "growing" of new structures might be adaptive to the existing structures. Future research in digital art creation can take a hybrid approach to combine physical simulation and data-driven techniques. For example, one can use physical simulation to guide the exemplar selection during synthesis or use the acquired exemplars to improve the design of physical simulation and train the parameters best suited for a target style. The data-driven painting concept can be extended to the time domain. For example, watercolor strokes have a dynamic diffusion effect that can be captured as short videos and displayed along the course of painting a new stroke. One might also provide users with a way to paint with videos of flowing waterfall, shaking trees, falling snows etc, creating animations by example using sketch-based interface. With the increasing computational power and memory capacity, we may break down the concept of a discrete stroke, and directly use the paintings on the Internet as exemplars for synthesis. We can synthesize the appearance of new "regions" by searching for the most suitable regions within the paintings. We need to address the challenge of region matching as well as blending different exemplar regions that have different colors and lightings. Paintings contain infinite number of possible "stroke regions" and implicitly embody the stroke interactions. Using paintings as exemplars will potentially improve the complexity and variety in the synthesis results.

Visual data analysis and human perception. Of equivalent importance to art creation, art understanding should be emphasized in future research. We can utilize the advances in crowd-sourcing, statistics and machine learning to enhance the scientific understanding of art and build better model for human perception. We can design and conduct user studies to investigate questions like "what makes computer-generated imagery look fake? What visual artifacts are the most noticeable to human eyes?", "what defines the style of Van Gogh?", "How do human interpret the illusion paintings?", "where do people look in a painting?", "How do image and video caricature influence human perception about the subject?", etc. HelpingHand and RealBrush conducted user studies to evaluate the synthesis results based on pairwise image comparisons (Section 2.7.2 and Section 3.8). Future work can be done to further formulate such a systematic evaluation framework for more general NPR research, and the study statistics can be used to develop better models of human perception and suggest future research directions.

Better painting interface. We believe a more radical change can be made to the current digital painting interface. We envision a painting interface composed of mixed reality. The users draw with the physical instrument of their choice on actual substrate surface. As they draw, both the gesture of the user and the physical appearance of the strokes are captured by cameras and displayed on the screen for immediate visual feedback and digital editing. This interface takes advantage of people's familiarity and comfort with existing physical drawing tools and provides all the richness of the medium as well as true haptic feedback. On the other hand, it can potentially integrate digital synthesis into the physically captured imagery, perhaps through a data-driven system like RealBrush, but taking the physical capture as exemplars. With ever increasing computational power and network bandwidth, online collaborative art creation is not far out of reach. A generic interface will enable physical and digital artists to work together and jointly contribute to large scale, ever-evolving art pieces that were not possible before.

Bibliography

[1] Zainab AlMeraj, Craig S. Kaplan, and Paul Asente. Patch-based geometric texture synthesis. In *Proceedings of the Symposium on Computational Aesthetics*, CAE '13. ACM, 2013.

[2] Dustin Anderson and Zoë Wood. User driven two-dimensional computer-generated ornamentation. In *Proc. International Symposium on Advances in Visual Computing*, 2008.

[3] Ryoichi Ando and Reiji Tsuruno. Segmental brush synthesis with stroke images. In *Proc. Eurographics – Short papers*, 2010.

[4] Michael Ashikhmin. Synthesizing natural textures. In *Proceedings of Symposium on Interactive 3D Graphics*. ACM, 2001.

[5] Autodesk. Sketchbook pro 6. http://www.autodesk.com/, 2012.

[6] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 1996.

[7] Pascal Barla, Simon Breslav, Joëlle Thollot, François Sillion, and Lee Markosian. Stroke pattern analysis and synthesis. In *Computer Graphics Forum (Proc. of Eurographics 2006)*, 2006.

[8] Connelly Barnes, Dan Goldman, Eli Shechtman, and Adam Finkelstein. The patchmatch randomized matching algorithm for image manipulation. *Commun. ACM*, 54:103–110, November 2011.

[9] William Baxter and Naga Govindaraju. Simple data-driven modeling of brushes. In *I3D*, pages 135–142, 2010.

[10] William Baxter and Ming Lin. A versatile interactive 3d brush model. In *Pacific Graphics*, pages 319–328, 2004.

[11] William Baxter, Yuanxin Liu, and Ming C. Lin. A viscous paint model for interactive applications: Research articles. *Computer Animation and Virtual Worlds*, 15(3-4):433–441, 2004.

[12] William Baxter, Vincent Scheib, Ming Lin, and Dinesh Manocha. Dab: Interactive haptic painting with 3d virtual brushes. In *Proceedings of SIGGRAPH*, pages 461–468, 2001.

[13] William Baxter, Jeremy Wendt, and Ming C Lin. IMPaSTo: a realistic, interactive model for paint. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 45–148, 2004.

[14] William V. Baxter. *Physically-based Modeling Techniques for Interactive Digital Painting*. Ph.D. Thesis, University of North Carolinal at Chapel Hill, Chapel Hill, North Carolina, 2004.

[15] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *Trans. on Patt. Anal. and Mach. Int.*, 24:509–522, 2001.

[16] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. State-of-the-art report on temporal coherence for stylized animations. In *Computer Graphics Forum*, 2011.

[17] Roy S. Berns and E. René De la Rie. The effect of the refractive index of a varnish on the appearance of oil paintings. *Studies in Conservation*, 48(4):251–262, 2003.

[18] Yu-Sheng Chen, Jie Shie, and Lieu-Hen Chen. A npr system for generating floral patterns based on l-system. *Bulletin of Networking, Computing, Systems, and Software*, 1(1), 2012.

[19] Nelson Chu, William Baxter, Li-Yi Wei, and Naga Govindaraju. Detail-preserving paint modeling for 3d brushes. In *NPAR*, pages 27–34, 2010.

[20] Nelson Chu and Chiew-Lan Tai. Real-time painting with an expressive virtual chinese brush. *Computer Graphics and Applications*, 24(5):76–85, 2004.

[21] Nelson Chu and Chiew-Lan Tai. Moxi: Real-time ink dispersion in absorbent paper. In *Proceedings of SIGGRAPH*, pages 504–511, 2005.

[22] Forrester Cole, Kevin Sanik, Doug DeCarlo, Adam Finkelstein, Thomas Funkhouser, Szymon Rusinkiewicz, and Manish Singh. How well do line drawings depict shape? In *SIGGRAPH*, volume 28, aug 2009.

[23] Corel. Painter 12. http://www.corel.com/, 2011.

[24] Microsoft Corporation. Microsoft freshpaint. http://www.microsoft.com/en-us/freshpaint/, 2013.

[25] Cassidy Curtis, Sean Anderson, Joshua Seims, Kurt Fleischer, and David Salesin. Computer-generated watercolor. In *Proceedings of SIGGRAPH*, pages 421–430, 1997.

[26] S. Darabi, E. Shechtman, C. Barnes, D.B. Goldman, and P. Sen. Image melding: combining inconsistent images using patch-based synthesis. *ACM Transactions on Graphics*, 31(4):82:1–82:10, 2012.

[27] James Davis, Maneesh Agrawala, Erika Chuang, Zoran Popović, and David Salesin. A sketching interface for articulated figure animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 320–328. Eurographics Association, 2003.

[28] Ambient Design. Artrage 3.5. http://www.ambientdesign.com/, 2011.

[29] Stephen DiVerdi. An abstract model of digital painting: Theory and applications. *Submitted to Transactions on Visualization and Computer Graphics*, 2013.

[30] Stephen DiVerdi. A brush stroke synthesis toolbox. In Paul Rosin and John Collomosse, editors, *Image and Video-based Artistic Stylisation*. Springer, 2013.

[31] Stephen DiVerdi, Aravind Krishnaswamy, and Sunil Hadap. Industrial-strength painting with a virtual bristle brush. In *Virtual Reality Software and Technology*, pages 119–126, 2010.

[32] Stephen DiVerdi, Aravind Krishnaswamy, Radomír Měch, and Daichi Ito. A lightweight, procedural, vector watercolor painting engine. In *Proceedings of I3D*, pages 63–70, 2012.

[33] Stephen DiVerdi, Aravind Krishnaswamy, Radomír Měch, and Daichi Ito. Painting with polygons: A procedural watercolor engine. *Transactions on Visualization and Computer Graphics*, 19(5), 2013.

[34] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei A Efros. What makes paris look like paris? *ACM Transactions on Graphics (TOG)*, 31(4):101, 2012.

[35] Per Edström. Examination of the revised Kubelka-Munk theory: considerations of modeling strategies. *Journal of the Optical Society of America A*, 24(2):548–556, 2007.

[36] Per Edström. *Mathematical Modeling and Numerical Tools for Simulation and Design of Light Scattering in Paper and Print*. PhD thesis, Mid Sweden University, 2007.

[37] Per Edström. Next generation simulation tools for optical properties in paper and print. In *International Conference MOdeling and Simulation in the Pulp and Paper Industry*, pages 156–169, 2008.

[38] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001.

[39] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, Corfu, Greece, September 1999.

[40] Katrin Franke, Lambert Schomaker, and Mario Köppen. Pen force emulating robotic writing device and its application. In *Workshop on Advanced Robotics and its Social Impacts*, pages 36–46, 2005.

[41] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. In *ACM Transactions on Graphics (TOG)*. ACM, 2004.

[42] R. G. Giovanelli. Reflection by semi-infinite diffusers. *Optica Acta: International Journal of Optics*, 2(4), 1995.

[43] Andrew S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994.

[44] Bruce Gooch and Amy Gooch. *Non-photorealistic rendering*. AK Peters, Ltd., 2001.

[45] Todd Goodwin, Ian Vollick, and Aaron Hertzmann. Isophote distance: a shading approach to artistic stroke thickness. In *Non-Photorealistic Animation and Rendering*, pages 53–62, 2007.

[46] Nathan Gossett and Baoquan Chen. Paint inspired color mixing and compositing for visualization. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 113–118, 2004.

[47] H. Granberg and Per. Edström. Quantification of the intrinsic error of the Kubelka-Munk model caused by strong light absorption. *Journal of Pulp and Paper Science*, 29(11):386–390, 2003.

[48] James Hays and Alexei A Efros. Scene completion using millions of photographs. In *ACM Transactions on Graphics (TOG)*. ACM, 2007.

[49] Harry G. Hecht. A comparison of the Kubelka-Munk, Rozenberg, and Pitts-Giovanelli methods of analysis of diffuse reflectance for several model systems. *Applied Spectroscopy*, 37(4):315–403, 1983.

[50] Aaron Hertzmann. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, 2003.

[51] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M Seitz. Curve analogies. In *Rendering Techniques*, pages 233–246, 2002.

[52] Geoffrey Hinton and Vinod Nair. Inferring motor programs from images of handwritten digits. In *Advances in Neural Information Processing*, pages 515–522, 2005.

[53] Øyvind Hjelle. Approximation of scattered data with multilevel b-splines. Technical report, SINTEF, 2001.

[54] Donald House and Mayank Singh. Line drawing as a dynamic process. In *Pacific Graphics*, pages 351–360, 2007.

[55] Eugene Hsu, Sommer Gentry, and Jovan Popović. Example-based control of human motion. In *Symposium on Computer Animation*, pages 69–77, 2004.

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

[57] Siu Chi Hsu and Irene H. H. Lee. Drawing and animation using skeletal strokes. In *Proceedings of SIGGRAPH*, pages 109–118, 1994.

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

[59] Takeo Igarashi and John F. Hughes. Smooth meshes for sketch-based freeform modeling. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, I3D '03. ACM, 2003.

[60] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, 1999.

[61] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. In *SIGGRAPH*, 2005.

[62] Takashi Ijiri, Radomír Měch, Takeo Igarashi, and Gavin S. P. Miller. An example-based procedural system for element arrangement. *Comput. Graph. Forum*, 27(2):429–436, 2008.

[63] Takashi Ijiri, Shigeru Owada, Makoto Okabe, and Takeo Igarashi. Floral diagrams and inflorescences: interactive flower modeling using botanical structural constraints. In *ACM Transactions on Graphics (TOG)*. ACM, 2005.

[64] Adobe Systems Inc. Adobe eazel for photoshop cs5. http://itunes.apple.com/us/app/adobe-eazel-for-photoshop/id421302663, 2011.

[65] Adobe Systems Inc. Illustrator cs6. http://www.adobe.com/illustrator/, 2012.

[66] Adobe Systems Inc. Photoshop cs6. http://www.adobe.com/, 2012.

[67] Henry Kang, Seungyong Lee, and Charles K. Chui. Coherent line drawing. In *Proc. of Non-photorealistic Animation and Rendering*. ACM, 2007.

[68] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006.

[69] Rubaiat Habib Kazi, Takeo Igarashi, Shengdong Zhao, and Richard Davis. Vignette: interactive texture design and manipulation with freeform gestures for pen-and-ink illustration. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, pages 1727–1736. ACM, 2012.

[70] Mikyung Kim and Hyun Joon Shin. An example-based approach to synthesize artistic strokes using graphs. *Computer Graphics Forum*, 29(7):2145–2152, 2010.

[71] Vasiliki Kokla, Alexandra Psarrou, and Vassilis Konstantinou. Ink recognition based on statistical classification methods. In *Document Image Analysis for Libraries, 2006. DIAL'06. Second International Conference on*, pages 11–pp, 2006.

[72] Jonathan Konieczny, John Heckman, Gary Meyer, Mark Manyen, Marty Rabens, and Clement Shimizu. Automotive spray paint simulation. In *Proceedings of the International Symposium on Visual Computing*, pages 998–1007, 2008.

[73] Jonathan Konieczny and Gary Meyer. Airbrush simulation for artwork and computer modeling. In *Proceedings of Non-Photorealistic Animation and Rendering*, pages 61–69, 2009.

[74] Luca Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *SIGGRAPH*, pages 473–482, 2002.

[75] Paul Kubelka. New contributions to the optics of intensely light-scattering material, part i. *Journal of the Optical Society of America*, 38(5):448–457, 1948.

[76] Paul Kubelka. New contributions ot the optics of intensely light-scattering material, part ii. *Journal of the Optical Society of America*, 44(4):330–335, 1954.

[77] Paul Kubelka and Franz Munk. An article on optics of paint layers. *Z. Tech. Phys*, 12:593–601, 1931.

[78] Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.

[79] J Kyprianidis, John Collomosse, Tinghuai Wang, and Tobias Isenberg. State of the art: A taxonomy of artistic stylization techniques for images and video. *IEEE Transactions on Visualization and Computer Graphics*, 2013.

[80] Pierre-Edouard Landes, Bruno Galerne, and Thomas Hurtut. A shape-aware model for discrete texture synthesis. In *Computer Graphics Forum*, volume 32, pages 67–76. Wiley Online Library, 2013.

[81] Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. Interactive control of avatars animated with human motion data. In *ACM Transactions on Graphics (TOG)*. ACM, 2002.

[82] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *ACM Trans. Graph.*, 24(3):777–786, jul 2005.

[83] Sylvain Lefebvre and Hugues Hoppe. Appearance-space texture synthesis. In *SIGGRAPH*, pages 541–548. ACM, 2006.

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

[85] Tommer Leyvand, Daniel Cohen-Or, Gideon Dror, and Dani Lischinski. Data-driven enhancement of facial attractiveness. *ACM Transactions on Graphics (TOG)*, 27(3):38, 2008.

[86] Kai Li, Feng Xu, Jue Wang, Qionghai Dai, and Yebin Liu. A data-driven approach for facial expression synthesis in video. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 57–64. IEEE, 2012.

[87] Guodong Liu. *A data-driven, piecewise linear approach to modeling human motions.* PhD thesis, University of North Carolina, 2007.

[88] Jingwan Lu, Connelly Barnes, Stephen DiVerdi, and Adam Finkelstein. Realbrush: painting with examples of physical media. *ACM Transactions on Graphics (TOG)*, 32(4):117, 2013.

[89] Jingwan Lu, Connelly Barnes, Connie Wan, Paul Asente, Radomír Měch, and Adam Finkelstein. Decobrush: Drawing structured decorative patterns by example. *ACM Transactions on Graphics (TOG)*, 2014.

[90] Jingwan Lu, Stephen DiVerdi, Connelly Barnes, and Adam Finkelstein. Realpigment: Paint compositing by example. *Submitted to Non-photorealistic Animation and Rendering (NPAR)*, 2014.

[91] Jingwan Lu, Fisher Yu, Adam Finkelstein, and Stephen DiVerdi. Helpinghand: Example-based stroke stylization. *ACM Transactions on Graphics (TOG)*, 31(4):46, 2012.

[92] The Kiet Lu and Zhiyong Huang. A GPU-based method for real-time simulation of eastern painting. In *Proceedings of GRAPHITE*, pages 111–118, 2007.

[93] Michal Lukáč, Jakub Fišer, Jean-Charles Bazin, Ondřej Jamriška, Alexander Sorkine-Hornung, and Daniel Sýkora. Painting by feature: Texture boundaries for example-based image creation. *ACM Trans. Graph.*, 32(4):116:1–116:8, jul 2013.

[94] Chongyang Ma, Li-Yi Wei, and Xin Tong. Discrete element textures. In *ACM Transactions on Graphics (TOG)*, volume 30, page 62. ACM, 2011.

[95] James McCrae and Karan Singh. Sketching piecewise clothoid curves. In *Sketch-Based Interfaces and Modeling*, 2009.

[96] James McCrae and Karan Singh. Neatening sketched strokes using piecewise french curves. In *Sketch-Based Interfaces and Modeling*, pages 141–148, 2011.

[97] Radomír Měch and Gavin Miller. The *Deco* framework for interactive procedural modeling. *Journal of Computer Graphics Techniques (JCGT)*, 1(1):43–99, Dec 2012.

[98] Xiao-Feng Mi, Min Tang, and Jin-Xiang Dong. Droplet: A virtual brush model to simulate chinese calligraphy and painting. *Journal of Computer Science & Technology*, 19(3):393–404, 2004.

[99] Xiao-Feng Mi, Jie Xu, Min Tang, and Jin-Xiang Dong. The droplet virtual brush for chinese calligraphic character modeling. In *Proceedings of the Workshop on Applications of Computer Vision*, pages 330–334, 2002.

[100] Mahnaz Mohammadi and Roy S. Berns. Testing instrumental-based color matching for artist acrylic paints. Technical report, Rochester Institute of Technology, College of Science, Munsell Color Science Laboratory, 2006.

[101] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, pages 331–340, 2009.

[102] N. Ohta and A. R. Robertson. Chapter 3. CIE standard colorimetric system. In *Colorimetry: Fundamentals and Applications*. John Wiley & Sons, Ltd, Chichester, UK, 2006.

[103] Makoto Okabe, Shigeru Owada, and Takeo Igarash. Interactive design of botanical trees using freehand sketches and example-based editing. In *Computer Graphics Forum*. Wiley Online Library, 2005.

[104] Yuta Okabe, Suguru Saito, and Masayuki Nakajima. Paintbrush rendering of lines using HMMs. In *GRAPHITE*, pages 91–98, 2005.

[105] Naoto Okaichi, Henry Johan, Takashi Imagire, and Tomoyuki Nishita. A virtual painting knife. *The Visual Computer*, 24(7):753–763, 2008.

[106] Yoshio Okumura. *Developing a spectral and colorimetric database of artist paint materials*. PhD thesis, Citeseer, 2005.

[107] Miguel A Otaduy, Bernd Bickel, Derek Bradley, Rita Borgo, Min Chen, Markus Höferlin, Kuno Kurzhals, Phil Legg, Simon Walton, Daniel Weiskopf, et al. Data-driven simulation methods in computer graphics: cloth, tissue and faces. In *Eurographics 2013-Tutorials*. The Eurographics Association, 2013.

[108] F. Pitié, A.C. Kokaram, and R. Dahyot. Automated colour grading using colour distribution transfer. *Computer Vision and Image Understanding*, 107(1):123–137, 2007.

[109] Réjean Plamondon. A kinematic theory of rapid human movements. *Biological Cybernetics*, 72:295–320, 1995.

[110] Michael JD Powell. Radial basis functions for multivariable interpolation: a review. In *Algorithms for approximation*, pages 143–167. Clarendon Press, 1987.

[111] Thierry Pudet. Real time fitting of hand-sketched pressure brushstrokes. *Computer Graphics Forum*, 13(3):205–220, 1994.

[112] Eric Risser, Charles Han, Rozenn Dahyot, and Eitan Grinspun. Synthesizing structured image hybrids. In *Proc. of SIGGRAPH*, pages 85:1–85:6. ACM, 2010.

[113] Lincoln Ritter, Wilmot Li, Brian Curless, Maneesh Agrawala, and David Salesin. Painting with texture. In *Proceedings of the Eurographics conference on Rendering Techniques*, pages 371–376, 2006.

[114] A. R. Robertson. Historical development of CIE recommended color difference equations. *Color Research & Application*, 15(3):167–170, 1990.

[115] Dave Rudolf, David Mould, and Eric Neufeld. A bidirectional deposition model of wax crayons. *Computer Graphics Forum*, pages 27–39, 2005.

[116] Suguru Saito, Akane Kani, Youngha Chang, and Masayuki Nakajima. Curvature-based stroke rendering. *Visual Computing*, 24:1–11, November 2007.

[117] Suguru Saito and Masayuki Nakajima. 3D physics-based brush model for painting. In *SIGGRAPH conference abstracts and applications*, page 226, 1999.

[118] Christopher Sandoval and Arnold D. Kim. Deriving Kubelka-Munk theory from radiative transport. *Journal of the Optical Society of America A*, 31:628–636, 2014.

[119] J. L. Saunderson. Calculation of the color of pigmented plastics. *Journal of the Optical Society of America*, 32:727–736, 1942.

[120] Frank R Schmidt, E Toppe, and Daniel Cremers. Efficient planar graph cuts with applications in computer vision. In *CVPR*, pages 351–356. IEEE, 2009.

[121] Ryan Schmidt. Stroke parameterization. In *Computer Graphics Forum*, volume 32, pages 255–263. Wiley Online Library, 2013.

[122] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 489–498, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[123] Colas Schretter. A brush tool for interactive texture synthesis. *ICGST International Journal on Graphics, Vision and Image Processing*, 6(5):55–60, 2005.

[124] Tarja Shakespeare and John Shakespeare. A fluorescent extension to the KubelkaMunk model. *Color Research & Application*, 28(1):4–14, 2003.

[125] J. M. Shearer and M. A. Wolfe. Alglib, a simple symbol-manipulation package. *Commun. ACM*, 1985.

[126] N.C. Silver and W. P. Dunlap. Averaging correlation coefficients: Should fisher's z transformation be used? *Journal of App. Psych.*, 72(1):146–148, 1987.

[127] Mario Costa Sousa and John Buchanan. Computer-generated graphite pencil rendering of 3D polygonal models. *Computer Graphics Forum*, 18:195–208, 1999.

[128] Mario Costa Sousa and John Buchanan. Observational model of blenders and erasers in computer-generated pencil rendering. In *Proceedings of Graphics Interface*, pages 157–166, 1999.

[129] Thomas Strothotte and Stefan Schlechtweg. *Non-photorealistic computer graphics: modeling, rendering, and animation*. Elsevier, 2002.

[130] Qian Sun, Long Zhang, Minqi Zhang, Xiang Ying, Shi-Qing Xin, Jiazhi Xia, and Ying He. Texture brush: an interactive surface texturing interface. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 153–160. ACM, 2013.

[131] Saeko Takagi, Masayuki Nakajima, and Issei Fujishiro. Volumetric modeling of colored pencil drawing. In *Proceedings of Pacific Graphics*, pages 250–258, 1999.

[132] Yannick Thiel, Karan Singh, and Ravin Balakrishnan. Elasticurves: exploiting stroke dynamics and inertia for the real-time neatening of sketched 2d curves. In *User Interface Software and Technology*, pages 383–392, 2011.

[133] Matthew Thorne, David Burke, and Michiel van de Panne. Motion doodles: an interface for sketching character motion. In *ACM SIGGRAPH 2007 courses*, page 24. ACM, 2007.

[134] Emmanuel Turquin, Marie-Paule Cani, and John F Hughes. Sketching garments for virtual characters. In *ACM SIGGRAPH 2007 courses*, page 28. ACM, 2007.

[135] William Van Haevre, Tom Van Laerhoven, Fabian Di Fiore, and Frank Van Reeth. From dust till drawn: A real-time bidirectional pastel simulation. *The Visual Computer*, pages 925–934, 2007.

[136] Tom Van Laerhoven and Frank Van Reeth. Real-time simulation of watery paint. *Computer Animation and Virtual Worlds*, 16:429–439, 2005.

[137] Tom Van Laerhoven and Frank Van Reeth. Brush up your painting skills: Realistic brush design for interactive painting applications. *The Visual Computer*, 23(9):763–771, 2007.

[138] Peter Vandoren, Tom Van Laerhoven, Luc Claesen, Johannes Taelman, Chris Raymaekers, and Frank Van Reeth. Intupaint: Bridging the gap between physical and digital painting. In *Proceedings of Horizontal Interactive Human Computer Systems*, pages 65–72, 2008.

[139] Tamás Varga, Daniel Kilchhofer, and Horst Bunke. Template-based synthetic handwriting generation for the training of recognition systems. In *Conference of the International Graphonomics Society*, pages 206–211, 2005.

[140] William E. Vargas and Gunnar A. Niklasson. Applicability conditions of the Kubelka-Munk theory. *Applied Optics*, 36(22):5580–5586, 1997.

[141] Bin Wang, Wenping Wang, Huaiping Yang, and Jiaguang Sun. Efficient example-based painting and synthesis of 2d directional texture. *Visualization and Computer Graphics, IEEE Transactions on*, 10(3):266–277, 2004.

[142] Chung-Ming Wang and Ren-Jie Wang. Image-based color ink diffusion rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 13(2):235–246, 2007.

[143] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra, and Greg Turk. State of the art in example-based texture synthesis. In *Eurographics 2009, State of the Art Report*, 2009.

[144] Stephen Westland, Laura Iovine, and John M. Bishop. Kubelka-Munk or neural networks for computer colorant formulation? In *Proceedings of SPIE Vol. 4421, the 9th Congress of the International Colour Association*, pages 745–748, 2002.

[145] Michael T. Wong, Douglas E. Zongker, and David H. Salesin. Computer-generated floral ornament. In *SIGGRAPH*, SIGGRAPH '98, pages 423–434, New York, NY, USA, 1998. ACM.

[146] Grady Wright. *Radial basis function interpolation: numerical and analytical developments*. PhD thesis, University of Colorado, 2003.

[147] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

[148] Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. In *Proceedings of the International Conference on Machine Learning*, pages 153–160, 2012.

[149] Ning Xie, Hamid Laga, Suguru Saito, and Masayuki Nakajima. Contour-driven sumi-e rendering of real photos. *Computers & Graphics*, 35(1):122–134, 2011.

[150] Miao Xu and Jun Dong. Generating new styles of chinese stroke based on statistical model. In *Proc. International Multiconference on Computer Science and I.T.*, pages 215–222, 2006.

[151] Songhua Xu, Francis Lau, Feng Tang, and Yunhe Pan. Advanced design for a realistic virtual brush. *Computer Graphics Forum*, 22(3):533–542, 2003.

[152] Songhua Xu, Haisheng Tan, Xiantao Jiao, Francis C.M. Lau, and Yunhe Pan. A generic pigment model for digital painting. *Computer Graphics Forum*, 26(3), 2007.

[153] Songhua Xu, Min Tang, Francis Lau, and Yunhe Pan. A solid model based virtual hairy brush. *Computer Graphics Forum*, 21(3):299–308, 2002.

[154] Songhua Xu, Min Tang, Francis Lau, and Yunhe Pan. Virtual hairy brush for painterly rendering. *Graphical Models*, 66(5):263–302, 2004.

[155] Songhua Xu, Yingqing Xu, Sing Bing Kang, David H. Salesin, Yunhe Pan, and Heung-Yeung Shum. Animating chinese paintings through stroke-based decomposition. *ACM Transactions on Graphics*, 25(2):239–267, 2006.

[156] Chung-Ren Yan, Ming-Te Chi, Tong-Yee Lee, and Wen-Chieh Lin. Stylized rendering using samples of a painted image. *IEEE Trans. Visualization and Computer Graphics*, 14(2):468–480, 2008.

[157] Li Yang and Björn Kruse. Revised Kubelka-Munk theory. I. theory and application. *Journal of the Optical Society of America A*, 21(10):1933–1941, 2004.

[158] Li Yang, Björn Kruse, and Stanley J. Miklavcic. Revised Kubelka-Munk theory. II. unified framework for homogeneous and inhomogeneous optical media. *Journal of the Optical Society of America A*, 21(10):1942–1952, 2004.

[159] Kun Yu, Yunhong Wang, and Tieniu Tan. Writer identification using dynamic features. In David Zhang and Anil Jain, editors, *Biometric Authentication*, volume 3072 of *Lec. Notes in Comp. Sci.*, pages 1–8. Springer, 2004.

[160] Lap Fai Yu. *Data-Driven Optimization for Modeling in Computer Graphics and Vision*. PhD thesis, University of California, 2013.

[161] Robert C Zeleznik, Kenneth P Herndon, and John F Hughes. Sketch: An interface for sketching 3d scenes. *Computer Graphics*, 1996.

[162] Kun Zeng, Mingtian Zhao, Caiming Xiong, and Song-Chun Zhu. From image parsing to painterly rendering. *ACM Transactions on Graphics*, 29(1), 2009.

[163] Shizhe Zhou, Anass Lasram, and Sylvain Lefebvre. By-example synthesis of curvilinear structured patterns. *Computer Graphics Forum*, 32(2), 2013.