Interactive Image Editing Methods for Reproducing Real-world Appearance Variations

March ２０１７

Yuki Endo

Interactive Image Editing Methods for Reproducing Real-world Appearance Variations

Graduate School of Systems and Information Engineering

University of Tsukuba

March ２０１７

Yuki Endo

Abstract

Photorealistic reproduction of real-world phenomena such as weathering (appearance change of materials over time) and reflection on the surfaces of materials is crucial for image manipulation to represent scenes as realistic and natural ones. Thanks to the ease and ubiquity of digital cameras and broadband Internet, a large amount of digital image data can be easily obtained, and thus 2D image operations are important for not only professionals in several industries (e.g., advertising and computer game) but also amateurs that share image contents on the web. However, it is difficult for many users to create photorealistic effects on images by using general image editing tools. This is because these operations require *sophisticated expertise for individual phenomena* as well as *specialized skills of image processing*. Therefore, this thesis presents a framework containing solutions to these problems.

In particular, our framework contains four solutions: (1) physical simulation-based method that generates specific weathering, (2) example-based method that can handle various types of weathering, (3) matting and compositing method that synthesizes effects caused by optical phenomena by considering reflection and refraction of light, and (4) interactive image preprocessing method that deeply considers user intent for image editing.

First, we focus on water flow stains that are salient effects among weathering phenomena. Our method employs particle simulations in order to reproduce water flow stains on images. We expand an exiting simulation scheme, and our simulation scheme generates realistic results on images by considering surface roughness on a wall and perspective of a scene. The perspective is easily specified by the user using our interactive interface.

Next, we propose a method that can reproduce various types of weathering including rust and moss with bumpy surfaces that existing methods cannot handle. The example-based method exploits weathering examples in images to generate weathering effects. In our method, the pixels of a target object containing weathered regions is separated into two components, shading and reflectance. The reflectance component of weathering spreads as the weathering progresses. On the other hand, to handle bumpy surface of weathering, fine-scale details of shading are extracted as high-frequency components of the shading image, and they are added onto weathered object surfaces.

In addition, a matting method is proposed to decompose pixels on the surface of an object into reflection and transmission layers, and alpha matte (mixing ratio of reflection and transmission layers) on the basis of optical phenomena called Fresnel reflection. The decomposed layers are composited to synthesize reflections of newly inserted objects by using ray tracing. This is the first attempt to extract the complicated reflection for Fresnel reflection in a single image.

For the above methods, a user needs to preliminarily specify a regions of interest (ROIs) where several effects are synthesized. In general image editing, this operation is also essential to create desired images. However, carefully specifying ROIs is tedious. Therefore, we

propose a technique that can automatically extract ROIs from a few sparse user inputs on the basis of image features. This technique is generally called *edit propagation* but existing methods cannot sufficiently consider user intent for feature extraction, and thus extracting appropriate ROIs is difficult in some cases. In order to solve this problem and accurately extract ROIs, we propose a method that can automatically learn appropriate image features using deep neural networks (DNNs). We also present an efficient learning scheme of our DNN model. Our method can be applied to not only ROIs extraction but also several image editing tasks such as colorization.

We conduct extensive experiments including user studies and demonstrate that our methods enable users to more easily create realistic images with real-world phenomena than previous approaches. Finally, we conclude this thesis by providing future vision of photorealistic image editing techniques.

Acknowledgements

First and foremost, I would like to thank my supervisors, Prof. Jun Mitani and Prof. Yoshihiro Kanamori who give insightful comments and suggestions. This thesis has been done under the direction of them, and would not have been possible without their helpful advices and encouragement.

I am sincerely and extremely grateful to Prof. Yoshihiro Kanamori. His constant encouragement and very broad knowledge in computer graphics as well as his expertise in paper writing allowed me to grow as a research scientist. Furthermore, his enthusiasm and attitude toward research were valuable experiences for me, and his sense of humor made working at the lab a pleasant experience.

In addition, I would like to express my gratitude to so many my colleagues of Nippon Telegraph and Telephone (NTT) Corporation for allowing me to grow as a researcher and person. I would like to especially thank the members at NTT Service Evolution Laboratories, Dr. Hiroyuki Toda, Takafumi Inoue, Dr. Kyosuke Nishida, Dr. Makoto Nakatsuji, Dr. Yoshihiko Suhara, Yasuyuki Kataoka , Jun Ito, Mayumi Hadano, and Takuya Nishimura. I learned visions and philosophy of research from them.

I would also like to thank Emeritus Professor Seiichi Nishihara, Emeritus Professor Yukio Fukui, and so many members of non-numerical processing algorithms laboratory for giving me constructive comments and warm encouragement.

I am grateful to great faculty members, Prof. Kazuhiro Fukui, Prof. Hitoshi Kanoh, Prof. Suguru Saito, and Prof. Ko Sakai for reviewing this thesis and giving insightful comments.

Finally, I would like to thank my parents for always being supportive.

Contents

List of Figures

Chapter 1

Introduction

This chapter gives a brief introduction in the main goals of this thesis. It summarizes methods and results contributed by the thesis and concludes with a short overview over the organizational structure of this document.

1.1 Motivation

In the field of computer graphics (CG), many approaches for reproducing appearance of materials have been studied. For example, as shown in Figure 1.1, most materials in the real world change their appearance over time because they sometimes get rust and dirty. The appearance also varies depending on materials themselves, that is, glass transmits light, and water surfaces reflect scenery and objects around. Reproducing such real-world appearance variations by using CG techniques is not only crucial for enhancing realism in rendering 3D scenes but also important for image manipulation to represent scenes as realistic and natural ones. This thesis studies the image editing techniques for reproducing real-world appearance variations on 2D images.

An alternative route to reproducing real-world appearance variations on 2D images is to use commercial image editing software such as Adobe Photoshop. A user generates desired image contents using many types of tools such as a color brush and spray on specific regions. This approach enables users to edit images in detail by making full use of many tools and parameter settings. However, due to such wide degree of freedom for editing operations, the user needs *sophisticated expertise for appearance variations* as well as *specialized skills of image editing* to create photorealistic effects on 2D images. If we can devise reproduction techniques of real-world appearance variations on 2D images and can provide interactive user interface with a few user inputs, such complicated editing operations are simplified and the burdens imposed on the user are reduced.

To compensate for these requirements, the purposes of this thesis are (1) to model real-world appearance variations on 2D images. Additionally, in order to apply such models

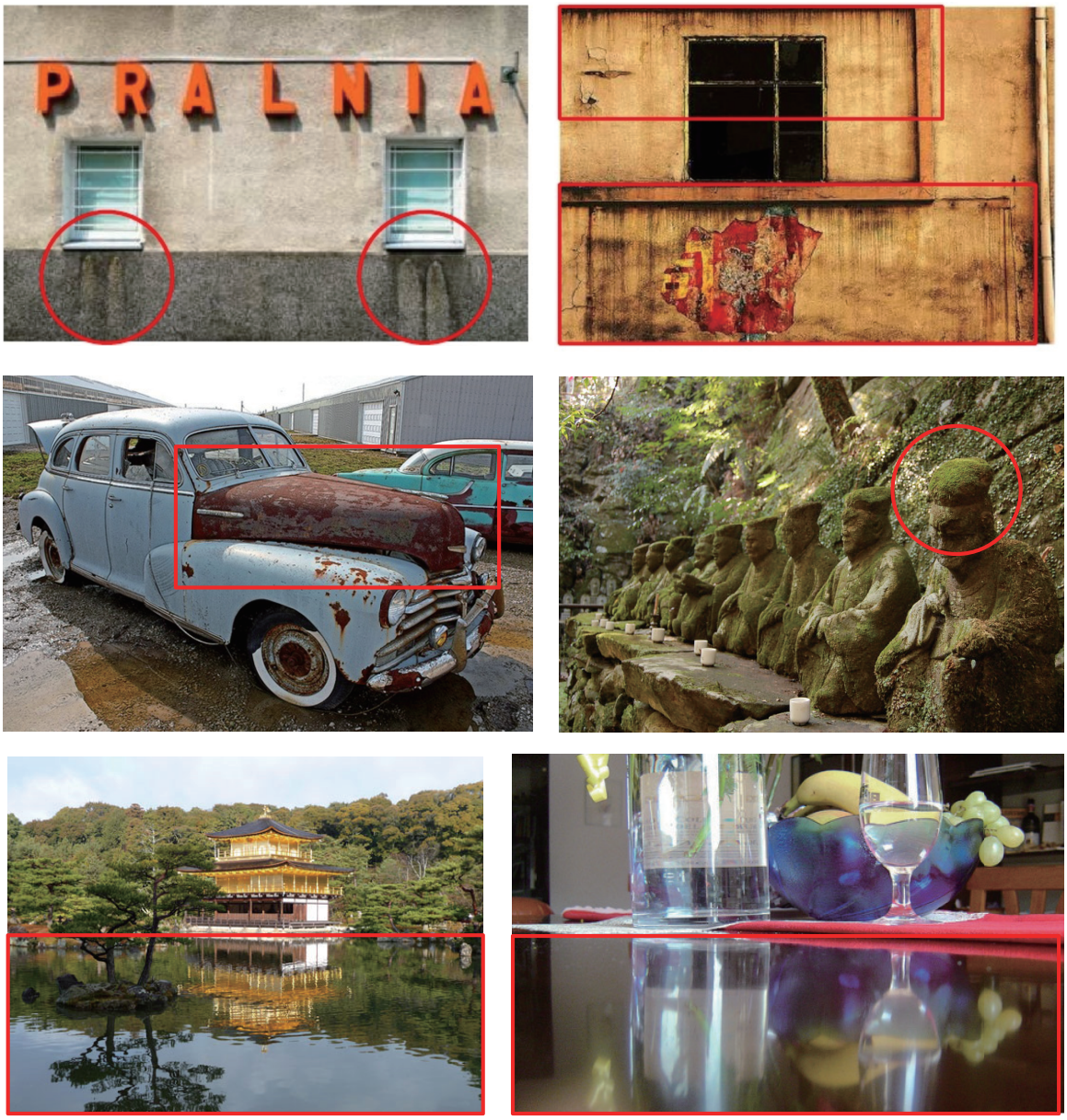


Figure 1.1: Examples of real-world appearance variations, water flow stains (top row), rust and moss (middle row), and reflection (bottom row).

to specific objects in images, (2) extracting regions of interest (ROIs) as preprocessing is an important editing operation. This procedure is also useful for other basic editing operations, such as colorization, smoothing, and tone adjustment. As for (1), however, most existing methods require 3D geometry of target objects, optical attributes of materials and light sources, and object behavior based on physical laws. Additionally, it is difficult for most users to handle generic 3DCG modeling software. For this reason, image-specific methods that can reproduce real-world phenomena are needed. As for (2), although a number of approaches have been proposed to efficiently extract ROIs with a few user inputs, they cannot appropriately consider user intent. Consequently, the extracted regions often protrude outside the ROIs or become smaller than the ROIs, and thus the user takes a lot of trial and error.

In summary, the goal of this thesis is to establish an efficient framework consisting of preprocessing and reproduction methods for real-world appearance variations on 2D images.

1.2 Principal Contributions

As this thesis mentioned before, existing general image editing software requires sophisticated expertise for individual appearance variations and specialized skills of image processing .

For the solution to the former problem, A number of approaches to modeling real-world appearance variations on 2D images have been proposed in computer graphics. As shown in Figure 1.2, this thesis focuses on two main categories in real-world appearance variations; time-varying and optical appearance variations. For example, the former category includes weathering effects and different times of day (morning, daytime, and night), and the latter category includes surface reflection, shadow, and haze. This thesis focuses on weathering effects and surface reflection because these phenomena are ubiquitous in our daily lives but methods for them have not been well established.

Furthermore, as we explained before, specifying ROIs is especially important for our applications. Given sparse user strokes or bounding boxes as input, most existing methods extract ROIs on the basis of visual and spatial features such as color, textures, and coordinates. In the existing methods, the accuracy of extracted ROIs heavily depends pre-selected features and their weights. However, these conditions have been determined by the system designers on the basis of their heuristics. That is, we need to preliminarily define the similarity measures between image features but this procedure is difficult, and the existing methods takes a lot of user inputs and trial and error.

In particular, we present the following solutions to the above challenges:

1. Physical simulation-based method that generates specific weathering

2. Example-based method that can handle various types of weathering

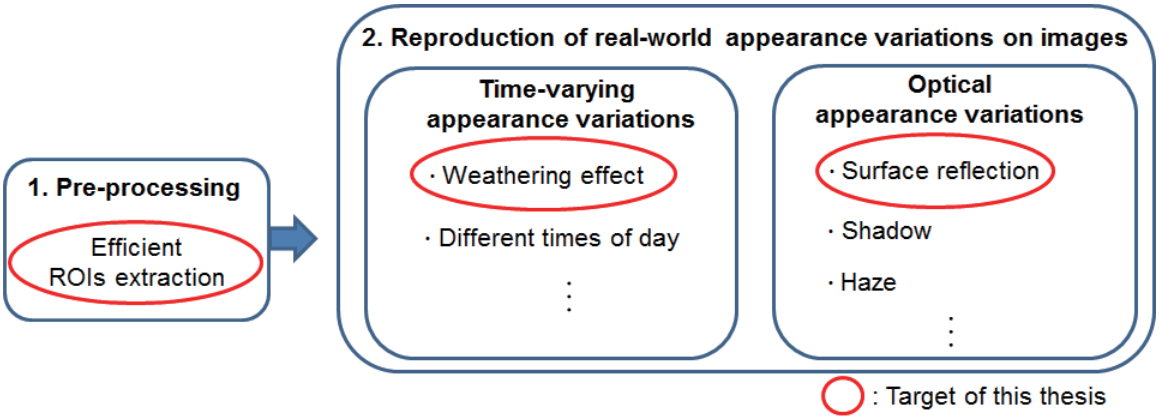


Figure 1.2: The framework and categorization of reproducing real-world appearance variations and research focuses in this thesis.

3. Matting and compositing method that synthesizes effects caused by optical appearance variations (reflection of light)

4. Interactive image preprocessing method that appropriately considers user intent using deep neural networks for efficient image editing such as ROIs extraction, (re)colorization, etc.

The detailed contributions to each method are described as follows:
Reproduction of weathering effect based on physical simulation: We focus on a salient aging effect, *stains by water flows*, and present a system that allows the user to add such stains directly and easily onto building walls in outdoor images. Our system represents a water droplet with a particle and simulates the dissolution, transport and sedimentation of deposits using particles in the regions specified by the user. While simulation scheme is based on a simple model proposed for 3D models by Dorsey et al. [27], we modify it to improve the performance and the usability for image editing. In the simulation, realistic and complex stains can be obtained by accounting for the surface roughness where water flows. The user can specify the initial and terminal positions of particles by drawing a few lines. Furthermore, the user can adjust the amount of deposits according to the perspective in the image; using a control mesh drawn by the user, our system makes water flow stains shorter and thinner as the distance from the viewpoint gets longer. The quick feedback of the simulation enables interactive manipulation.
Reproduction of weathering effect based on examples: In addition to the simulation-based approach, we propose example-based approach that uses weathering exemplars on images. Although the example-based approach cannot reproduce weathering effects based on physical laws in the same way as the simulation-based approach, it can handle many kinds of weathering if exemplars are available on input images. However, existing example-

based methods cannot handle weathering effects with geometric details such as moss and rust. To solve this problem, we propose a technique for modeling weathering effects with time-varying geometric details in images. Specifically, we focus on spatio-temporal variations of shading as well as reflectance due to weathering. We extract fine-scale details of shading as high-frequency components of the shading image, and add them onto object surfaces as weathering progresses. This extraction can be done with only a few additional user inputs. De/weathering with time-varying shading is accomplished in our work for the first time, and substantially enhances the reality of de/weathering effects. Moreover, we introduce a brush-like user interface for editing weathering effects locally and interactively. This interface allows the user to edit arbitrary weathering distributions, for example, similar to those observed in the real world.

Reproduction of surface reflection: We propose a matting method that handles surface reflection in a single image based on user markups. Targeting reflection at surfaces such as the surface of deep water, glossy table top or floor, we introduce the following three assumptions; 1) the transmission color can be approximated as uniform, 2) a pair of an object that is the source of reflection and the corresponding reflection image can be found in the input image, and 3) the reflection surface is mostly planar but possibly wavy. We first estimate the transmission color based on the first and second assumptions, using *color transfer*. We then roughly estimate the reflection component and alpha matte as well as camera parameters, assuming the alpha matte is smooth. However, the alpha matte should contain high-frequency regions in case of wavy reflection surfaces. We thus propose a novel filter to refine the alpha matte, which is validated using ground-truth data. Using the calculated information, we also provide a compositing system with which the user can composite new objects onto the input image with plausible reflection in real time.

ROIs extraction for efficient image editing: To extract ROIs with a few user inputs, edit propagation is a well-known approach. In this approach, a user roughly specifies sparse strokes in target regions, and then ROIs are automatically extracted on the basis of the specified user strokes and image features. By propagating image edits on user strokes to the extracted ROIs, edit propagation can be used for various applications such as colorization and tone adjustment. However, previous work must heuristically select the image features and adjust parameters for the features in accordance with a user's needs and target images. To address this issue, we employ deep neural networks (DNNs). Our method uses low-level visual patches and spatial pixel coordinates as input of a DNN that automatically extracts features adapted to user-specified strokes from a single image. In contrast to most previous work, we do not need to adjust the importance of the input features. Then, we use the DNN as a classifier that estimates user stroke probabilities, which represent how likely it is that each pixel belongs to each stroke, from extracted features on the entire image.

1.3 Outline

This thesis consists of six chapters on the following pages.

This thesis begins by reviewing related work in Chapter 2. We overview existing studies for reproducing real-world appearance variations such as weathering and surface reflection. Specifically, we introduce methods for several types of weathering and optical appearance variations for 3DCG as well as 2DCG and explain challenges existing in these methods. Besides, we describe related studies to edit propagation that can efficiently extract image masks.

In Chapter 3, this thesis proposes an interactive design system for water flow stains on outdoor images. In our system, a user specifies some control lines and tunes simulation parameters using our user interface explained in this chapter. The thesis also proposes a particle simulation model that can reproduce water flow stains on 2D images. We conduct a user study to verify the effectiveness of our method by comparing our system with an existing image editing software

Different from the above simulation-based approach, Chapter 4 focuses on example-based approach that can handle various types of weathering with geometric details. We first introduce an existing example-based method and then describe our model for weathering with geometric details. Furthermore, we explain a weathering transfer method and a brush interface for interactive editing. Comparing our method with the existing method, we evaluate the effectiveness of our method.

In Chapter 5, this thesis proposes a matting and compositing framework for Fresnel reflection on wavy surfaces. We first describe our matting procedure that decomposes target surfaces on an input image into a reflection layer, transmission layer, and mixing ratio of them (alpha matte). The thesis also describes a method that composites new reflection on the basis of decomposed layers. We discuss qualitative results generated by our method and a related method.

In Chapter 6, this thesis presents a variant of edit propagation, which can efficiently extract ROIs from sparse user stroke input. To automatically extract appropriate images features and determine the weights of them, a novel DNN architecture is proposed. The DNN model learns image features from user strokes and propagates image edits to regions where the user strokes are not specified. In our experiments, we apply our method to several image editing such as segmentation for mask extraction and (re-)colorization. We evaluate the result of these application with our DNN model by using existing image segmentation dataset and conducting a user study for colorization.

Chapter 7 summarizes the conclusions of this thesis and provides future research directions.

1.4 Publications and Awards

1.4.1 Reference papers

This thesis is based on the following publications:

Journal papers (with peer review)

1. Y. Endo, S. Iizuka, Y. Kanamori, and J. Mitani: "DeepProp: Extracting Deep Features from a Single Image for Edit Propagation," *Computer Graphics Forum, Vol.35, No.2*, pp.189-201, May 2016.

2. Y. Endo, Y. Kanamori, J. Mitani, Y. Fukui: "A Design System for Water Flow Stain Images Using Particle Simulation ," *IPSJ Journal, Vol.56, No.3*, pp.1049-1058, 2015 (in Japanese with English Abstract).

3. Y. Endo, Y. Kanamori, Y. Fukui, and J. Mitani: "Matting and Compositing for Fresnel Reflection on Wavy Surfaces," *Computer Graphics Forum, Vol.31, No.4*, pp.1435-1443, June 2012.

International conference papers (with peer review)

1. Y. Endo, Y. Kanamori, J. Mitani, and Y. Fukui: "Weathering Effects with Geometric Details for Images," *In Proc. of Computer Graphics International 2011*, s-24 pp.1-4, Otawa, June 2011.

2. Y. Endo, Y. Kanamori, J. Mitani, and Y. Fukui: "An Interactive Design System for Water Flow Stains on Outdoor Images," *In Proc. of Smart Graphics 2010*, pp.160-171, Banff, June 2010.

Domestic conference papers (with peer review)

1. Y. Endo, Y. Kanamori, J. Mitani, and Y. Fukui: "Water Flow Stain Generation System in Images Using Particle Simulation," *VC/GCAD Symposium 2010*, 2010 (in Japanese).

2. Y. Endo, Y. Kanamori, J. Mitani, and Y. Fukui: "Water Flow Stain Generation System in Images Using Particle Simulation," *NICOGRAPH 2010*, 2010 (in Japanese).

Domestic conference papers (without peer review)

1. Y. Endo, Y. Kanamori, J. Mitani, and Y. Fukui: "Reflection Matting and Compositing in Images," *Proceedings of the 146th GCAD*, 2012 (in Japanese).

2. Y. Endo, Y. Kanamori, J. Mitani, and Y. Fukui: "Interactive Weathering Editing System for Scene Images Using Appearance Maps," *Proceedings of the 139th GCAD*, 2010 (in Japanese).

1.4.2 Other papers

The additional following papers were also published but not directly related to this thesis:

Journal papers (with peer review)

1. Y. Endo, H. Toda, K. Nishida, J. Ikedo: "Classifying spatial trajectories using representation learning," *International Journal of Data Science and Analytics*, pp 1-11, 2016.

2. S. Iizuka, Y. Endo, Y. Kanamor, and J. Mitani: "Single Image Weathering via Exemplar Propagation," *Computer Graphics Forum, Vol.35, No.2*, pp.501-509, May 2016.

3. M. Kawano, Y. Endo, H. Toda, Y. Koike, K. Ueda: "Feature Extraction from Movement Trajectory Based on Recursive Autoencoder," *DBSJ Journal, Vol.14, No.12*, pp.1-6, 2016 (in Japanese with English Abstract).

4. Y. Endo, H. Toda, Y. Koike: "Feature Extraction from GPS Logs Using Representation Learning for Transportation Mode Estimation," *IPSJ Transaction on Databases, Vol.8, No.3*, pp.12-23, 2015 (in Japanese with English Abstract).

5. S. Iizuka, Y. Endo, Y. Kanamori, J. Mitani, and Y. Fukui: "Object Repositioning Based on the Perspective in a Single Image," *Computer Graphics Forum, Vol.33, No.8*, pp.157-166, Dec. 2014.

6. S. Iizuka, Y. Endo, Y. Kanamori, J. Mitani, and Y. Fukui: "Efficient Depth Propagation for Constructing a Layered Depth Image from a Single Image," *Computer Graphics Forum, Vol.33, No.7*, pp.279-288, Oct. 2014.

7. S. Iizuka, Y. Endo, J. Mitani, Y. Kanamori, and Y. Fukui: "An Interactive Design System for Pop-Up Cards with a Physical Simulation," *The Visual Computer, Vol.27, No.6*, pp.605-612, June 2011.

International conference papers (with peer review)

1. Y. Endo, H. Toda, K. Nishida, and A. Kawanobe: "Deep Feature Extraction from Trajectories for Transportation Mode Estimation," *In Proc. of Pacific Asia Conference on Knowledge Discovery and Data Mining (PAKDD) 2016*, pp.54-66, Auckland, Apr. 2016.

2. Y. Endo, H. Toda, and Y. Koike: ”What's Hot in The Theme: “Query Dependent Emerging Topic Extraction from Social Streams,” *In Proc. of World Wide Web 2015 Companion*, pp.31-32, Florence, May 2015.

Domestic conference papers (with peer review)

1. S. Iizuka, Y. Endo, Y. Kanamori, J. Mitani: “Reproduction of Weathering with Texture Variations on Object Surfaces in Images,” *VC/GCAD Symposium2016*, pp.1524 - 1536, 2016 (in Japanese).

2. Y. Endo, H. Toda, K. Nishida, and Y. Koike: “Feature Extraction from GPS Trajectories Using Representation Learning for Transportation Mode Estimation,” *WebDB Forum 2014*, 2016 (in Japanese).

3. S. Iizuka, Y. Endo, J. Mitani, Y. Kanamori, and Y. Fukui: “Interactive Layered 3D Model Generation from Single Image Using Scribbles,” *VC/GCAD Symposium 2013*, 2013 (in Japanese).

4. S. Iizuka, Y. Endo, M. Hirose, J. Mitani, Y. Kanamori, and Y. Fukui: “Interactive Object Repositioning on Scene Images Considering Scene Depth,” *VC/GCAD Symposium 2012*, 2012 (in Japanese).

5. S. Iizuka, Y. Endo, J. Mitani, Y. Kanamori, and Y. Fukui: “Pop-up Card Design Support System Using Physical Simulation,” *VC/GCAD Symposium 2011*, 2011 (in Japanese).

Domestic conference papers (without peer review)

1. Y. Endo, K. Nishida, H. Toda, and H. Sawada: “Destination Prediction Considering Long-term Dependency,” *Multimedia, Distributed, Cooperative, and Mobile Symposium (DICOMO2016)*, pp.1524 - 1536, 2016 (in Japanese).

2. S. Takimoto, K. Nishida, Y. Endo, H. Toda, H., Sawada, Y. Ishikawa: “Personalized Destination Prediction Considering Time of Day,” *Forum on Data Engineering and Information Management(DEIM2016)*, 2016 (in Japanese).

3. M. Kawano, Y. Endo, H. Toda, Y. Koike, K. Ueda: “Automatic Feature Extraction from Movement Trajectories Using Recursive Autoencoder,” *Forum on Data Engineering and Information Management (DEIM2015)*, 2015 (in Japanese).

4. Y. Endo, H. Toda, S. Suzaki: “Query-dependent Local Emerging Topic Extraction on Time Series Texts,” *Forum on Data Engineering and Information Management (DEIM2014)*, 2014 (in Japanese).

Technology reports of University of Tsukuba (without peer review)

1. S. Iizuka, Y. Endo, M. Hirose: "Interactive Scene Image Editing System Considering 3D structures," *Advanced Research and Development Solution Projects Report 2011*, 2012 (in Japanese).

2. S. Iizuka, Y. Endo: "Pop-up Card Design Support System Using Mass-Spring Model," *Advanced Research and Development Solution Projects Report 2010*, 2011 (in Japanese).

1.4.3 Grants and awards

First Author

1. July 2016: Paper award, in the 2016 Multimedia, Distributed, Cooperative, and Mobile Symposium.

2. May 2016: Grants for Researchers Attending International Conferences from NEC C&C Foundation.

3. March 2016: IPSJ Yamashita SIG Research Award from Information Processing Society of Japan.

4. December 2015: Research and Development Encouragement Award from NTT Service Innovation Laboratory Group.

5. November 2014: Paper award, in WebDB Forum 2014.

6. March 2013: IPSJ Yamashita SIG Research Award from Information Processing Society of Japan.

7. February 2012: GCAD award, in 146th GCAD.

8. July 2010: GCAD award, in 139th GCAD.

9. June 2010: GCAD award, in VC/GCAD Symposium 2010.

Coauthor

1. May 2016: Paper Award, in the 8th Forum on Data Engineering and Information Management.

2. October 2015: Highest Award, pedestrian movement support ideathon and hackathon hosted by Director-General for Policy Planning, Ministry of Land, Infrastructure, Transport and Tourism.

3. March 2015: Student presentation award, in the 7th Forum on Data Engineering and Information Management.
4. June 2013: GCAD award, in VC/GCAD Symposium 2013.
5. June 2011: GCAD award, in VC/GCAD Symposium 2011.

Chapter 2

Related Work

In this chapter, we explain existing studies for reproducing real-world appearance variations using CG techniques. We also explain related work of edit propagation for mask extraction and briefly introduce deep learning related to our method.

2.1 Reproduction of Real-world Appearance Variations

2.1.1 Time-varying appearance variations

Realistic representation of weathering and aging phenomena has been an important theme in the field of computer graphics. The previous methods can be broadly grouped into *example-based approach* that obtains information from real photographs and *simulation-based approach* that handles specific targets such as rusts, cracks and dirt. Here we briefly introduce some of them, and refer to the survey paper by Merillou et al. [59] for more information.

Simulation-based approach: Physically-based simulations of weathering and aging have been applied to various targets such as stones [25], paint peeling [63], cracks of clay-like materials [37] and wooden materials [100]. These methods are accompanied by geometric changes. Previous methods that do not handle geometric changes include weathering and aging by dust [38], rust [26] and moss [24].

Dorsey et al. [27] employed a particle simulation to reproduce stains caused by flows streaming on 3D models. In their method, particles dissolve and carry deposits, and the deposits are accumulated on the surface according to a set of partial differential equations. Their method can represent various patterns of realistic stains by tuning parameters, but lacks flexible control for designing stains. On the other hand, our simulation scheme is a modified version of Dorsey et al.'s method specialized for 2D image editing, and collaborates with the user interface for designing water flow stains.

Example-based approach: There are many techniques to change the appearance of ob-

jects in images, such as retexturing objects with arbitrary textures [29, 101], and creating translucent or specular objects rendered with arbitrary bidirectional distribution functions (BRDFs) [40]. However, these methods do not take into account the time-varying appearance due to weathering.

Gu et al. [35] obtained and modeled time-space varying bidirectional distribution functions (BRDFs), and then reproduced temporal variations of materials. Wang et al. [84] constructed a manifold that represents the temporal variation of a material using BRDFs measured at various points on the material in order to reproduce the transition of complicated texture patterns. However, these techniques cannot be applied to a single image because of the requirement of captures of full time sequences.

Xue et al. [95] applied Wang et al.'s method to objects in 2D images to synthesize the weathered or de-weathered appearance. Although example-based approaches successfully reproduce various texture patterns, they suffer from handling physical properties of the target objects because they do not account for the physical law of the phenomena. Their method can not only achieve realistic results but also be applied to images including complicated shading variations. Later Bandeira and Walter [5] used an *appearance map*, a simplified version of an appearance manifold, for real-time editing. Despite faster computation time, the method of Bandeira and Walter can achieve sufficiently realistic weathering effects. However, these methods cannot handle time-varying geometric details caused by weathering. Recently, Bandeira and Walter [6] tackled this problem by using normal mapping. Their approach seems to succesufully reproduce the examples of cracks. However, the propagation of cracks is separately handled without using weathering degrees, and thus their method cannot handle the cases where both shading and reflectance change simultatenously according to de/weathering.

2.1.2 Optical appearance variations

There have been many image matting methods for optical appearance variations. Wu et al. [88] extracted shadows from natural images, assuming the colors in a shadowed region are the products of the shadow color and the ground colors, based on Retinex. Similarly, haze removal techniques assume that an input image is a linear blending of the haze color and background colors. Tan's method [80] enhances the image contrast, and does not decompose the image. Fattal [31] estimates a uniform haze color as well in dehazing by assuming that the transmission and surface shading are locally uncorrelated. In our case this statistical approach will suffer from reflection regions where the transmission color is dominant or a sky without shading is largely reflected. Methods using the dark channel prior [36, 42] are based on the assumption that in natural images at least one color channel tends to be almost zero, which is not valid in regions where a blight sky is reflected in our case. Most importantly, applying these methods to our case is not trivial due to the difference of targets.

Methods for *natural image matting*, as introduced in a survey [83] and the evaluation

website of [71], extract foreground objects and an alpha matte from a single image, assuming that the user specifies definitely-foreground and definitely-background regions using a *trimap* or scribbles. Unfortunately, such definite separation is not available in reflection images, and thus we require alternatives.

While a basic way of reproducing reflection/refraction is the use of ray tracing with 3D geometries, several methods do so without geometric information. Environment matting [102, 19, 86, 65] measures the light transport for an reflective/refractive object in the real world using specialized devices or multiple photographs. Khan et al. [40] proposed a method that can change the material of an object in a photograph as if it were made of metal or glass, assuming that the input object is opaque and the depth can be estimated from the luminance intensity. However, our target object is not opaque and its luminance intensity does not provide depth information.

Yeung et al. proposed a matting method for separating glossy reflection and refraction at a glass surface in an image [99]. In their method, the reflection component is limited to white highlights that are extracted as definitely foreground in a *trimap*, and an alpha matte as well as a warping function of refraction are calculated based on user-specified strokes. We consider their method is the most relevant to our matting method. However, in our case, the reflection component consists of a variety of colors and definitely-foreground regions are not available. Additionally, it is impractical to manually specify small-scale warping caused by a wavy reflection surface.

Separating reflection and refraction on a glass has been studied in computer vision. Whilst some methods handle a single image [50, 49] and others multiple images [13, 33] as input, here we introduce the former. The key idea here is that luminance edges belong to either reflection or refraction components. However, an automated approach [50] cannot achieve sufficient results in many cases and manually specifying edges [49] is labor intensive. In our matting method, we do not handle complicated refraction but target a uniform transmission color as observed in deep sea water or a non-textured table top.

2.1.3 Material editing based on human perception

Different from the modeling methods based on physical phenomenon, other methods that simply adjust image statistics by considering human perception have been proposed. Motoyoshi et al. [60] found that brightness and specularity of object surfaces that human perceives depend on skewness of the histogram of the image. Sawayama et al. [73] proposed a wet filter. The wet filter transforms objects to wet ones by changing the luminance histogram positively skewed and enhancing the color saturation. Boyadzhiev et al. [12] proposed a method that can generate and remove several effects on human skin in images, such as blemishes and oily skin. This method decompose input image into high-frequency and low-frequency components and edits each component for any purpose. Although not all appearance variations still can be handled by the methods that exploit human perception, they can easily generate realistic materials through simple image processing operations.

Boyadzhiev et al.'s work is closely related to our sample-based weathering method proposed in this thesis in that the both methods edit images on the basis of high-frequency and low-frequency components.

2.2 Edit Propagation

The first studies related to edit propagation are colorizations using optimization [47] and chrominance blending [98]. In addition to these approaches based on pixel colors, colorizations using texture features in manga [68] and natural images [58] have been also proposed. Furthermore, we can regard tone adjustment [55], material editing [66], intrinsic images [10] and editing of bidirectional texture functions (BTFs) [92] as applications of edit propagation because they all propagate user edits based on similarities between on-stroke pixels and the rest of the image. Such stroke-based approaches are also used in matting [48, 72], edge-preserving smoothing [93, 16], and temporally consistent propagation for video editing [14].

Other than the various applications, advances have been also made in the fundamental algorithms for edit propagation. Li et al. proposed an approach using a gentle boost classifier by formulating edit propagation as a pixel classification problem [52]. *AppProp* [3] yields better propagation by optimizing color differences for optimizing color differences not only between nearby pixels, but also between non-neighboring ones. Computational efficiency has been also improved by using a kd-tree [91], continuously approximating feature space using *radial basis functions* (RBFs) [53], manifold learning [61], efficient stroke sampling [9], and sparse pixel sampling [97]. Most of the previous approaches share a common issue, namely that halo artifacts occur across object boundaries [45]. Chen et al. [45] achieved graceful color blending along object boundaries based on *locally linear embedding* (LLE), which was further improved in terms of efficiency using dictionary learning [14]. Xu et al. proposed a stochastic modeling of the similarities between on-stroke and off-stroke pixels, based on iterative feature discrimination and sparse sampling of on-stroke pixels [94]. Also, there are approaches using specific distance metrics such as geodesic distance [20] and diffusion distance [30]. Whereas these previous approaches use various image features, effective features vary depending on input images and user-specified strokes. Therefore, if multiple image features are considered simultaneously, we have to tune the importance that enhances or suppresses each feature for each image. Even worse, the more image features are used, the more over-fitting occurs.

2.3 Deep Learning

Recently, *deep learning* has gained significant attention in computer science. Deep learning is a generic term for techniques using deep neural network (DNN), and has achieved

outstanding results in various fields. An advantage of DNN is that it can be used for representation learning, where low-level features (e.g., pixel values) are directly used as input and automatically converted to higher-level features through intermediate layers, without conventional manual design of discriminative features. A commonly occurring problem is that of vanishing gradients of the energy functions, which hinders deep layers from sufficiently learning effective features, especially in deeper neural networks. Recent progress including the unsupervised pretraining with deep belief network (DBN) [8] and the use of rectified linear unit (ReLU) [34] as well as performance improvement of computers in the last few years have enabled efficient learning using DNN.

With regards to the types of neural networks used for deep learning, fully-connected neural network, convolutional neural network (CNN) and recurrent neural network (RNN) are commonly chosen [7]. Fully-connected neural networks such as multilayer perceptron (MLP) are typical examples in DNN and have been applied to classification and regression problems in several research fields (e.g., speech recognition [43, 23], activity recognition [44], and exemplar-based photo adjustment [96]). On the other hand, CNNs are often used if the inputs are images. Unlike general fully-connected networks, a CNN is a network consisting of convolutional layers with multiple filters and pooling layers, which enables CNN to achieve invariant features (e.g., shift invariance) under various deformations during feature extraction. CNNs have been applied to many image processing tasks, such as image classification [90], contour detection [76], motion deblurring [79], saliency detection [51], and depth estimation [56]. We adopt a combination of a CNN and a fully-connected network for edit propagation for the first time, and demonstrate its effectiveness through several applications.

Chapter 3

Reproduction of Water Flow Stains

Photorealistic reproduction of weathering and aging phenomena often plays an important role in the field of film production and scene prediction. Especially, *weathering and aging of buildings*, e.g., rusts, stains and pollution caused by rainfalls, chemical reaction, ultraviolet light and other various factors, have high demand because buildings are ubiquitous in our daily lives and thus might look unnatural without such phenomena. A usual way to seek such realism is to use photographed materials as texture images and edit them to obtain desired results. However, looking for appropriate materials itself is not easy, and editing materials requires labor-intensive work as well as professional skills.

In this chapter, we focus on a salient aging effect, *stains by water flows*, and present a system that allows the user to add such stains directly and easily onto building walls in outdoor images. Our system represents a water droplet with a particle and simulates the dissolution, transport and sedimentation of deposits using particles in the regions specified by the user. This simulation scheme is based on a simple model proposed for 3D models by Dorsey et al. [27], but we modify it to improve the performance and the usability for image editing. In the simulation, realistic and complex stains can be obtained by accounting for the surface roughness where water flows. The user can specify the initial and terminal positions of particles by drawing a few lines. Furthermore, the user can adjust the amount of deposits according to the perspective in the image; using a pair of auxiliary lines drawn by the user, our system makes water flow stains shorter and thinner as the distance from the viewpoint gets longer. The quick feedback of the simulation enables interactive manipulation. We demonstrate the effectiveness of our system through various results and user evaluations.

3.1 User Interface

This section describes our prototype system for designing images containing water flow stains. Our system adds such stains using particle simulations, and allows the user to easily

specify the regions where the water particles flow on the input image.

Figure 3.1 shows the screenshot of our prototype system. In the left window that displays the input image and flowing particles (blue), the user can directly specify the regions for simulation and the perspective of the input image, and then obtain simulated results quickly. In the right window, the user can configure the parameters and other settings of the simulation. In the middle panel, the user can select the color of deposits using either of the color pallet, color chooser or color picker. The current color is displayed at the bottom.

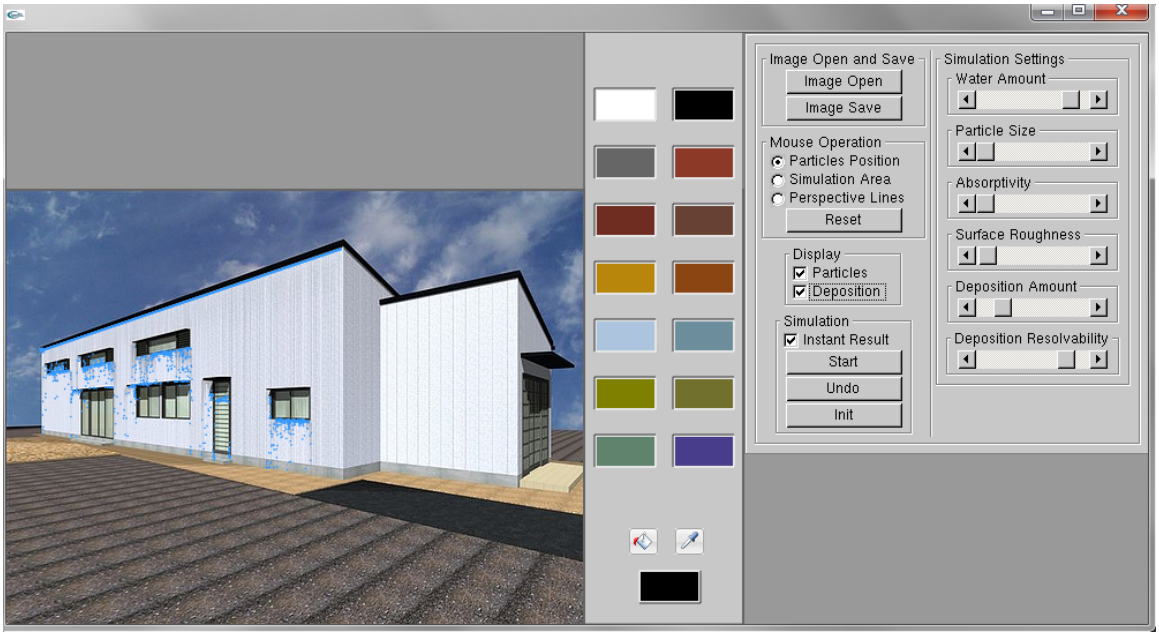


Figure 3.1: Screenshot of our system.[1]

3.1.1 Control lines

In the left window in Figure 3.1, the user specifies two types of control lines and a control mesh to control the particle simulation (Figure 3.2);

Source line

(the blue line in Figure 3.2(a)): to specify the initial position of particles. Particles are emitted from these lines.

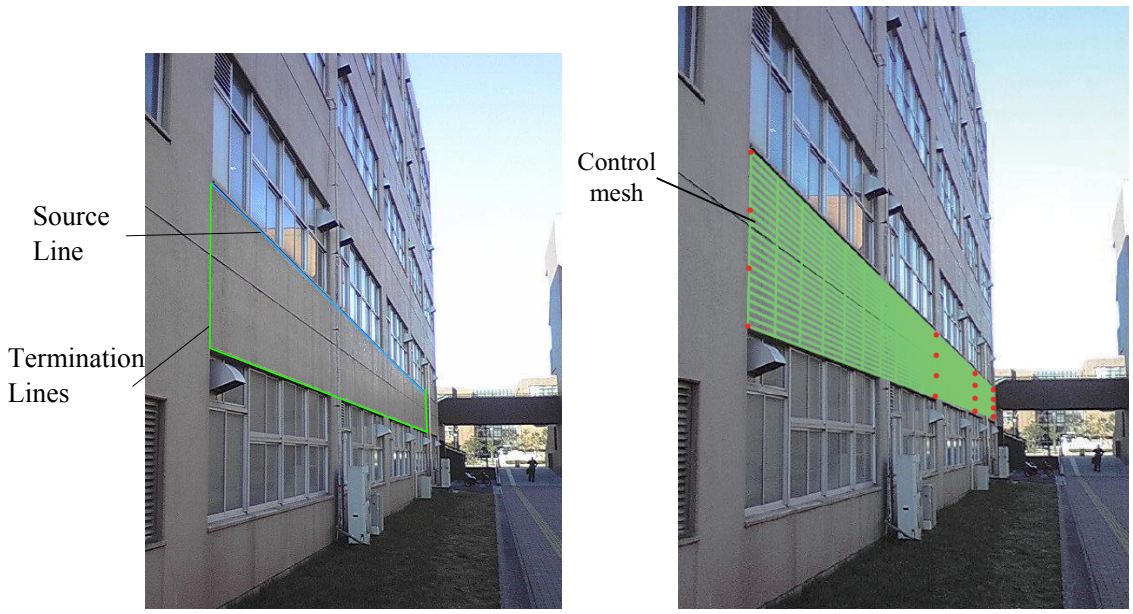
[1] Author of the input image: heavymoonj
URL: http://www.flickr.com/photos/heavymoonj/261996484/

Terminal lines

(the green lines in Figure 3.2(a)): to specify the positions where the simulation terminates. Particles are removed when they touch these lines.

Control mesh

(the green mesh and red control points in Figure 3.2(b)): to specify the perspective of the input image. The amount of deposits is adjusted according to the perspective of the input image, specified by these lines. See Section 3.2.2 for more details.



(a) (b)

Figure 3.2: Two types of control lines and a control mesh for the particle simulation. (a) The blue line specifies the source position of particles, and the green lines represent the terminal positions; particles are removed when they touch these lines. (b) The perspective of the input image is specified by the control mesh in order to adjust the amount of deposits according to the perspective.

3.1.2 Simulation parameters

The user can adjust the following six parameters to make a variety of water flow stains using scroll bars. The way each parameter influences the simulation is described in Section

3.2.2.

Water Amount.

The parameter for the water amount specifies the number of particles. By increasing this parameter, more complex stains caused by more flows can be obtained (Figure 3.3).

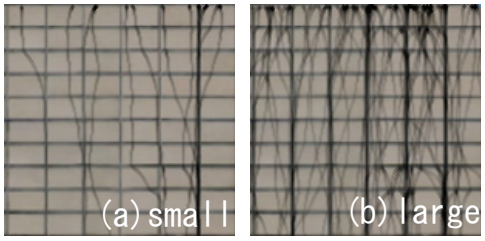


Figure 3.3: Effect of Water Amount.

Particle Size.

The relative size of particles with regard to the input image can be adjusted using this parameter (Figure 3.4).

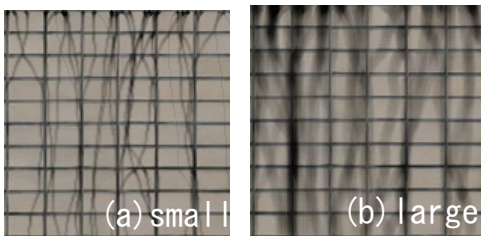


Figure 3.4: Effect of Particle Size.

Absorptivity.

This parameter specifies the rate with which the mass of each particle decreases. The flow distance of each particle becomes short by increasing this parameter (Figure 3.5).

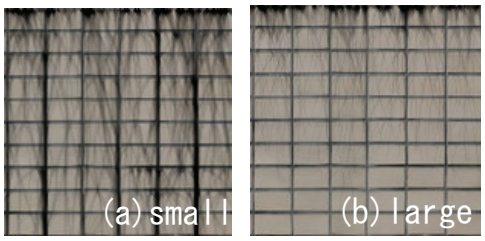


Figure 3.5: Effect of Absorptivity.

Surface Roughness.

This parameter determines how much each particle is diffused and decelerated due to the bumps on the surface. By increasing this parameter, more deposits tend to accumulate at cracks and ditches (Figure 3.6).

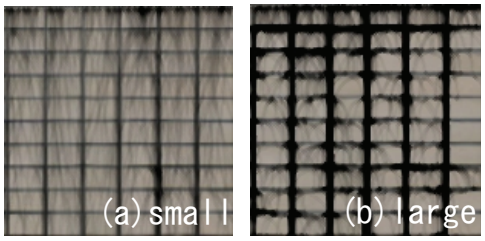


Figure 3.6: Effect of Surface Roughness.

Deposition Amount.

This parameter specifies the amount of deposits attached on the surface. A large value yields thick stains (Figure 3.7).

Deposition Resolvability.

This parameter determines how easily deposits dissolve in water particles. A large value yields a blurred image (Figure 3.8).

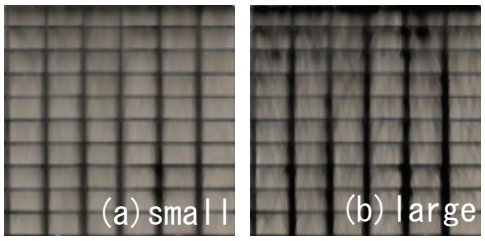


Figure 3.7:Effect of Deposition Amount.

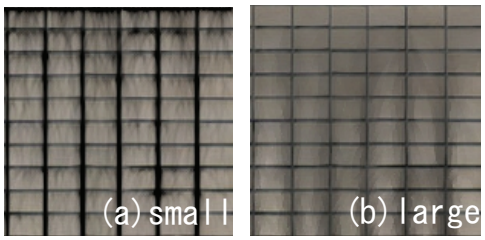


Figure 3.8:Effect of Deposition Resolvability.

3.2 Simulation Scheme

This section describes the simulation scheme to control water particles. Our scheme is based on Dorsey et al.'s model [27] because of its simplicity. In their model, each particle represents a water droplet that dissolves and transports deposits which are then accumulated along the tracks of particles. While their model targets the 3D space, we simulates on the 2D domain, i.e., the input image. We simplify their model to reduce the number of parameters and to provide faster feedback to the user. Furthermore, we extend it to handle the surface roughness on which particles flow and to adjust the amount of deposits according to the perspective of the image. In the following subsections, we first describe Dorsey et al.'s model briefly, and then elaborate on our model.

3.2.1 Dorsey et al.'s model

In Dorsey et al.'s model, a particle represents a water droplet parameterized by the mass m, position $\mathbf{x}$, velocity $\mathbf{v}$, and the amount of dissolved deposits S. Particles are initially

assigned on the surface of a 3D model at random by a rainfall, and then flow downward under the influence of the gravity and frictions. Particles interact with each other due to the repulsive forces among them. Water is absorbed into the object surface, and thus the surface also has a set of parameters, namely, the surface roughness r, the amount of absorbed water w, the rate for absorption k_a, the saturated amount of absorption a, and the amount of sediment D. Regarding the deposits carried by water particles, there are attributes such as the adhesion rate constant k_S, the solubility rate constant k_D, the evaporation rate I_{sun}, and the initial deposition amount on the surface I_D. Dorsey et al.'s model uses a scalar surface roughness r or a displacement map on the surface in order to obtain the interesting movement of particles; the roughness makes particles disperse whereas the displacement map let particles move slowly across a bumpy surface, yielding more sediment along cracks and valleys. Consequently, the absorption of water is modeled as follows;

$$\frac{\partial m}{\partial t} = -k_a \frac{a-w}{a}\frac{A}{m}, \tag{3.1}$$

$$\frac{\partial w}{\partial t} = k_a \frac{a-w}{a}\frac{m}{A} - I_{sun}, \tag{3.2}$$

where t denotes the simulation time and A is the diameter of the water particle. Similarly, the solution and sedimentation of deposits are modeled as follows;

$$\frac{\partial S}{\partial t} = -k_S S + k_D D\frac{m}{A}, \tag{3.3}$$

$$\frac{\partial D}{\partial t} = k_S S\frac{A}{m} - k_D D + I_D. \tag{3.4}$$

When the mass m becomes smaller than a certain threshold, the particle is removed from the simulation.

3.2.2 Our simulation model

Basic Modifications to Dorsey et al.'s Model.

As described above, we modify Dorsey et al.'s model for interactive simulations on a 2D image. We define the xy coordinate system along the horizontal and vertical edges of the image. To accelerate the simulation, we ignore the interaction among particles and frictions but only consider the gravity. According to Eq. (3.1), (3.2), (3.3) and (3.4), the variations of the mass and amount of absorbed water depend on the diameter of particles, which means the "thickness" of the deposit color changes according to the diameter. This is not desirable for our purpose because we want to control the "thickness" only by I_d (i.e., *Deposition Amount* in Section 3.1.2), and thus we omit m/A and A/m from the equations. In Eq. (3.3) and (3.4), we set the range of k_D and k_S [0, 1], and let $k_S = 1 - k_D$ to reduce the number of parameters. The user can control k_a and k_D as *Absorptivity* and *Deposition Resolvability* respectively, as described in Section 3.1.2.

Displacement due to Luminance Variations.

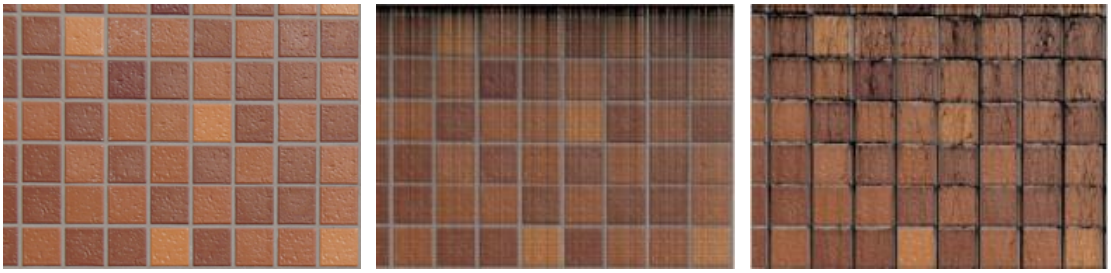
As shown in Figure 3.9(a), the luminance values change greatly in general across cracks and valleys in images. Therefore, we construct the displacement map from the variations of the luminance values in the input image, and use the surface roughness r as a coefficient to amplify the influence of the map. The particle velocity $\mathbf{v} = (v_x, v_y)$ is reduced according to the variations of the luminance values;

$$v_x^{diffuse} = b\, r\, \xi\, M_y(\mathbf{x}), \tag{3.5}$$

$$\frac{\partial v_x}{\partial t} = v_x^{diffuse} - v_x \frac{M_x(\mathbf{x})}{r_{max} - r + 1}, \tag{3.6}$$

$$\frac{\partial v_y}{\partial t} = -v_y \frac{M_y(\mathbf{x})}{r_{max} - r + 1}, \tag{3.7}$$

where $M_x(\mathbf{x})$ and $M_y(\mathbf{x})$ are the horizontal and vertical variations of luminance values at particle position $\mathbf{x}$, b is a constant, $\xi \in [-1, 1]$ is a random value, and r_{max} is the maximum of r. The user can control r as *Surface Roughness* as described in Section 3.1.2. Figure 3.9 shows the results (c) with and (b) without accounting for the surface roughness. Compared to Figure 3.9(b), Figure 3.9(c) exhibits more sedimentation along the ditches, which makes the result look much more realistic.



(a) (b) (c)

Figure 3.9: Simulation results (c) with and (b) without accounting for the displacement defined by the variations of luminance values in (a) the input image.

Adjustment to the Perspective in the Image.

Inspired by Eisenacher et. al 's texture synthesis method [28], we use a control mesh on a surface on an object to reproduce water flow stains according to perspective on input image. The control mesh is represented as a curved or rectangular surface, and its shape is deformed by moving multiple control points of a Bézier patch. We empirically use a cubic

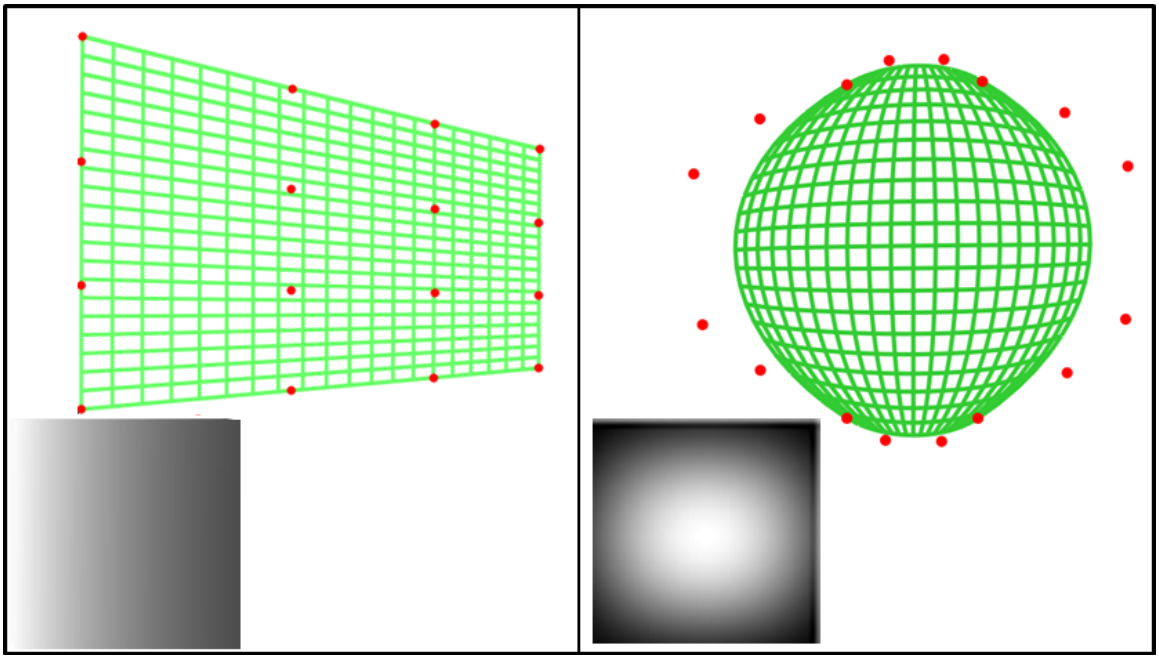


Figure 3.10: The upper right: control mesh. The lower left: visualized results of $\rho(\mathbf{x})$ on the depth map calculated from each control mesh. The maps are rectangle because these are defined in a parameter space. The darker the locations, the father from the viewpoints

Bézier patch consisting of 4×4 control points because a user can easily and flexibly edit the control mesh using the appropriate number of control points. Specifically, the cubic Bézier patch is formulated as follows:

$$\mathbf{S}(u,v) = \sum_{j=0}^{3}\sum_{i=0}^{3} \mathbf{b}_{ij} B_j^3(v) B_i^3(u) \quad (0 \leq u, v \leq 1), \tag{3.8}$$

where $\mathbf{b}$ is the coordinate of the control points, and B^3 is cubic Bernstein Basis Polynomials.

A user specifies the control mesh as follows; First, the user moves the four-cornered control points in order to globally adjust the effect of perspective. In this process, the other control points are automatically positioned according to the adjusted perspective effect. In particular, a 3×3 homography matrix is computed on the basis of the coordinates of the original four-cornered control points and moved control points, and then the coordinates of the other control points is computed by multiplying the original control points by the homography matrix (see the left in Figure 3.10). Finally, to fit the control mesh to the curved surface, the control points are individually moved as shown in the right in Figure 3.10.

While water flow stains are reproduced on an object surface according to a Bézier patch in a simulation space, actual particle coordinates is computed in a parameter space and then

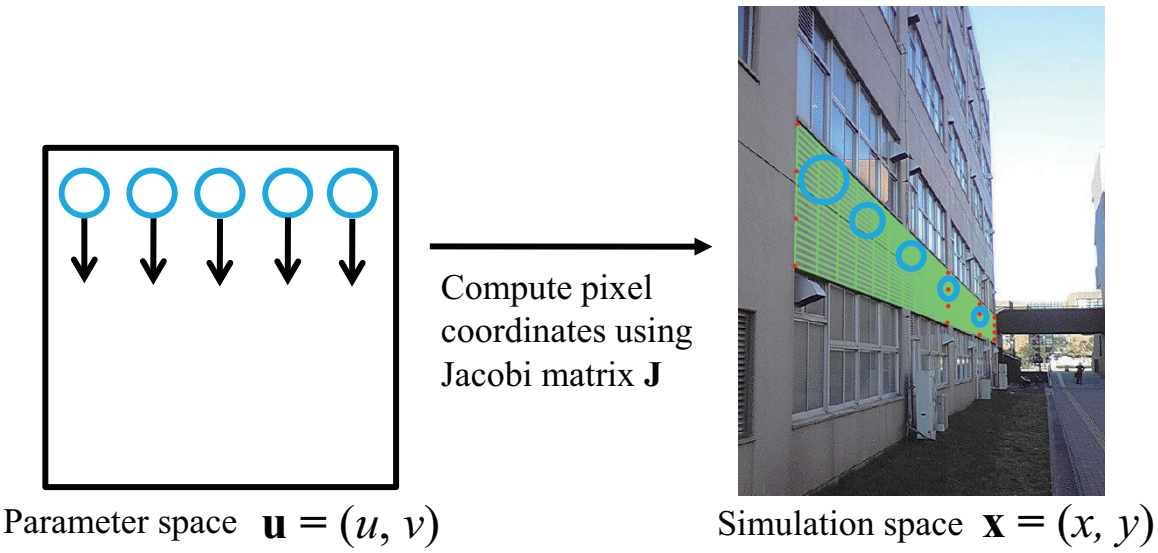


Figure 3.11: Particle simulation using the control mesh. The pixel coordinates of particles on the simulation space is computed from the coordinates of the particles in the parameter space using a Jacobi matrix $\mathbf{J}$ calculated from the adjusted control mesh.

projected onto the simulation space as shown in Figure 3.11. That is, particle coordinates $\mathbf{x} = (x, y) = \mathbf{S}(u, v)$ in the simulation space is computed from the coordinates $\mathbf{u} = (u, v)$ in a parameter space. Particle coordinates in the simulation space can be computed using Equation (3.8). However, this procedure on each timestep of simulation is too heavy. To reduce time complexity, we compute particle coordinates in the simulation space using a Jacobi matrix $\mathbf{J}$ obtained from Equation (3.8) as follows:

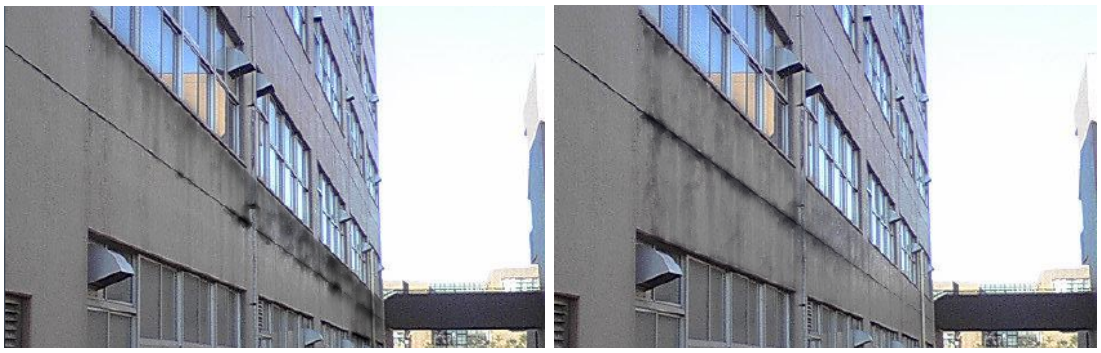
$$\mathbf{x}(\mathbf{u}) = \mathbf{x}(\mathbf{u}') + \mathbf{J}(\mathbf{u})(\mathbf{u} - \mathbf{u}'), \tag{3.9}$$

$$\mathbf{J}(\mathbf{u}) = \begin{pmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{pmatrix}, \tag{3.10}$$

where $\mathbf{u}'$ denotes particle coordinates in the parameter space at a previous timestep.

In this simulation scheme, we need to refine a displacement map according to a control mesh because the velocity of particles depends on the displacement map. For this purpose, we generate a displacement map in a parameter space based on the control mesh. In particular, we compute the variation of pixel intensity on each coordinate in a parameter space on the basis of the correspondence between coordinates in simulation and parameter spaces. Then the velocity of particles are obtained from Equations (3.5) and (3.7) using the displacement map.

Note that the farther the view point from the water flows, the thinner the water flows seem. However, this fact is not considered in above-mentioned simulation scheme based on a control mesh. Besides, when particles exist on denser regions of a control mesh, they



(a) (b)

Figure 3.12: Comparison of the results without (left) and with (right) using control mesh.

stay on the same pixels for a longer time, and thus the amount of stains becomes larger. Essentially, the amount of stains must not depend on the distance between the target and viewpoint.

To address this issue, we adjust the amount of stains and size of particles according to intervals between cells in a control mesh. These parameters are automatically adjusted by multiplying each parameter and a coefficient $\rho(\mathbf{x})$ based on Jacobian $\det(\mathbf{J}(\mathbf{x}))$ that can be used to estimate intervals between cells:

$$\rho(\mathbf{u}) \quad = \quad \frac{|\det(\mathbf{J}(\mathbf{u}))|}{\max\limits_{0\leq u,v\leq 1}\{|\det(\mathbf{J}(\mathbf{u}))|\}}, \tag{3.11}$$

where $\rho(\mathbf{u})$ takes a value within the range of [0, 1]. The lower left images in Figure 3.10 illustrate the visualization results of $\rho(\mathbf{u})$ in parameter spaces based on the control meshes in the upper right. Darker locations mean smaller $\rho(\mathbf{u})$ and farther ones from a viewpoint. The results show that coefficients can be appropriately computed according to perspective on rectangular and curved surfaces.

Figure 3.12 shows results with and without using a control mesh. In the former results, the water flow stains uniformly adhered to the wall. In contrast, in the latter results, the water flow stains adhered to the wall according to the perspective.

3.3 Results

We implemented our prototype system using C++ language, OpenGL and GLUI, and executed our system on a PC with a Xeon 2.66GHZ, 2GB RAM, and an NVIDIA Quadro FX 3450 graphics card. Figure 3.13 shows the resultant images with water flow stains synthesized using our system. The user succeeded to design realistic stains within moderate

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

(e) [2] (f) (g) [3] (h)

Figure 3.13: (b)(d)(f)(h) Synthesized images with water flow stains using our system. The image sizes of their inputs are (a) 500×375, (c) 512×384, (e) 452×491, and (g) 300×500 pixels. The times for editing are about (b) 3 min, (d) 10 min, (f) 4 min, and (h) 5 min.

time. Figure 3.14 shows the resultant images without and with applying the control mesh to the curved surface. As can be seen in these results, it is obvious that the control mesh was effective for generating natural water flow stains according to the curved surface.

Regarding the time for editing, the user generally spent the most of time in tuning parameters. As the image size becomes larger, the time for simulations takes longer as shown in Figure 8. We believe that the time for simulations will be much shorter using minified images and GPU implementation in future work.

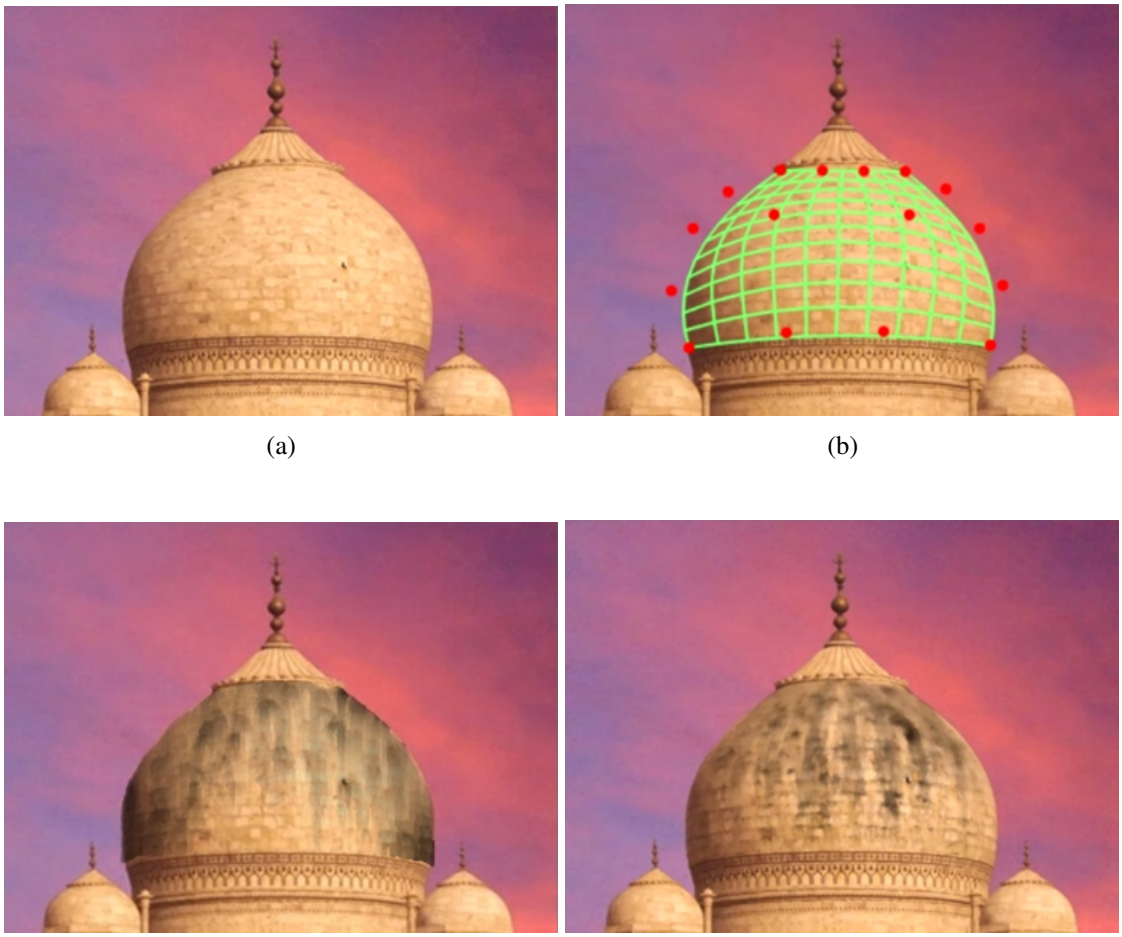
User Test.

To validate the effectiveness of our system, we performed a user test to compare its performance to the *Adobe Photoshop*®. Six students, all novice users of our system and image editing software, participated in the study. After a brief tutorial, each subject was asked to synthesize water flow stains on the tile image (Figure 3.17(b)) in reference to the stained

tile image (a real photograph, Figure 3.17(a)), using either our system or the alternative software (Photoshop). The testing order of the software was alternated between subjects; three used Photoshop first, and the other three used our prototype system first. The subjects were allowed to work on the task until satisfied, for up to 20 minutes. Figure 10 shows some of the resulting images. Figure 3.16(a) shows the time of design process across six subjects and two design tools. Subjects using our system overall took less than 70% of the time when they did using Photoshop. Figure 3.16(b) shows the subjective evaluation of the produced images. Ten people voted on the resulting images, regarding how natural each image is. While a paired t test of the operation time does no show statistical significance, a binomial test of the scores shows that our system performed better than the Photoshop in making water flow stains.

[2]Author of the input image: new hobby
URL: http://www.flickr.com/photos/newhobby/2427677211/

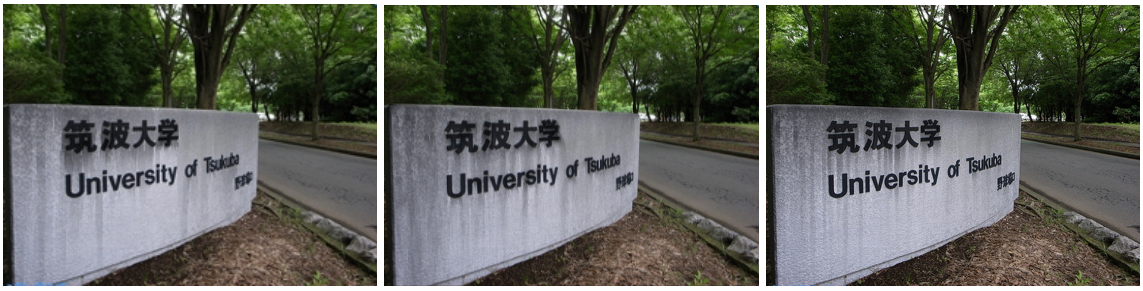
[3]Author of the input image: peterjr1961
URL: http://www.flickr.com/photos/peterjr1961/3506811726/



(a) (b)

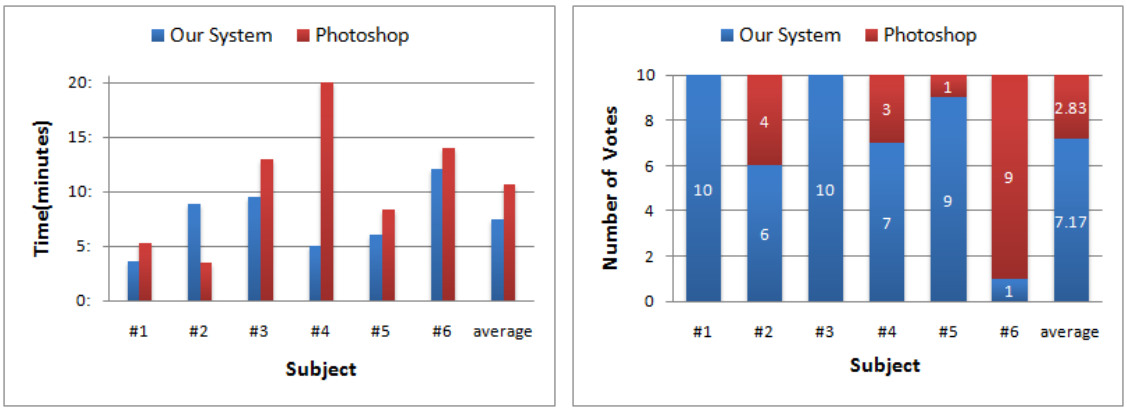
(c) (d)

Figure 3.14: The resultant images without (c) and with (d) applying the control mesh (b) to input image (a).



(a) 256 × 192, 0.35 sec (b) 500 × 375, 3.5 sec (c) 1024 × 768, 9.9 sec

Figure 3.15: Resultant images with different image sizes for the same scene as Figure 3.3. The image size and simulation time for a single execution are listed under each image.



(a) Time for operations (b) Votes on the qualities

Figure 3.16: Graphs summarizing the results of a user test. Each of six subjects was asked to synthesize two images, one using our system and the other using Photoshop, so that the images became satisfying for the subject. (a) shows the time of design process across six subjects and two design tools. (b) Then ten people voted on the result, regarding how natural each image is. The number represents that of votes for each system.

(a) Reference example (b) Given image

(c) Our system # 1 (d) Our system # 4 (e) Our system # 6

(f) Photoshop # 1 (g) Photoshop # 4 (h) Photoshop # 6

Figure 3.17: Results of the user test, comparing our system to Photoshop. The users were presented (a) a reference example (a real photograph) and asked to edit (b) a given image until the user gets satisfied. The middle row (c)(d)(e) presents the results by different users using our system whereas the bottom row (f)(g)(h) shows those using Photoshop.

Chapter 4

Example-based Weathering with Geometric Details

Outdoor objects in the real world change their appearance due to weathering over time. For example, metals get rust, paints get peeled, and stones become mossy. Reproduction of such weathering phenomena is very important to enhance the reality of object appearance in computer graphics.

Whereas there have been many techniques that handle weathering of 3D models, there are only a few for objects in 2D images. Recently, Xue et al. [95] as well as Bandeira and Walter [5] proposed methods for reproducing weathering in images. Both of their methods construct the models of the time-variant appearance of objects, *appearance manifolds* [95] or *appearance maps* [5], and calculate the distribution of weathering degrees in the image based on the models. The weathering degrees are then used to decompose the image into reflectance and illuminance, which is often interpreted as *shading* in this context. Consequently, de/weathering is accomplished by changing only the reflectance according to weathering degrees while the illuminance (shading) remains unchanged. This framework allows applications to objects with complicated shading, but cannot handle time-varying shading due to weathering, that is, shading caused by time-varying geometric details of rust, peeled paint or moss. Another concern on these methods is the usability. While these methods are not physically accurate, they produce plausible results to some extent. However, any user interface that allows local and interactive editing is not provided.

In this chapter, we propose a technique for modeling weathering effects with time-varying geometric details in images. Specifically, we focus on spatio-temporal variations of shading as well as reflectance due to weathering. We extract fine-scale details of shading as high-frequency components of the shading image, and add them onto object surfaces as weathering progresses. This extraction can be done with only a few additional user inputs. De/weathering with time-varying shading is accomplished in our work for the first time, and substantially enhances the reality of de/weathering effects as shown in Figure 4.1. Moreover, we introduce a brush-like user interface for editing weathering effects locally

and interactively. This interface allows the user to edit arbitrary weathering distributions, for example, similar to those observed in the real world.

4.1 The Method of Bandeira and Walter

Similar to Xue et al.'s method [95], Bandeira and Walter's method [5] models an input image as the product of reflectance and illuminance, and assumes that only the reflectance changes due to weathering phenomena.

To decompose an input image into reflectance and illuminance, they use *Lab color space* that allows to handle luminance (L) and chroma (ab) separately. They assume that chroma values of illuminance are constant and chroma values of reflectance are given as ab channels of each pixel. On the other hand, luminance values of reflectance and illuminance are modeled as follows.

$$I_l(i,j) = W_l(i,j) \times S(i,j) \tag{4.1}$$

where i and j are horizontal and vertical coordinates of each pixel, I_l is a luminance value of the image, W_l is a reflectance luminance value represented as the weathering component, S is an illuminance luminance value represented as the shading component. The reflectance luminance values W_l are computed based on a weathering degree map.

The weathering degree map is calculated with an appearance map that is constructed in advance based on the least weathered point and the most weathered point selected by the user. Weathering degree values are within the range of [0, 1] where 0 indicates the least weathered and 1 indicates the most weathered.

The reflectance luminance values are the average of luminance values of pixels whose weathering degrees are almost the same. The illuminance luminance values are calculated as $S(i,j) = I_l(i,j)/W_l(i,j)$. After the decomposition, the weathering degree map is updated, and then the deweathered or weathered reflectance values are computed with the appearance map and the weathering degree map. Finally, a resulting image is obtained by combining reflectance with illuminance.

4.2 Modeling Weathering Effects with Geometric Details

We handle de/weathering effects where both illuminance (shading) and reflection change simultaneously. Following the method of Bandeira and Walter [5], the region of interest (ROI) in the input image is decomposed into illuminance and reflectance based on a weathering degree map computed with the appearance map (see Figure 4.3).

The left image in Figure 4.2 shows a photograph of a rusting iron surface, where not only reflectance but also shading varies due to the geometric variations. The right image in Figure 4.2 illustrates the intensity plots along a scanline of illuminance (shading) in the

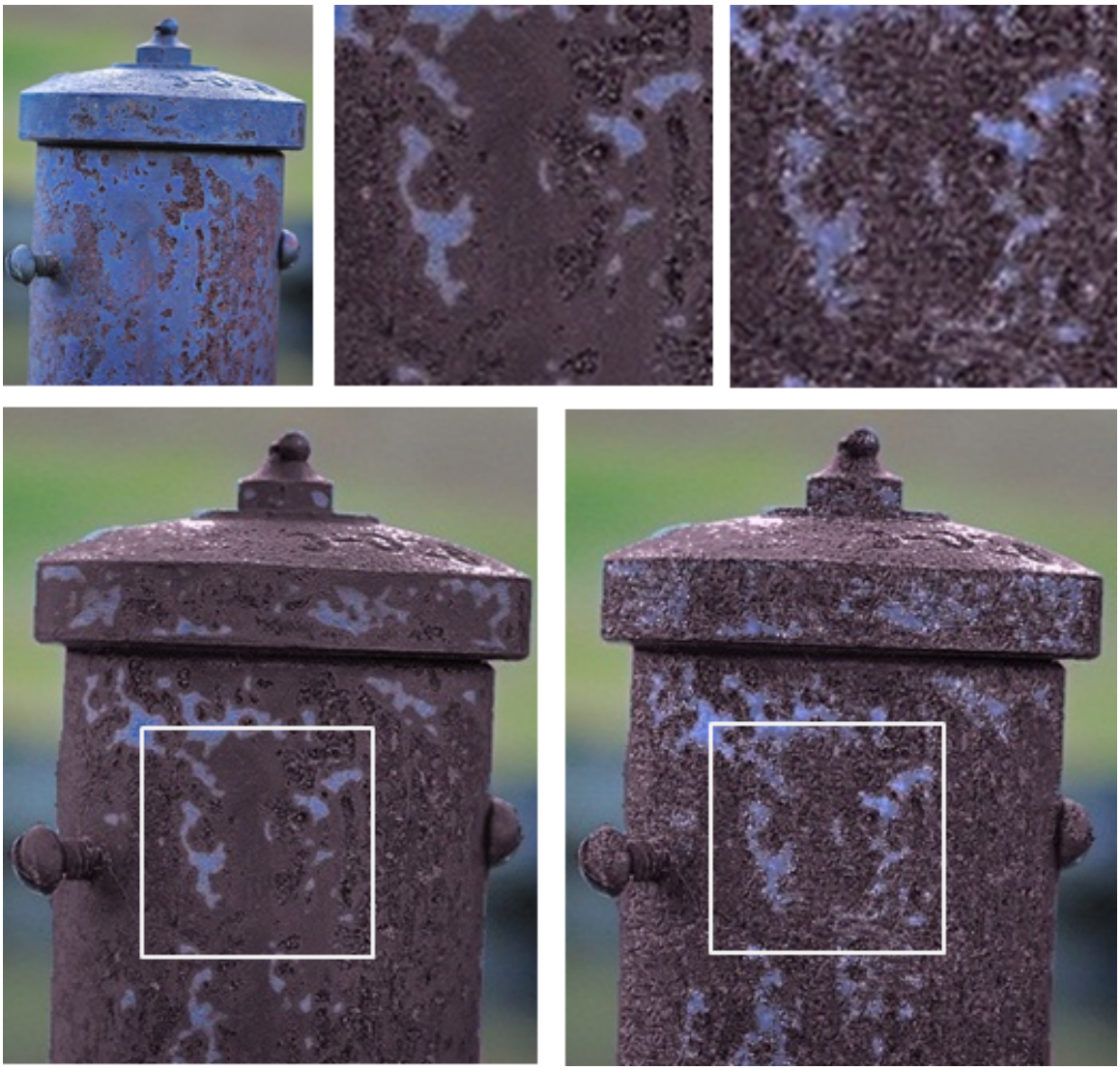


Figure 4.1: Example images of rust effects. Top left: the input image. Bottom left: a resultant image by Bandeira and Walter [5]. Bottom right: a resultant image by our method, with geometric details caused by rusting. Each magnified figure is shown at the top right.

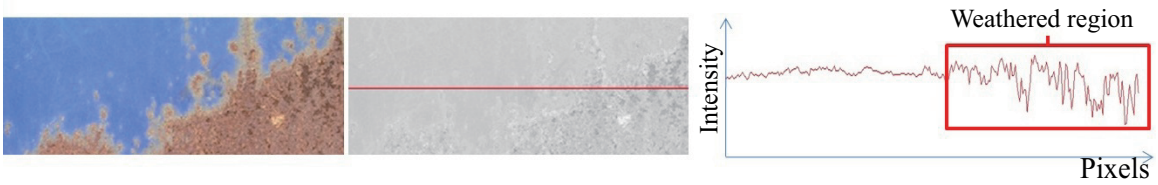


Figure 4.2: A typical example of the fact that weathered regions often become rough. Left: A photograph of a rusting surface. Middle: illuminance of the left image. Right: intensity plots along a red scanline of illuminance.

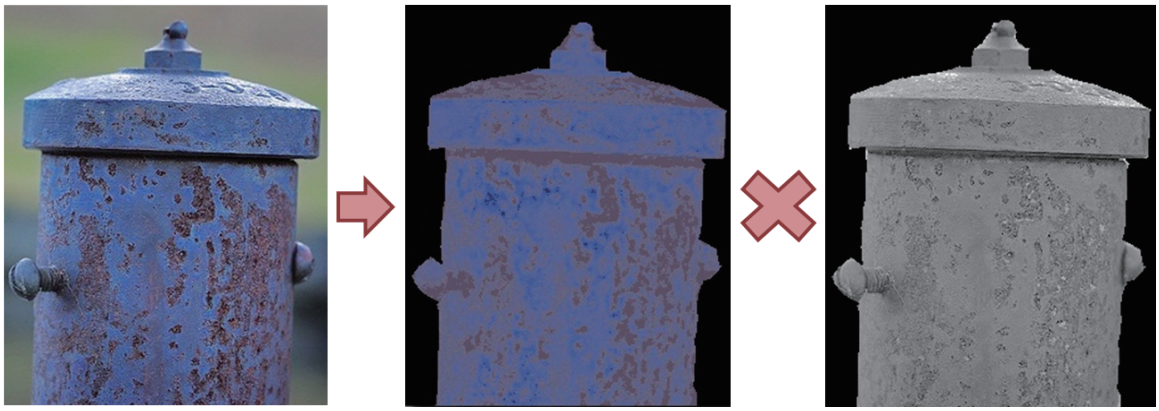


Figure 4.3: Decomposition of the input image into reflectance and illuminance.

middle image in Figure 4.2. The high-frequency pattern found here is typical in weathered regions in images.

We assume that the geometric variations caused by weathering are represented as high-frequency components in shading. As a preprocess, we extract the high-frequency patterns and fill the ROI with the patterns using a texture synthesis method. Then, we add these synthesized patterns onto illuminance according to the updated weathering degrees. Reflectance is also calculated similary to Bandeira and Walter's method [5]. Finally, we obtain resulting images by multiplying reflectance and illuminance. In the following sections, we describe each of these steps in detail.

4.2.1 Extraction of shading details

Given an illuminance image S, we decompose S into coarse features and fine features using a multi-scale image decomposition presented by Subr et al. [78]. Because this decomposition can separate the fine texture from shading if the texture and shading are of different scales, it works well for our purpose of extracting the high-frequency patterns. The illuminance image S is decomposed into coarse features S_c and fine features S_f as follows:

$$S(i,j) = S_c(i,j) + S_f(i,j) \tag{4.2}$$

The decomposed result is shown in Figure 4.4.

4.2.2 Texture synthesis

. To fill the ROI with extracted high-frequency patterns, we employ a texture synthesis algorithm by Ashikhmin [4]. This algorithm runs very fast and constructs results that often look good. Although it sometimes brings abrupt discontinuity depending on an input texture, it suffices for our purpose because the input texture in our case is just a rough pattern that does not require continuity.

The synthesis process is shown in Figure 4.5. To ensure that the source texture pattern is obtained only from the weathered region, we calculate a binary mask M from the weathering distribution map d.

$$M(i,j) = \begin{cases} 1 & (d(i,j) \geq t) \\ 0 & (d(i,j) < t) \end{cases} \tag{4.3}$$

where t denotes a weathering threshold. We found $t = 0.5$ worked well for our experimental results. The source texture pattern is sampled from regions where $M(i,j) = 1$ in order to obtain synthesized fine features S'_f. Note that the binary mask may contain non-weathered regions if the weathering degree map is constructed inaccurately, resulting in a failure of texture synthesis. This problem can be avoided by letting the user to manually

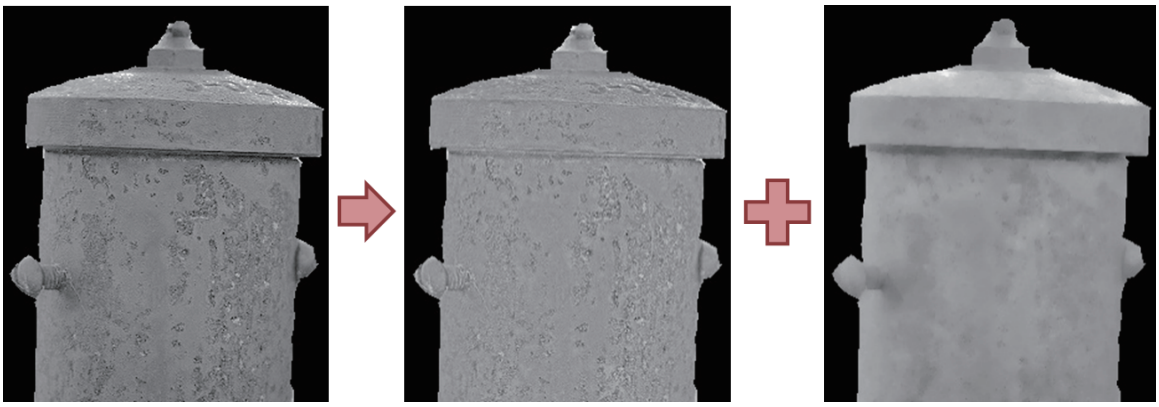


Figure 4.4: Decomposition into coarse features and fine features.

specify a rectangular region in which the binary mask is construcated and the source texture is sampled. Fortunately, such additional user input is not required in most cases by setting the rectangular region around the most weathered point specified when constructing the appearance map.

4.2.3 Weathering with geometric details

After the preprocess described above, the resultant illuminance is computed with the synthesized fine shading components S'_f, the initial illuminance S, the initial weathering degree map d and the updated weathering degree map d' as shown in Figure 4.6. The fine components are added to the regions where weathering progresses to a certain extent:

$$S'(i,j) = \begin{cases} S(i,j) + S'_f(i,j) & (d'(i,j) - d(i,j) \geq t) \\ S(i,j) & (d'(i,j) - d(i,j) < t) \end{cases} \tag{4.4}$$

On the other hand, the reflectance variations are calculated using the appearance map according to the weathering degree for each pixel. Finally, a resutant image with detailed shading is obtained by mutiplying the shading and reflectance values.

4.2.4 Deweathering with geometric variations

We also try deweathering taking into account the geometric variations, i.e., recovery of the original surfaces of objects before going through weathering. This is a challenging task in general because the original surfaces are unknown. In contrast to weathering described in Section 4.2.2, we synthesize the high-frequency components of illuminance in non-weathered regions, hence we invert the mask: $M'(i,j) = 1 - M(i,j)$. Because non-weathered regions are often smooth, the resultant illuminance also tends to be smooth.

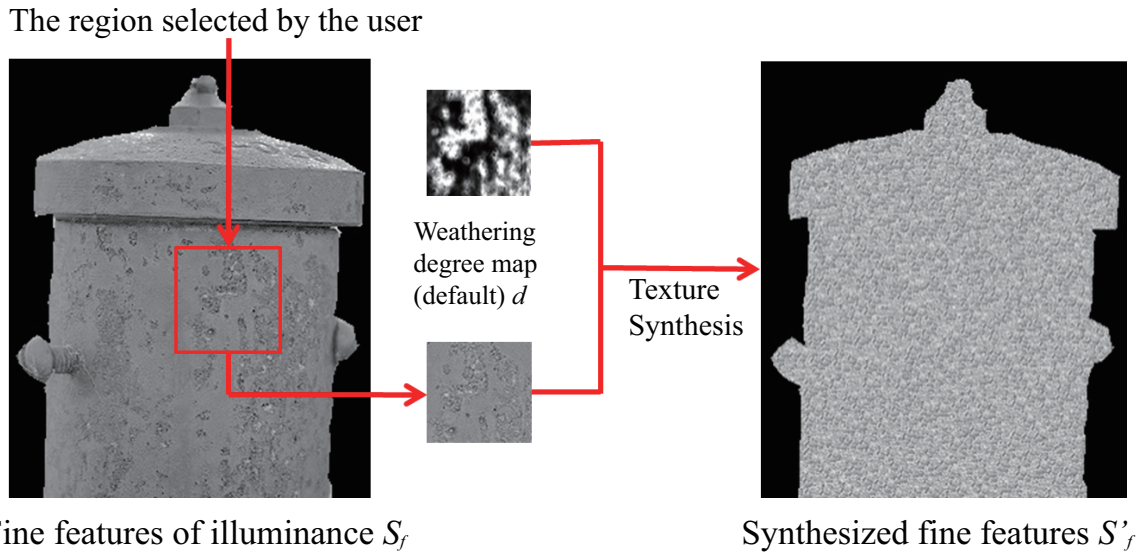


Figure 4.5: An outline of the synthesis. Fine features of illuminance are synthesized in a weathered region based on a weathering degree map.

Using the synthesized components S'_f, the resultant illuminance is defined as:

$$S'(i,j) = \begin{cases} S(i,j) & (d'(i,j) - d(i,j) \geq t) \\ S_c(i,j) + S'_f(i,j) & (d'(i,j) - d(i,j) < t) \end{cases} \tag{4.5}$$

A comparison of deweathering effects between the previous method [5] and our method is shown in Figure 4.7. We could eliminate geometric details caused by weathering to some extent. Here our approach assumes that the original surfaces are smooth; otherwise separating the fine-scale geometries from the original surface becomes quite difficult unless extra knowledge is provided.

4.3 Weathering Transfer with Geometric Details

Our method can be used for weathering transfer between one image to another, taking into account the geometric variations. For explanation, we use the terms 'source' and 'target' to distinguish the source of weathering effects and the target to be modified, respectively. Reflectance is calculated using the appearance map for each of source and target images, similar to Bandeira and Walter [5]. To transfer shading variations, we prepare the synthesized fine components and the binary mask of the source image. The weathering effects in the source image is then transferred according to the weathering degree map of the target image.

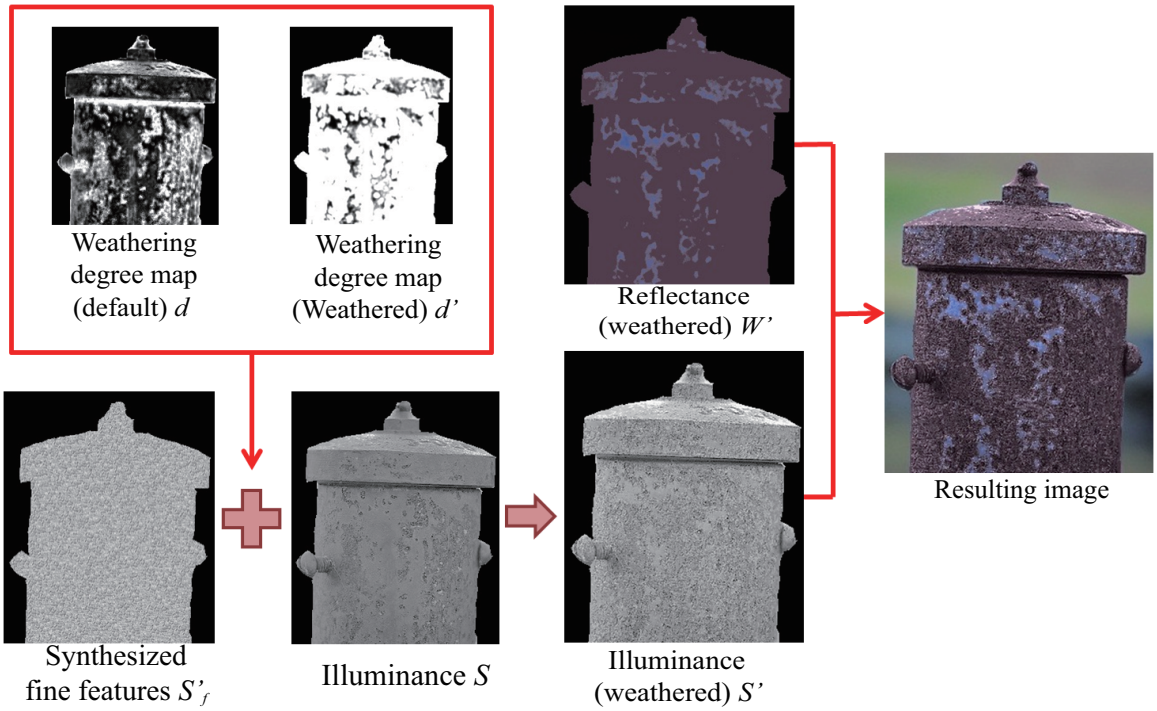


Figure 4.6: An overview of synthesis of weathering effects with geometric variations. The weathered illuminance is calculated based on the weathering degree map and the synthesized fine features. The resultant image is obtained by multiplying the precomputed reflectance and synthesized illuminance.

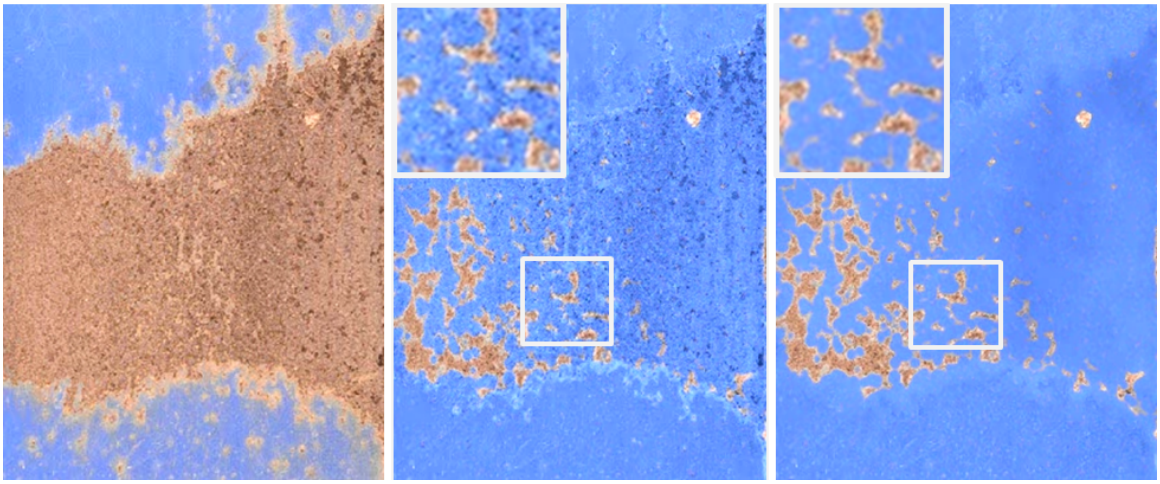


Figure 4.7: A comparison of deweathering processes. Left to right: the input image, a deweathered result by Bandeira and Walter [5] and a deweathered result by our method. Magnified images are shown at upper left of the resuls.

Alternatively, as an initial weathering degree map for the target image, we can use the weathering degree map of the source image as well. In this case, we synthesize the weathering degree map using Ashikhmin's method [4] to adjust the map size to the target image size, and use the synthesized map as an initial weathering degree map. This yields good weathering effects similar to those in the source image, as shown in Figure 4.9.

4.4 A De/Weathering Brush Tool

Although the previous methods [5, 6, 95] can synthesize nice weathering effects, they cannot emulate the real process of weathering faithfully due to the ignorance the physical laws. Indeed, the speed of weathering is different by region due to the difference of exposure to environmental factors such as rains or winds. However, finding out such information from the input image is difficult.

We provide a brush tool that allows the user to locally edit de/weathering effects. The user can select weathering or deweathering, and adjust the speed of de/weathering. The user can intuitively edit weathering effects by drawing brush strokes.

For the progress of weathering effects within a brush stroke, we use the weathering factor parameter K_w described in [5]. We set $K_w = 0$ for the outside of the brush and $K_w = 1$ for the inside, or use a Gaussian fall-off for local weathering effects.

Figure 4.8 illustrates an example edited with our brush tool. We took a minute in the designing session.

Figure 4.8: An editing result with our brush tool. The red circular regions are weathered and the green circular region is deweathered. The side-by-side comparisons before/after editing are shown above.

Figure 4.9: A result of weathering transfer with geometric details. To reproduce the weathering distribution of the source material, we mapped a weathering degree map synthesized from the source material onto the target object. Top left: the input image. Top right: the source material. Bottom left: a weathered result. Bottom right: a more weathered result.

4.5 Results

Our implementation was written in C++, using OpenGL, GLUT and GLUI. We ran our program on a PC with an Intel Core i7 2.80 GHz CPU and an NVIDIA Quadro FX 580 GPU. The input images we used in Figure 4.2, Figure 4.7, and Figure 4.9 were directly taken from the paper of Xue et al. [95], and the others were downloaded from flickr (http://www.flickr.com/).

The sizes of the input images and the time for preprocessing (Sections 4.2.1 and 4.2.2) are shown in Table 4.1. Although the preprocessing took a few seconds, it does not matter as it does not affect the editing process. The time required to update a weathering degree map and to calculate reflectance and illuminance is almost the same as Bandeira and Walter's method [5] because the additional process is just to incorporate the high-frequency components that are synthesized in the preprocess. The time required for editing with our brush tool relies on the brush size. The user could edit in real time using a rather large brush with radius 400 pixels.

Figure 4.1 and Figure 4.10 show comparisons between our method and the previous method [5] to reproduce rusting and mossy effects. The bottom left in Figure 4.10 was locally edited with the brush tool. Note that in our results, the weathering effects with geometric details look more realistic compared to the results by the previous method. Fig-

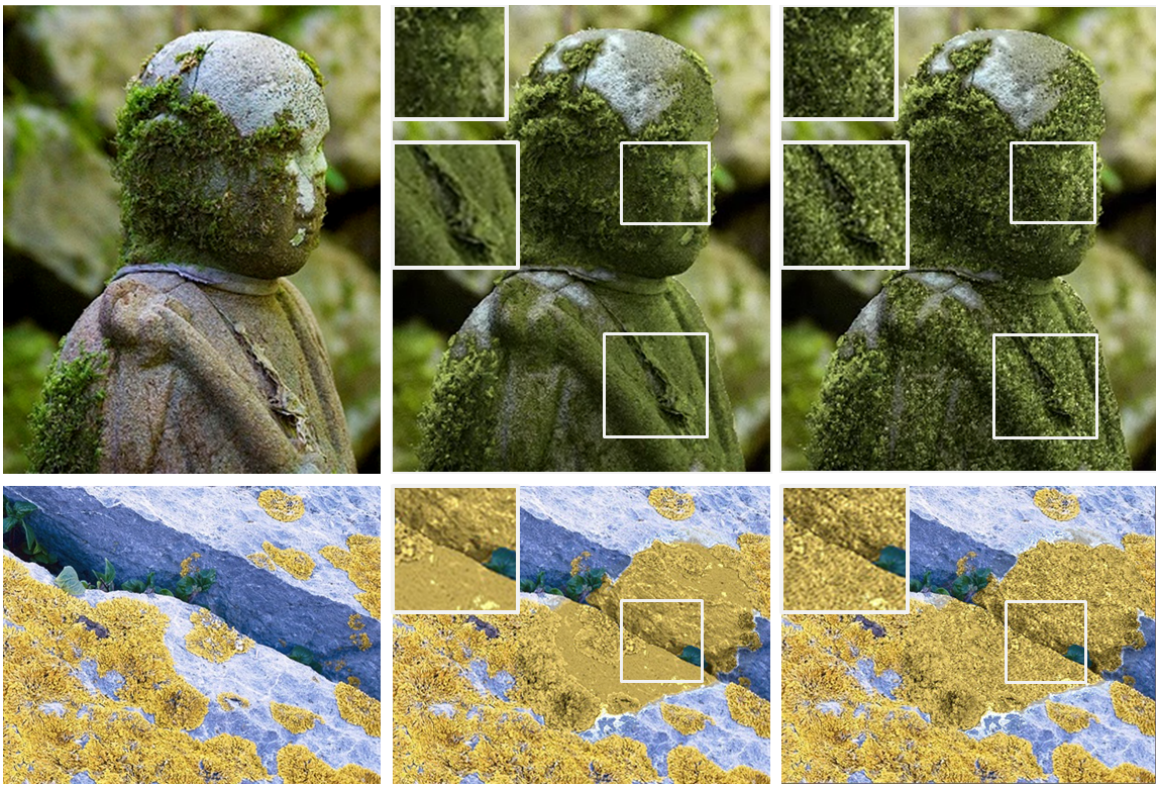


Figure 4.10: Comparisons of mossy effects. Left: input images. Middle: resultant images by Bandeira and Walter [5]. Right: resultant images by our method.

Figure 4.11: Comparisons of rusting effects. Left: input images. Middle: resultant images by Bandeira and Walter [5]. Right: resultant images by our method.

Table 4.1: The time (seconds) for preprocessing (Sections 4.2.1 and 4.2.2) required for each image.

image	size (pixels)	time
Figure 4.1	260×320	4.26
Figure 4.7	280×348	3.32
Figure 4.9, transfer material	541×291, 174×189	2.41
Top left in Figure 4.10	265×333	4.31
Bottom left in Figure 4.10	409×318	5.39
Figure 4.11	637×421	7.77
Figure 4.12, transfer material	304×625, 131×147	1.67

ure 4.11 shows a comparison of deweathering effects of a rusting car. The geometric details of rust are smoothed out as well as the inherent shape is preserved in our method. Figure 4.12 and Figure 4.9 show examples of weathering transfer with geometric details. These also achieve good appearance, and the comparisons in Figure 4.9 demonstrates the effectiveness of our method.

Figure 4.12: A comparison of weathering transfer. Left to right: a input image, a resultant image by Bandeira and Walter [5], a resultant image by our method, and magnified images (top: the previous method, bottom: our method).

Chapter 5

Reproduction of Reflection on Surfaces

Beautiful sights reflected on water surfaces of the sea or a lake are favorable photographic subjects, as we see many on the Internet. Water surfaces can behave just like a flat mirror, while wavy surfaces yield interesting warping in reflection images. Such effects are caused by *Fresnel reflection*, which consists of a linear combination of reflected and transmitted light. It can be also observed in our daily lives, e.g., on a table top or a glossy floor.

Editing photographs with such effects would be quite interesting, especially with reflection on wavy surfaces. Indeed, beautiful images/animations including reflection on wavy surfaces can be created from a single photograph, with a plenty of manual labor [18, 64]. For semi-automating this making process, matting plays the key role; one first separates the reflected image into a reflection component, transmission component, and alpha matte, and then composite new objects using this information. However, matting is a challenging problem because it is ill-posed, and applying previous matting methods for optical phenomena (e.g., shadow [17, 88] and haze [80, 31, 36, 42]) to Fresnel reflection is difficult due to the difference of targets and incompatibility of formulations. Additionally, the matting method for *refraction* in a glass [99] does not support complicated *reflection* observed at water surfaces.

In this chapter, we propose a matting method that handles Fresnel reflection in a single image based on user markups. Targeting reflection at surfaces such as the surface of deep water, glossy table top or floor, we introduce the following three assumptions; 1) the transmission color can be approximated as uniform, 2) a pair of an object that is the source of reflection and the corresponding reflection image can be found in the input image, and 3) the reflection surface is mostly planar but possibly wavy. We first estimate the transmission color based on the first and second assumptions, using *color transfer* [69]. We then roughly estimate the reflection component and alpha matte as well as camera parameters, assuming the alpha matte is smooth. However, the alpha matte should contain high-frequency regions in case of wavy reflection surfaces. We thus propose a novel filter to refine the alpha matte, which is validated using ground-truth data. Using the calculated information, we also provide a compositing system with which the user can composite new objects onto the input

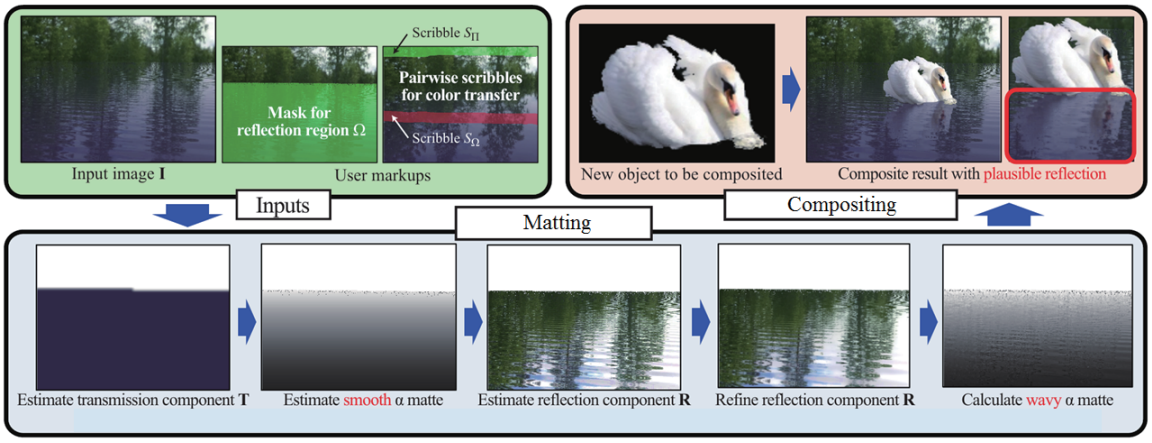


Figure 5.1: Overview of our system. To solve the matting problem, the user specifies the region of reflection surface and the pairwise scribbles. After the several matting steps, a composite result with plausible reflection can be obtained.

image with plausible reflection in real time.

Figure 5.1 illustrates the overview of our system. In the matting stage, our algorithm takes as input a single image including a reflection surface and a user-specified region of the reflection surface as well as a pair of scribbles for color transfer. Our matting algorithm computes a reflection component, a transmission component and an alpha matte, and then update the reflection component and the alpha matte to handle the effect of waves (Section 5.1). With this matting information, we subsequently render reflection images of newly composited objects using ray tracing (Section 5.2).

Note that this is the first attempt to extract the complicated reflection and alpha matte for Fresnel reflection in a single image. We demonstrate the effectiveness of our method using various examples (Section 5.3).

5.1 Reflection Matting

We start by defining our reflection matting problem and the assumptions for solving the problem. We then describe our matting algorithm that separates each component of an input image.

5.1.1 Reflection model and assumptions

In our work, we target reflection caused by the Fresnel effect, which occurs at an interface between substances with different refractive indices. The observed light intensity at the interface is a linear blending of those of the reflected light and transmitted light. For an

image, we have

$$\mathbf{I}(\mathbf{x}) \;=\; \alpha(\mathbf{x})\mathbf{R}(\mathbf{x}) + (1-\alpha(\mathbf{x}))\mathbf{T}(\mathbf{x}), \tag{5.1}$$

where $\mathbf{I}$, $\mathbf{R}$ and $\mathbf{T}$ are the 3-channel (RGB) input image, reflection component and transmission component, respectively. $\alpha \in [0,1]$ is the Fresnel coefficient and $\mathbf{x} = (x,y)^T$ is a pixel coordinate.

The Fresnel coefficient depends on the incident angle of the viewing ray. At the air-water interface, for example, α monotonically increases as the incident angle becomes sufficiently large, and thus the Fresnel coefficient becomes small at the near side and large at the far side of the camera, as shown in the alpha mattes in Figure 5.1 (each pixel intensity encodes the Fresnel coefficient). Correspondingly, the transmitted color becomes dominant at the near side while the reflected color dominant at the far side.

Solving for Eqn. (5.1) from a single input image $\mathbf{I}$ is an ill-posed problem because we have seven unknowns with only three knowns per pixel. Although this might seem similar to natural image matting [83] that decomposes an image into foreground and background components and an matte, or the recent refraction matting [99], the targets and formulations are different as described in Section 2.1.2. We instead focus on the typical cases observed at surfaces of deep water, a glossy table top or floor, which we see in many photographs on the Internet. Specifically, we make the following three assumptions.

1. The transmission component $\mathbf{T}$ can be approximated as a uniform color.

2. At least one real object and the corresponding reflection image can be found in the input images.

3. The reflection surface is mostly planar but possibly wavy.

Throughout this chapter, we use the term *real object* to denote an object that is the source of reflection above the reflection surface, in contrast to the corresponding reflection image. Our matting algorithm takes as inputs a single photograph including a reflection surface and a user-specified mask for the region of the reflection surface as well as a pair of scribbles for color transfer. We denote the real-object region and the reflection region in the input image as region Π and region Ω, respectively. Our matting algorithm calculates $\mathbf{T}$, $\mathbf{R}$ and α for region Ω. Region Π is above the upper part of boundary $\partial\Omega$ of region Ω.

Based on these assumptions, our matting algorithm proceeds in the several steps (see Figure 5.1). Because the transmission component $\mathbf{T}$ is assumed uniform (i.e., constant) over the entire region and easier to solve than the other components, we first estimate $\mathbf{T}$ (Sec. 5.1.2). We next calculate an α matte according to the Fresnel equation using viewing rays' incident angles calculated based on the camera parameters that we estimate (Sec. 5.1.3). We then compute the reflection component $\mathbf{R}$ using the other estimated information (Sec. 5.1.3). Note that the reflection surface can be wavy. We refine the reflection component $\mathbf{R}$ and α matte using our novel filter (Sec. 5.1.4). Algorithm 1 shows the

overview of our matting algorithm. In the following subsections, we describe the details of these steps.

Algorithm 1 Reflection matting.

Input: $\mathbf{I}$, reflection region Ω and pairwise scribbles $\{S_\Pi, S_\Omega\}$
Output: $\mathbf{T}$, $\mathbf{R}$ and α
$\mathbf{I}_{ds} \leftarrow$ DOWNSAMPLING($\mathbf{I}$);
$\mathbf{R}_\Omega \leftarrow$ COLORTRANSFER ($\mathbf{I}_{ds}$, S_Π, S_Ω); // Sec. 5.1.2
$\mathbf{T} \leftarrow$ CALCTRANSMISSIONCOLOR($\mathbf{I}_{ds}$, $\mathbf{R}_\Omega$); // Sec. 5.1.2
$\{\alpha(\mathbf{p}), y_p\} \leftarrow$ COLLECTSAMPLES($\mathbf{I}_{ds}$, $\mathbf{T}$, Ω); // Sec. 5.1.3
$(\theta_o,\phi) \leftarrow$ CALCCAMERAPARAM($\{\alpha(\mathbf{p}),y_p\}$); // Sec. 5.1.3
$\alpha \leftarrow$ CALCALPHAMATTE(θ_o, ϕ, Ω); // Sec. 5.1.3
for each $\mathbf{x} \in \Omega$ **do**
  $\mathbf{R}(\mathbf{x}) \leftarrow \frac{\mathbf{I}(\mathbf{x})-(1-\alpha(\mathbf{x}))\mathbf{T}}{\alpha(\mathbf{x})}$;
end for
if reflection region Ω is wavy **then**
  $\mathbf{R} \leftarrow$ APPLYSMOOTHINGFILTER($\mathbf{R}$, Ω); // Sec. 5.1.4
  for each $\mathbf{x} \in \Omega$ **do**
    $\alpha(\mathbf{x}) \leftarrow \frac{(\mathbf{I}-\mathbf{T})\cdot(\mathbf{R}(\mathbf{x})-\mathbf{T})}{\|\mathbf{R}(\mathbf{x})-\mathbf{T}\|^2}$;
  end for
end if

5.1.2 Estimating transmission component $\mathbf{T}$

Assuming the transmission component is uniform, $\mathbf{T}$ in Eqn. (5.1) becomes a constant vector.

$$\mathbf{I}(\mathbf{x}) \;=\; \alpha(\mathbf{x})\mathbf{R}(\mathbf{x}) + (1-\alpha(\mathbf{x}))\mathbf{T}. \tag{5.2}$$

We then consider the gradient of luminances of Eqn. (5.2).

$$\nabla I(\mathbf{x}) \;=\; (R-T)\nabla\alpha(\mathbf{x}) + \alpha(\mathbf{x})\nabla R(\mathbf{x}), \tag{5.3}$$

where I, R and T are the luminances of $\mathbf{I}$, $\mathbf{R}$ and $\mathbf{T}$, and $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y})^T$ is the gradient operator. If the reflection surface is sufficiently flat, the α matte is smooth because differences of incident angles are very small, and thus $\nabla\alpha$ is negligible. We therefore approximate Eqn. (5.3) as

$$\nabla I(\mathbf{x}) \;\approx\; \alpha(\mathbf{x})\nabla R(\mathbf{x}). \tag{5.4}$$

However, this approximation might not be valid in case of wavy surfaces because incident angles can vary greatly. To avoid the effect of wavy surface, we downsample the input

image using the Gaussian filter. For an input image of 600×400 pixels, for example, we scale it down to 1/4.

If we know α and $\mathbf{R}$, we can calculate $\mathbf{T}$ from Eqn. (5.2) just using pixels only in a partial region of reflection region Ω because $\mathbf{T}$ is constant. On the other hand, α can be computed using $\mathbf{I}$ and $\mathbf{R}$ according to Eqn. (5.4). We therefore seek for $\mathbf{R}$ in reflection region Ω locally. Ideally, $\mathbf{R}$ in reflection region Ω can be obtained by finding the corresponding region in real-object region Π and using the real-object colors, based on our second assumption. However, accurate alignment of the pair is difficult because in reflection region Ω the colors are changed due to the Fresnel effect and the shapes are also changed due to waves compared to those of real-object region Π. We thus do not find the exact matching of the pair, but bring the color distribution in Ω close to the distribution in Π using *color transfer* [69]. We use the color-transferred $\mathbf{I} \in \Omega$ as $\mathbf{R}$.

The pairwise regions for color transfer are specified by user-provided scribbles S_Π and S_Ω. As shown in Figure 5.1 as "User markups" in "Inputs", the user specifies a region in real-object region Π using a single green scribble S_Π, and the corresponding region in reflection region Ω using a single red scribble S_Ω. The colors in scribble S_Π are then transferred to those in scribble S_Ω to estimate the reflection component $\mathbf{R}$ in reflection region Ω. Note that, in this stage, we do not require $\mathbf{R}$ for every pixel in entire reflection region Ω because we just collect samples to estimate $\mathbf{T}$.

Color transfer works well due to the following reason. In scribble S_Ω, α can be considered constant because the region is vertically short, and then $\mathbf{I}$ in the region becomes scaled $\mathbf{R}$ with a constant offset according to Eqn. (5.2). That is, $\mathbf{R}$ in scribble S_Ω can be obtained by scaling the variance of $\mathbf{I}$ in scribble S_Π and offsetting the mean. We thus use color transfer to conform the mean and variance of $\mathbf{I}$ in scribble S_Ω to those in scribble S_Π.

Finally, we calculate $\mathbf{T}$ using the information we obtained above. Now having pixel samples of $\mathbf{R}$ in scribble S_Ω, we calculate α from Eqn. (5.4) by projecting $\nabla I(\mathbf{x})$ onto $\nabla R(\mathbf{x})$.

$$\tilde{\alpha}(\mathbf{x}) = \frac{\nabla I(\mathbf{x}) \cdot \nabla R(\mathbf{x})}{\|\nabla R(\mathbf{x})\|^2}. \tag{5.5}$$

Note that $\tilde{\alpha}$ is an intermediate value to estimate $\mathbf{T}$. We omit samples if $\|\nabla R(\mathbf{x})\|^2 < 0.01$ to reduce error caused by division. We calculate $\mathbf{T}$ for each sample using $\tilde{\alpha}$ based on Eqn. (5.2), and then average the values to reduce error.

5.1.3 Estimating α matte and reflection component $\mathbf{R}$

Whilst we obtain α values only in a partial region S_Ω in the previous subsection, we now estimate the entire α matte. The goal of this stage is to obtain a smooth α matte for capturing the entire (i.e., low-frequency) variation of the matte, and then calculate the corresponding $\mathbf{R}$. Here we again use the downsampled $\mathbf{I}$ as input to avoid the effect of waves. To reli-

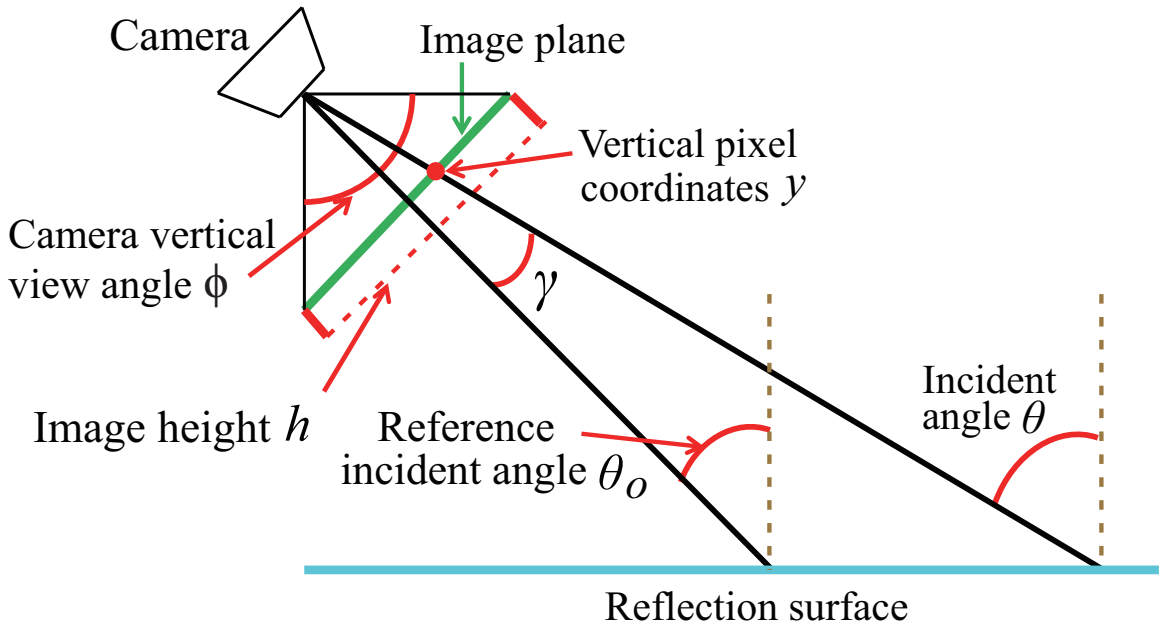


Figure 5.2: Relationship between a reflection surface and camera parameters.

ably estimate a smooth α matte, we exploit the analytic formula of the Fresnel coefficient. Specifically, we use the Schlick's approximation [74] for the Fresnel coefficient as follows.

$$\alpha(\theta) \quad = \quad \alpha_0 + (1 - \alpha_0)(1 - \cos(\theta))^5, \tag{5.6}$$

and, for a planar reflection surface, the incident angle θ of a viewing ray is given by

$$\theta(y) = \gamma(y) + \theta_o, \quad \gamma(y) = \arctan\left(\frac{2y}{h}\tan(\frac{\phi}{2})\right), \tag{5.7}$$

where h is the image height, y is the vertical pixel coordinate, ϕ is the vertical view angle of the camera and θ_o is the reference incident angle (see Figure 5.2) for the gazing direction of the camera. α_0 is a constant determined by the refraction index n of material as $\alpha_0 = \frac{n-1}{n+1}$. In our experiments, we simply use fixed values for refraction indices n according to materials; $n = 1.33$ for water, for example. Note that roughly specifying a refraction index does not matter because small variation of n does not affect the value of Eqn. (5.6) significantly.

To determine the incident angles θ, we have to estimate camera parameters θ_o and ϕ. We therefore solve the following minimization problem using the Lebenberg-Marquart method.

$$\underset{\theta_o,\phi}{\operatorname{argmin}} \sum_{\mathbf{p}\in P} \left|\alpha(\mathbf{p}) - \left(\alpha_0 + (1-\alpha_0)(1-\cos(\theta(y_p)))^5\right)\right|^2 \tag{5.8}$$

where $P = \{\mathbf{p} = (x_p, y_p) \,|\, \mathbf{p} \in \Omega\}$ is a set of positions of training samples. Although Eqn. (5.8) is non-linear, we found it converges to an appropriate minimum when we set $\theta_o = 70^\circ$ and $\phi = 30^\circ$ as initial values. Note that ϕ can be also specified from the EXIF data of the input photograph, and the refraction index n can be also estimated in the optimization of Eqn. (5.8).

We collect samples $\{\alpha(\mathbf{p}), y_p\}$ in entire region Ω, not in a partial region as done in the previous subsection, to avoid bias. To compute $\alpha(\mathbf{p})$ as a training sample, we search for an appropriate reflection-color sample in real-object region Π. Let $\mathbf{R}(\mathbf{q})$ $(\mathbf{q} \in \Pi)$ be the color of a reflection-color sample. We seek for the best $\mathbf{R}(\mathbf{q})$ based on the following metric ϵ.

$$\epsilon(\mathbf{R}(\mathbf{q}), \mathbf{T}) = \|\mathbf{I}(\mathbf{p}) - (\hat{\alpha}(\mathbf{p})\,\mathbf{R}(\mathbf{q}) + (1 - \hat{\alpha}(\mathbf{p}))\mathbf{T})\|, \tag{5.9}$$

$$\hat{\alpha}(\mathbf{p}) = \frac{(\mathbf{I}(\mathbf{p}) - \mathbf{T}) \cdot (\mathbf{R}(\mathbf{q}) - \mathbf{T})}{\|\mathbf{R}(\mathbf{q}) - \mathbf{T}\|^2}, \tag{5.10}$$

where $\hat{\alpha}(\mathbf{p})$ corresponding to the smallest ϵ is used as $\alpha(\mathbf{p})$. Both Eqns. (5.9) and (5.10) are derived from Eqn. (5.2). ϵ measures how close to $\mathbf{I}(\mathbf{p})$ the linear blending of $\mathbf{R}(\mathbf{q})$ and $\mathbf{T}$ is. Intuitively, ϵ represents the distance between point $\mathbf{I}(\mathbf{p})$ and a segment connecting points $\mathbf{R}(\mathbf{q})$ and $\mathbf{T}$ in RGB color space. Smaller ϵ means a better sample. The use of this metric is inspired by *robust matting* [82], where both background and foreground samples are collected based on the metric for natural image matting. Note that we do not use the map $\hat{\alpha}$ as an α matte but use it just as a set of samples because it may contain substantial noise and errors.

To efficiently search for appropriate samples $\mathbf{R}(\mathbf{q})$ in real-object region Π, we limit the search space so that corresponding real objects are likely to be found. Let d be the vertical distance between the pixel in question and the upper part of the boundary $\partial\Omega$ of the reflection region Ω, as shown in Figure 5.3. Within a search window W_ϵ (we use a 7×7 window in our experiments) centered at the vertically symmetric position of $\mathbf{p}$, we choose a sample that has the smallest ϵ.

$$\mathbf{q} = \underset{\tilde{\mathbf{q}} \in W_\epsilon}{\operatorname{argmin}}\ \epsilon(\mathbf{R}(\tilde{\mathbf{q}}), \mathbf{T}). \tag{5.11}$$

In case that d is too large and the corresponding search window protrudes out of the image, we just ignore such sample at $\mathbf{p}$. For reliable estimate of camera parameters, we collect 20% pixels in reflection region Ω with smallest ϵ.

In summary, we calculate α and $\mathbf{R}$ as follows. We first estimate camera parameters using Eqn. (5.8) as well as training samples based on Eqns. (5.9) and (5.10). We then obtain smooth α for the entire region Ω using the estimated camera parameters as well as Eqn. (5.6), and then calculate $\mathbf{R}$ via Eqn. (5.2). We update these two components in the next subsection to account for wavy reflection surfaces.

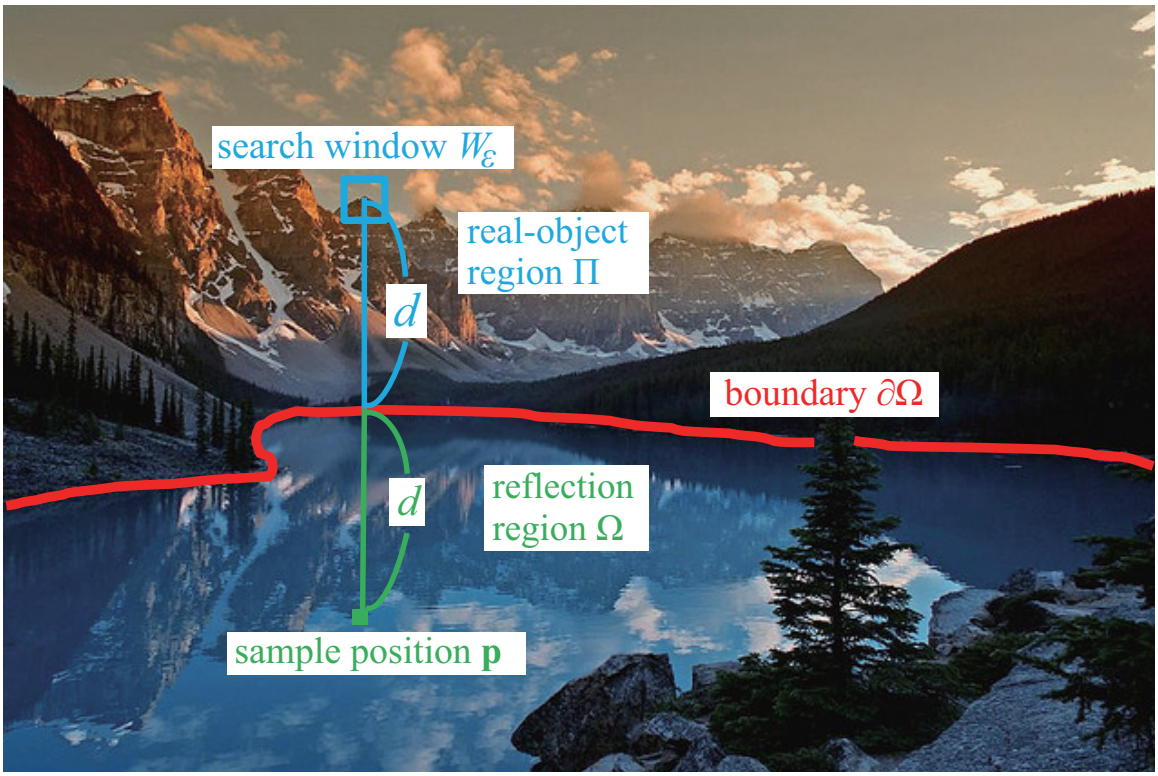


Figure 5.3: To calculate $\alpha(\mathbf{p})$ as a training sample, we search for a pixel that has the smallest ϵ within search window W_ϵ (blue square, enlarged for illustration purpose), and use it as sample $\mathbf{R}(\mathbf{q})$. The center of search window W_ϵ is vertically away from the boundary $\partial\Omega$ by distance d.

5.1.4 Updating $\mathbf{R}$ and α

Here we update $\mathbf{R}$ and α to account for wavy surfaces, where the α matte should contain high-frequency regions because of varying incident angles of viewing rays. However, the α matte obtained so far is smooth, and the current $\mathbf{R}$ contains the high-frequency residuals.

For this issue, we smoothen the residuals in $\mathbf{R}$ and then compute α using the smoothed $\mathbf{R}$ from Eqn. (5.2). Note that we should preserve inherent edges of the reflection and smoothen only the residuals. However, typical edge-preserving filters including the bilateral filter [81] are not optimal because they preserve not only edges of true $\mathbf{R}$ but also edges of high-frequency residuals, which should be smoothened. Therefore, we design a

new filter as a variant of the bilateral filter, and apply it to $\mathbf{R}$ as follows.

$$\mathbf{R}^*(\mathbf{x}) = \frac{\sum_{\mathbf{y}\in W_\mathbf{x}} G(\mathbf{x},\mathbf{y}) w(\mathbf{x},\mathbf{y}) \mathbf{R}(\mathbf{x})}{\sum_{\mathbf{y}\in W_\mathbf{x}} G(\mathbf{x},\mathbf{y}) w(\mathbf{x},\mathbf{y})}, \tag{5.12}$$

$$w(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{(\beta(\mathbf{x},\mathbf{y}))^2}{\sigma}\right), \tag{5.13}$$

$$\beta(\mathbf{x},\mathbf{y}) = \arccos\left(\frac{(\mathbf{I}(\mathbf{x})-\mathbf{T})\cdot(\mathbf{I}(\mathbf{y})-\mathbf{T})}{\|\mathbf{I}(\mathbf{x})-\mathbf{T}\|\,\|\mathbf{I}(\mathbf{y})-\mathbf{T}\|}\right), \tag{5.14}$$

where $\mathbf{R}^*$ is the filtered $\mathbf{R}$, $W_\mathbf{x}$ is a set of pixels in the $K \times K$ filter kernel (K is the kernel size) centered at $\mathbf{x}$, G is the spatial Gaussian weight, w is our novel weight function and σ indicates its variance. Our weight function w can detect the variability of the inherent colors of the reflection between neighbor pixels. The smaller the difference between these colors is, the more our filter smoothens $\mathbf{R}$. We set $\sigma = 0.01$ for all of our experiments whilst the kernel size was adjusted for each input image according to the image-space wavelength of wavy surfaces in the reflection region Ω.

We geometrically explain weight function $w(\mathbf{x},\mathbf{y})$ of our filter using Figure 5.4. Intuitively, $\beta(\mathbf{x},\mathbf{y})$ in $w(\mathbf{x},\mathbf{y})$ is the angle between vectors $\mathbf{I}(\mathbf{x})-\mathbf{T}$ and $\mathbf{I}(\mathbf{y})-\mathbf{T}$. Note that we should smoothen $\mathbf{R}$ according to true $\mathbf{R}$, but we do not know the exact values. Instead we know $\mathbf{I}$ and $\mathbf{T}$, and vector $\mathbf{I}-\mathbf{T}$ shares the same orientation with vector $\mathbf{R}-\mathbf{T}$ because $\mathbf{I}(\mathbf{x})-\mathbf{T} = \alpha(\mathbf{x})(\mathbf{R}(\mathbf{x})-\mathbf{T})$ as derived from Eqn. (5.2). Therefore, using the known vector $\mathbf{I}-\mathbf{T}$, $\beta(\mathbf{x},\mathbf{y})$ detects the difference of true $\mathbf{R}$ between pixels $\mathbf{x}$ and $\mathbf{y}$ whilst $\beta(\mathbf{x},\mathbf{y})$ is independent of α. Consequently, our filter can preserve the edges of true $\mathbf{R}$ and smoothen the residuals in current $\mathbf{R}$ caused by the variation of α.

Figure 5.5 compares our filter with the typical filters, Gaussian filter and bilateral filter. We use the input image in Figure 5.1, which is synthesized by ray tracing with an environment map and Stokes waves. We also compare our result with ground truth for this synthesized input image. We set the kernel sizes of all the filters as 27×27 because of the relatively wide waves in the input.

In the result by the Gaussian filter in Figure 5.5(a), the filtered $\mathbf{R}$ is overall too smooth whilst α includes undesired edges of the reflected objects. These errors cause the inappropriate composite result where the reflection image of woods is blurred. The result using the bilateral filter are shown in Figure 5.5(b). The bilateral filter successfully eliminates the edges of reflected objects in α. However, it fails to smoothen waves in $\mathbf{R}$ and thus fails to capture waves in α. As a result, the reflected image of the gull in the composite result is unnaturally flat. Figure 5.5(c) shows our result. Our filter successfully alleviates the errors mentioned above. Specifically, in our result, waves in true α are extracted whilst the reflection edges are preserved similarly to the bilateral filter. Compared to the ground truth as shown in Figure 5.5(d), our result extracts each component more faithfully than others.

5.2 Reflection Composition

After decomposing the input image into each component, we compute reflection of newly composited objects using the components and camera parameters. In the compositing stage, the reflection image $\mathbf{R}_{new}$ for a composited object is synthesized to create the final composite result $\mathbf{I}_{new}$.

$$\mathbf{I}_{new}(\mathbf{x}) = \alpha(\mathbf{x})\mathbf{R}_{new}(\mathbf{x}) + (1 - \alpha(\mathbf{x}))\mathbf{T}. \tag{5.15}$$

To compute the new reflection component $\mathbf{R}_{new}$, we employ ray tracing. The composited objects are represented as billboards facing to the camera and in contact with the reflection surface. The incident angles θ of viewing rays can be estimated according to Eqn. (5.6) as follows.

$$\theta(\mathbf{x}) \;=\; \arccos\left(1 - \left(\frac{\alpha(\mathbf{x}) - \alpha_0}{1 - \alpha_0}\right)^{\frac{1}{5}}\right), \tag{5.16}$$

where $\mathbf{x}$ denotes the pixel position that a viewing ray passes through. Note that naively applying ray tracing causes aliasing in reflection images especially around object silhouettes; a reflected image in a photograph should be slightly blurred because of subpixel-scale reflection due to small waves on the surface. Therefore, we apply prefiltering using the Gaussian filter when fetching colors of real objects.

In addition, we implemented two operations for more flexible compositing. One is the contact constraint. If the object actually faces in an oblique direction, a gap might be found between the bottom of the object and the reflection surface (Figure 5.6(b)). To correct this, the user provides a scribble to roughly specify the region where the bottom of objects are actually in contact with the reflection surface (Figure 5.6(c)). Our system then extracts the bottom of the object within the scribble. The other operation is the height adjustment, with which the user adjusts the height of objects from the reflection surface using the mouse wheel. This operation allows us to calculate reflection of the objects such as birds being in the air (Figure 5.7(b)).

5.3 Experimental Results and Discussion

We implemented our prototype system as a single-threaded C++ program, and conducted experiments on a computer with Intel i7 CPU at 2.8GHz and 8GB RAM. The statistics of our matting results are summarized in Table 5.1. The compositing process is done in real time, as demonstrated in the accompanying video.

Figure 5.8 shows several results of our method with natural images including water surfaces as inputs. We can see that reflection images are appropriately created according to the Fresnel effect. That is, reflection colors are dominant on the surface at the far side

Table 5.1: The computational times (sec) for matting in our results with different image sizes and filter kernel sizes (pixel).

Image	Image size	Kernel size	Time
Figure5.1	400×400	27×27	7.92
Figure5.8(b)	640×428	5×5	0.896
Figure5.8(d)	640×450	27×27	7.68
Figure5.8(f)	600×400	13×13	1.40
Figure5.10(b)	620×403	5×5	0.801
Figure5.10(d)	615×413	(w/o filter)	0.532
Figure5.10(f)	420×634	13×13	1.37

of the viewpoint whilst dark water colors dominant at the near side. Note that warping by waves are also successfully handled especially in the second and third columns.

The rightmost column of Figure 5.8 shows a comparison between our result and real reflection in a photograph. We extracted the real object that causes real reflection in the input image and synthesized it next to the original for a side-by-side comparison. Our result is visually plausible but the shape of the reflection image is subtly different from real one because of billboard approximation. Note that such composite might be also possible using *Poisson Image Editing* [67] if a real reflection image is available, but the method might produce unnatural reflection images on waves with different wavelength or different camera angles (Figure 5.9). We would like to emphasize that Poisson Image Editing cannot work without a real reflection image, but our method can synthesize plausible reflection even without any real reflection image.

As shown in Figure 5.10, our method can also handle reflection at other surfaces, such as a wet ground (left), a table top (middle), and a glossy floor (right). In the middle example, the relative image sizes of the cups are automatically adjusted according to the scene depth according to the estimated camera parameters (Section 5.1.3) whilst the original sizes of the cups are all the same. In the left and right examples, we also consider warping by wavy reflection surfaces. Note that our method can produce plausible reflection images even though the original objects (insets in (a), (c), (e)) do not have reflection images.

Limitation. Billboard approximation used in ray tracing might result in unnatural appearance of reflection images. As shown in Figure 5.11, the the top of the tea cups are rendered as reflection images but, in reality, the top of teacups should not be seen. This artifact would be alleviated by handling the geometries of real objects.

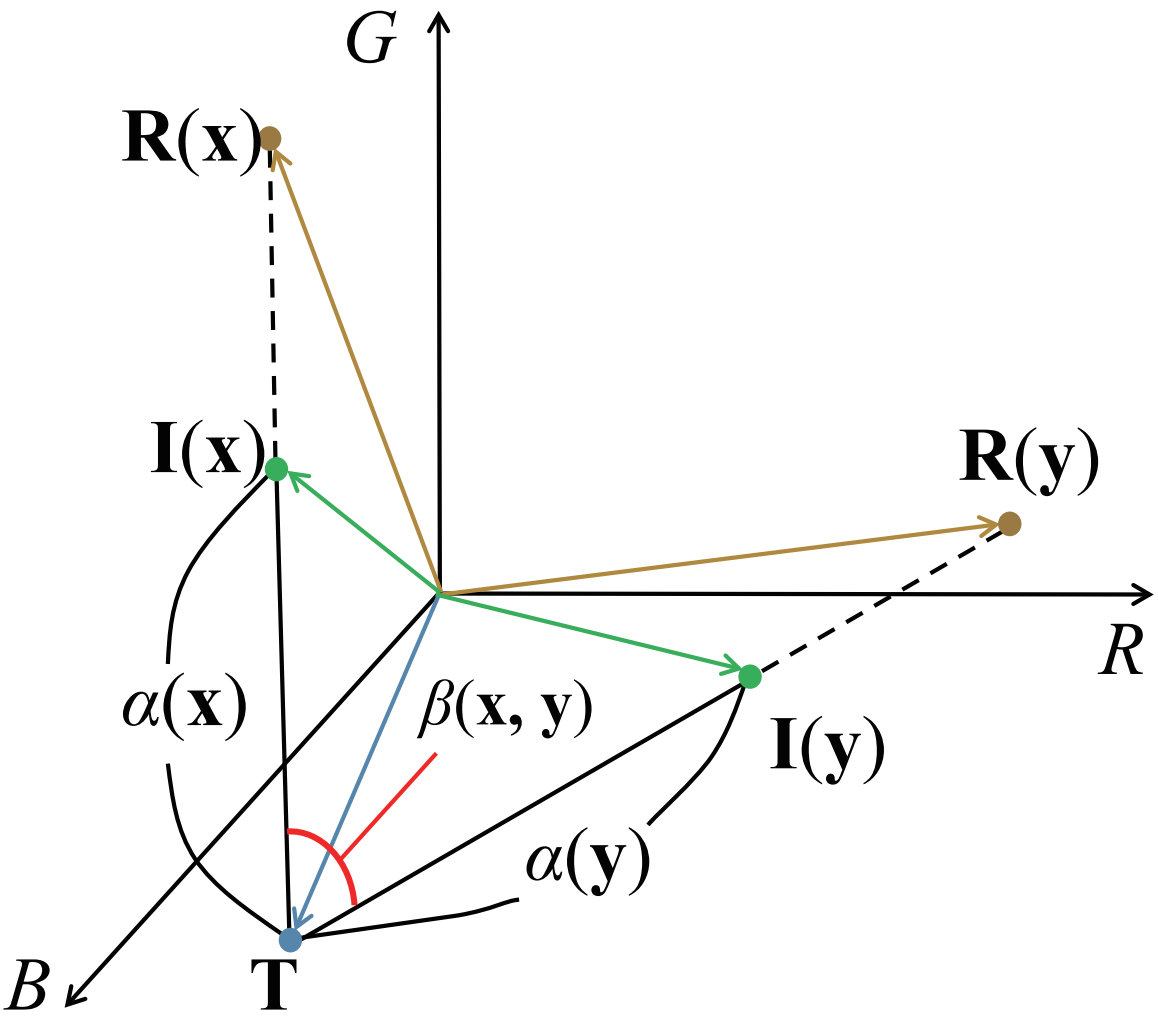


Figure 5.4: Geometric interpretation of our filter. The degree of smoothing by the filter depends on weight function β, which indicates the angle between vectors $\mathbf{I}(\mathbf{x}) - \mathbf{T}$ and $\mathbf{I}(\mathbf{y}) - \mathbf{T}$ in RGB color space, and can detects the color variation of true $\mathbf{R}$. With this weight function, our filter can smoothen waves only caused by the variation of α.

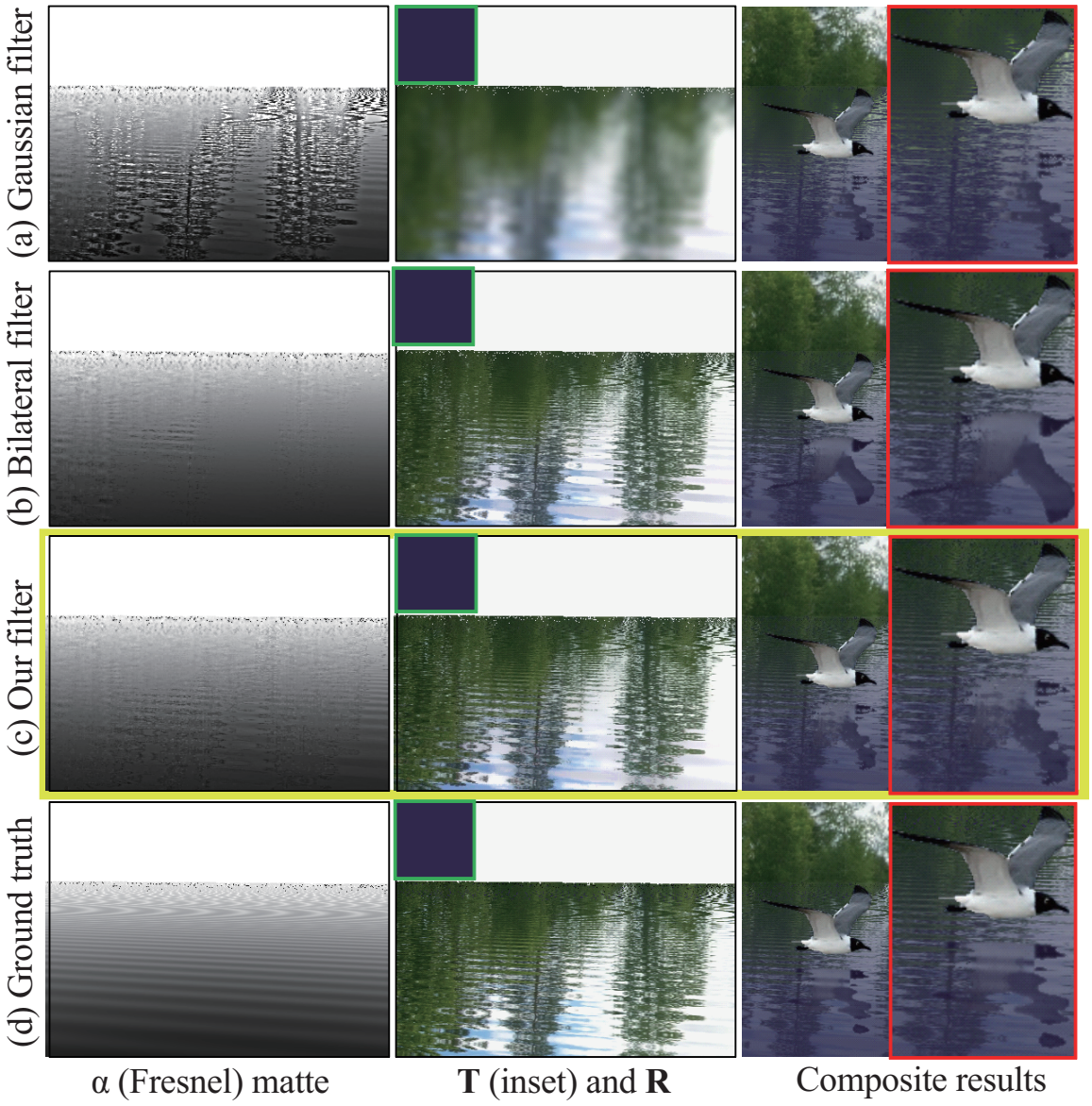
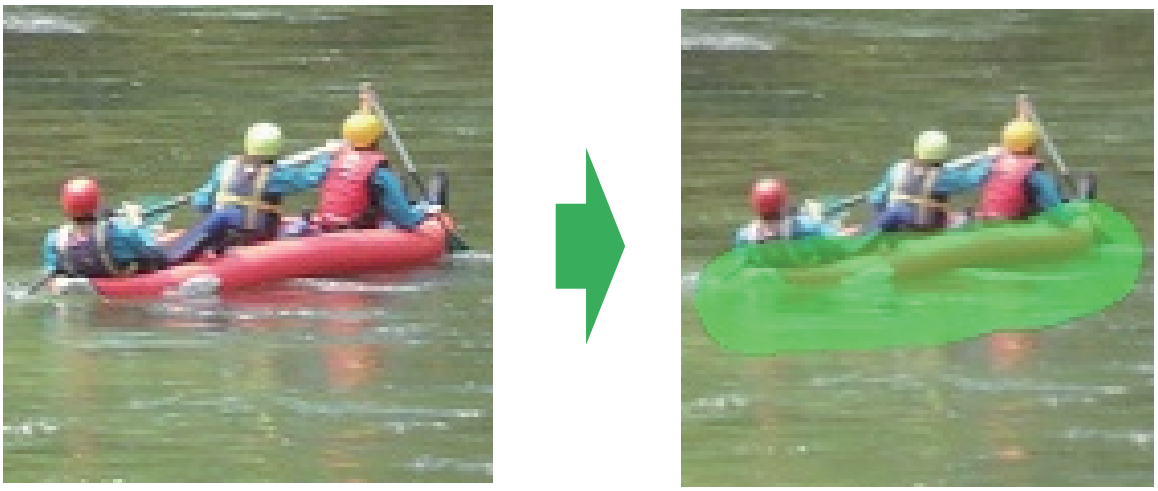


Figure 5.5: A comparison of the results between our filter and typical existing filters as well as ground-truth data. Note that our filter yields the most faithful α matte and the most plausible reflection image in the composite result.



(a) Specifying contact constraint

(b) w/o contact constraint

(c) with contact constraint

Figure 5.6: Contact constraint. (a) The user can specify the bottom of an object using a green scribble, (c) to make the object contact with the reflection surface appropriately.

(a) w/o height adjustment (b) with height adjustment

Figure 5.7: Height adjustment. The user can adjust the height of objects for flying objects using the mouse wheel.

Figure 5.8: Our synthetic results with photographs including water surfaces. Each inset shows a magnified image of our result.

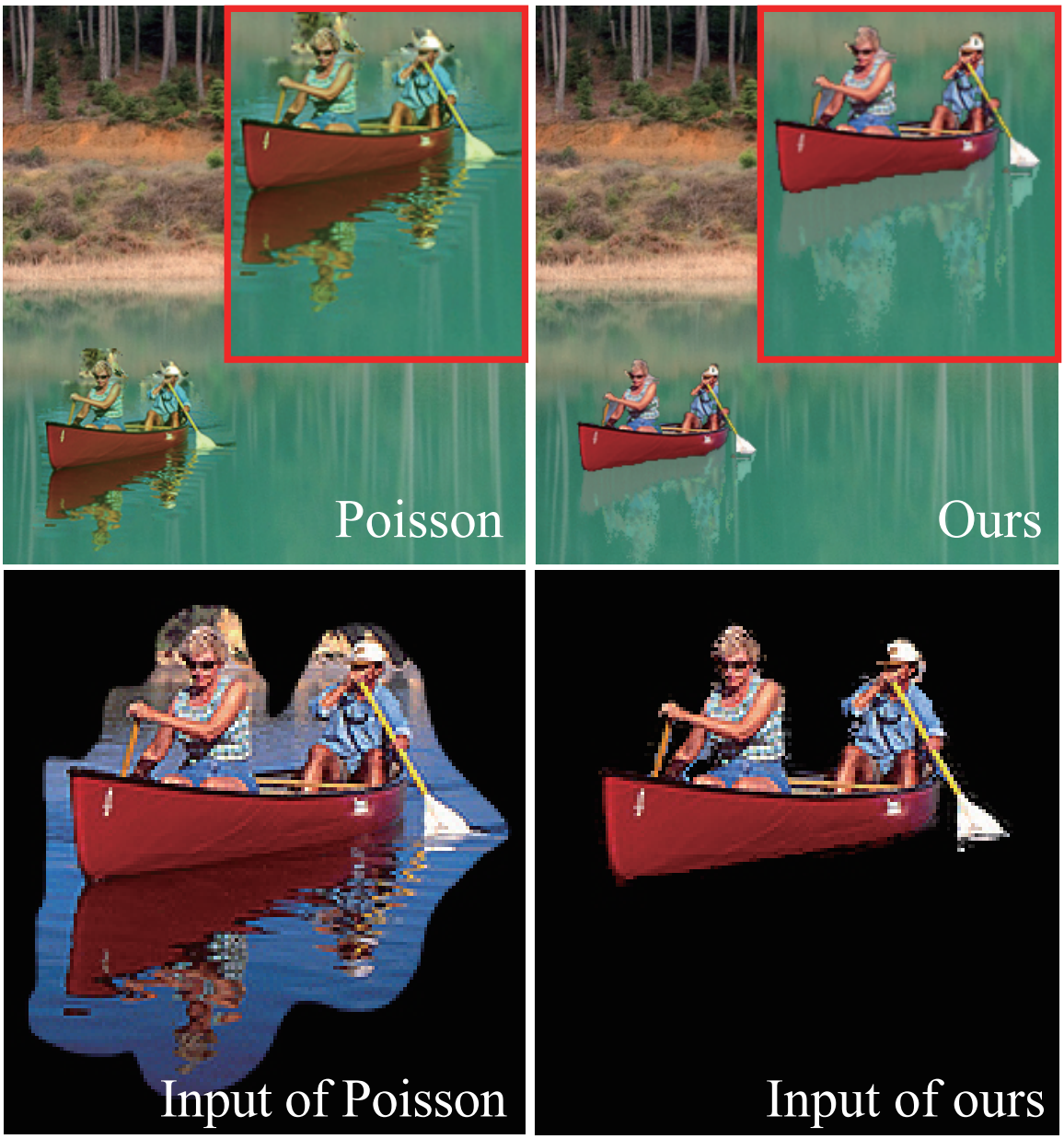


Figure 5.9: Comparison between the result of Poisson Image Editing [67] and ours. Whilst Poisson Image Editing requires a real reflection image and might produce an unnatural result due to waves of different wavelength or different camera angles, our method can synthesize plausible reflection even without any real reflection image.

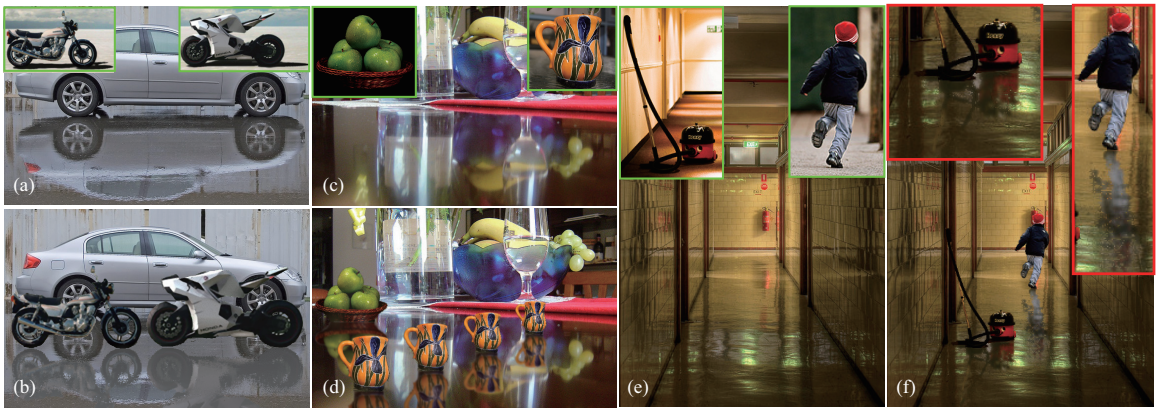


Figure 5.10: Our method can handle reflection at a wet ground (left) and off-specular reflection at glossy surfaces including a table top (middle) and a floor (right). Note that original objects (insets in (a), (c), (e)) do not have reflection images.

Figure 5.11: Limitation of our method. The top of the tea cups are rendered in reflection images (red oval), which is physically incorrect. This is because of the billboard approximation used in ray tracing.

Chapter 6

ROIs Extraction for Efficient Image Editing

Edit propagation enables us to easily edit images based on simple user input. Various applications of edit propagation exist, such as grayscale image colorization, color image recoloring, segmentation and tone adjustment. Many efforts have been made to attack the edit propagation problem [47, 58, 52, 91, 92, 53, 15, 94, 14, 97]. Specifying sparse image edits, users can propagate the image edits to the entire image according to the propagation principle based on pixel similarity (e.g., proximities of positions, colors or textures).

We can regard edit propagation as a sort of multi-class classification problem. From image feature vector $\mathbf{x}_i$ at pixel i, a typical edit propagation pipeline estimates multi-class probability vector $\mathbf{y}_i \in [0,1]^n$ that represents how likely it is that a pixel i belongs to each of n types of strokes (class labels). A model for estimating $\mathbf{y}_i$ is constructed from user-specified strokes and $\mathbf{x}_i$ on the strokes. A number of techniques have been proposed for edit propagation using various models, such as optimization-based methods [47, 3], gentle boost classifier [52], function interpolation with radial basis functions (RBFs) [53], manifold learning [15], and a probabilistic model [94]. Most of these previous studies use a combination of multiple features with different meanings as input. For example, they use concatenated features $\mathbf{x} = [\sigma_c \mathbf{c}^T, \sigma_t \mathbf{t}^T, \sigma_s \mathbf{s}^T]^T$ where $\mathbf{c}$ denotes pixel color, $\mathbf{t}$ texture feature, and $\mathbf{s}$ spatial coordinate whereas σ_c, σ_t, and σ_s denote parameters that determine the importance of each feature.

In most previous work, users must heuristically select the image features that they use and adjust parameters for the features in accordance with their needs and target images. As shown in Figure 6.1, the editing results can be drastically changed depending on the parameters. For example, edits on strokes with a given intensity propagate to pixels that have similar intensity when σ_c is large. Furthermore, using too many visual features might limit the range/diversity of input images on which the propagation succeeds. For example, the Gabor feature, which is typically used as a texture feature, has multiple parameters such as a kernel size and rotation angle. Using various patterns of the parameters as well

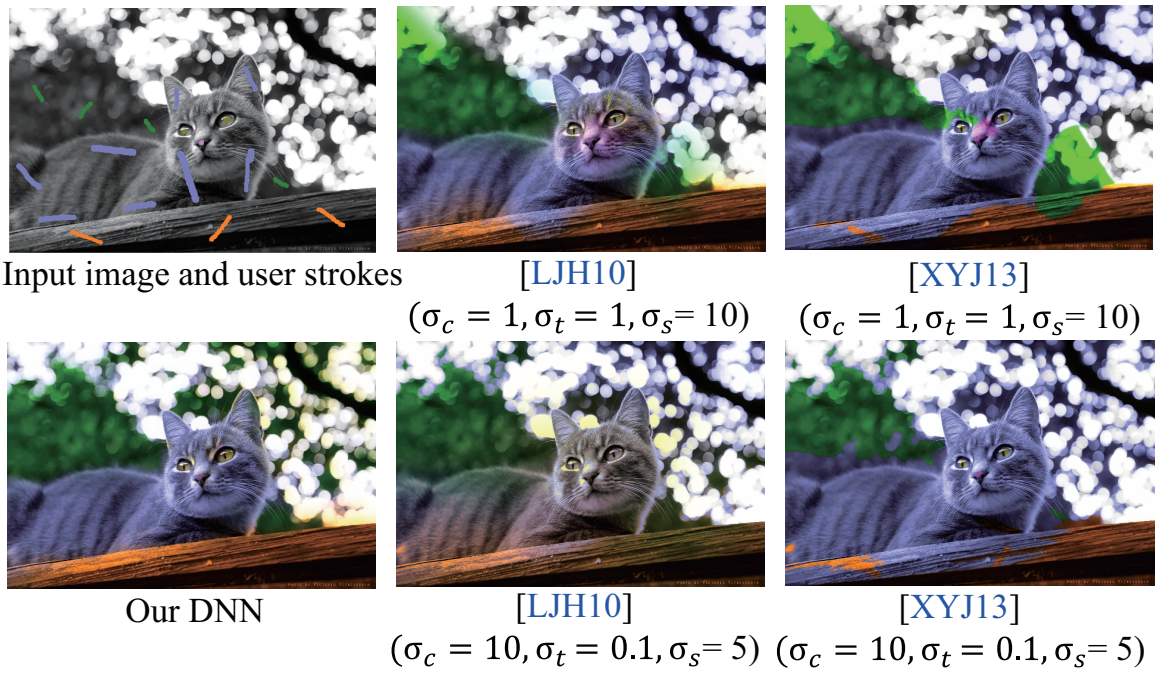


Figure 6.1: Image colorization using edit propagation. While existing methods [53] [94] (denoted as [LJH10] and [XYJ13]) require manual parameter tuning for each image feature, our DNN-based method automatically extracts stroke-adapted features.

as other image features causes an overfitting problem of the estimated model to training data on strokes due to the increased dimension of input vectors. Consequently, unneeded visual features degenerate estimated results [52]. On the other hand, manually selecting appropriate visual features from many candidates is labor intensive.

Considering the above, we conducted a study to address the following research question:

> *Can we automatically extract effective features for edit propagation without selecting image features manually?*

To this end, we adopt deep learning [7], which is a technique for learning deep neural network (DNN) models. Recently, this technique dramatically improved the accuracy of image recognition and speech recognition [43, 23]. While conventional approaches manually engineer features for classification, DNNs can be used for representation learning, which automatically extracts effective high-level features for a task from low-level features (e.g., image pixels and audio spectrum).

In this chapter, we propose *DeepProp* for extracting Deep features from a single image for edit Propagation. Our method uses low-level visual patches and spatial pixel co-

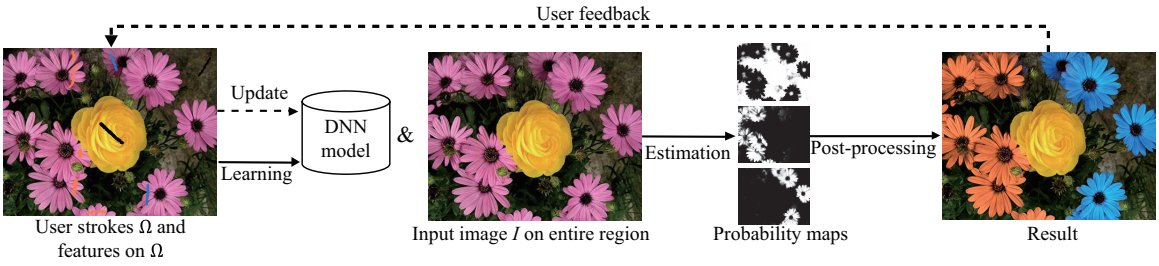


Figure 6.2: System overview. The system first learns a DNN model from an input image and user strokes. Next, stroke probabilities on all pixels are estimated using the DNN model, and probability maps are obtained. Finally, the probability maps are refined by post-processing. Every time the user updates strokes, the system updates the DNN model efficiently using previously learned parameters.

ordinates as input of a DNN that automatically extracts features adapted to user-specified strokes from a single image. In contrast to most previous work, we do not need to adjust the importance of the input features. Then, we use the DNN as a classifier that estimates user stroke probabilities, which represent how likely it is that each pixel belongs to each stroke, from extracted features on the entire image. Figure 6.1 demonstrates that our method can, without tuning parameters for image features, generate a better result than previous work.

For edit propagation with deep features, our work makes the following contributions:

- *Edit propagation system using deep learning*: We propose a system design of a framework that consists of the following three main modules (Figure 6.2): learning, estimation, and post-processing. First, a DNN model is learned using user strokes and pixels on the strokes as input. Then, using the learned DNN model, user stroke probabilities are estimated on the entire image (on pixels or, for more efficiency, superpixels), and probability maps are obtained. Finally, probability maps or application results (e.g., colorization) obtained from the probability maps are refined by post-processing. Every time the user updates strokes, the system re-learns the DNN model efficiently using previously learned parameters.

- *DNN architecture for edit propagation*: We propose a DNN architecture that extracts stroke-adapted visual features and spatial features using convolutional and fully-connected network structures. The importance of the extracted two features is also automatically determined using a feature combiner layer, and stroke probability vectors are computed from the combined features using a soft-max layer.

- *Learning algorithm for our DNN*: A number of local minima can be found during optimization of parameters of DNN models, especially when DNNs have complicated structures with deep layers and many neurons. Optimizing the entire network of our

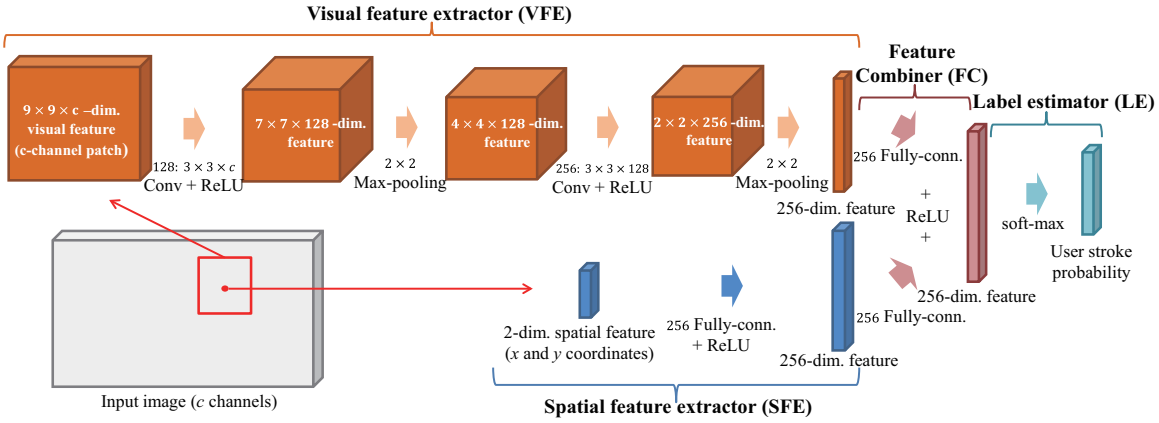


Figure 6.3: Our deep neural network architecture.

DNN tends to fall into such undesirable local minima due to the error function's vanishing gradients. This is because our DNN contains two types of the networks with different depths. We develop a learning algorithm that efficiently propagates the gradients to these networks and finds more desirable solutions, and can generate better results than a naïve learning algorithm.

6.1 DNN Architecture

The basic idea for designing a DNN is to extract high-level features for edit propagation from low-level visual and spatial features. In edit propagation, we can assume that the lowest-level features are color (or grayscale) values and coordinates. For visual features, it is known that effective high-level features can be extracted using convolutional layers. Instead of using single pixels, we use color (or grayscale) patches as low-level visual features for convolution. This is because a patch enables implicitly handling pixel gradients and capturing more various patterns than a single pixel. Additionally, we use pixel coordinates as low-level spatial features. To extract the corresponding high-level spatial features, we apply a non-linear transformation, since a convolutional layer cannot be applied in this case. Both of the extracted high-level features are then combined and used for edit propagation. We empirically experimented with several structures of the DNN, varying in, e.g., the sizes of convolutional kernels and the dimensions of high-level features.

As illustrated in Figure 6.3, our DNN is a feedforward neural network that has four structures: visual feature extractor (VFE), spatial feature extractor (SFE), feature combiner (FC), and label estimator (LE). The main notations used in this chapter are summarized in Table 6.1. The DNN estimates probability vector $\mathbf{y}_i$ of user strokes from feature vector $\mathbf{x}_i = [\mathbf{p}_i^T, \mathbf{s}_i^T]^T$ at pixel i in input image I. Here, $\mathbf{p}_i \in \mathbb{R}^{9\times9\times c}$ is a 9×9 patch feature centered

Table 6.1: Definitions of main symbols.

Symbols	Descriptions
I	input image.
$\mathbf{x}_i$	image feature vector at pixel i.
$\mathbf{g}_i$	binary vector for user strokes at pixel i.
Ω	user stroke region.
X_Ω	set of input features $\mathbf{x}_i$ in stroke region Ω.
$\mathcal{G}_\Omega$	set of binary vectors $\mathbf{g}_i$ in stroke region Ω.
$\mathbf{y}_i$	probability vector at pixel i.
Z	probability map or colorized result.
U	result obtained by post-processing.
θ_v, θ_s, θ_c, θ_l	parameters of DNN model.
G_v, G_s, G_c, G_l	layer functions of DNN model.

at pixel i with c color channels (e.g., three-channel RGB or one-channel grayscale), and $\mathbf{s}_i \in \mathbb{R}^2$ is a pixel coordinate. Each element of input feature vectors is normalized from 0 to 1. In contrast to most previous work, determining importance of each image feature is not required. In the following sections, we describe the role of each structure of our DNN.

6.1.1 Visual feature extractor (VFE)

The VFE extracts a high-level visual feature, such as complicated texture patterns or simple colors adapted to user strokes, from a low-level visual patch feature $\mathbf{p}$. Specifically, the VFE computes high-level visual features $\mathbf{f}_v \in \mathbb{R}^{256}$ from $\mathbf{p}$ using function G_v with model parameter θ_v. G_v consists of two convolutional functions f_{conv1} and f_{conv2}, a max-pooling function f_{mp}, and an activation function of the rectifier linear unit (ReLU) :

$$\mathbf{f}_v = G_v(\mathbf{p}; \theta_v), \tag{6.1}$$

$$G_v(\mathbf{p}; \theta_v) = f_{mp}(f_{ReLU}(f_{conv2}(f_{mp}(f_{ReLU}(f_{conv1}(\mathbf{p}; \theta_{conv1}))); \theta_{conv2}))), \tag{6.2}$$

where f_{conv1} applies 128 types of $3 \times 3 \times c$ filter kernels k with biases b to input patch feature $\mathbf{p}$. That is, f_{conv1} obtains 128 filtered maps $(k * \mathbf{p} + b)$ by convolving input patch features $\mathbf{p}$ with different filter kernels k and biases b. Then, the ReLU activation function $f_{ReLU}(\mathbf{v}) = [\max(0, v_1), \max(0, v_2), ..., \max(0, v_m)]^T$ is applied to the convolved patch (where the subscripts of v denote element indices of the vector). Next, we use a 2×2 max-pooling function f_{mp} with stride 2, which yields position invariance over the patches. We additionally apply convolution f_{conv2} with a $3 \times 3 \times 128$ kernel and the activation function f_{ReLU}. Finally, a 2×2 max-pooling with stride 2 is used again and a 256-dimensional

feature vector $\mathbf{f}_v$ is obtained. θ_v denotes filter parameter (i.e., filter kernels k and bias terms b) and is learned using user strokes as training data, as explained in Section 6.2.

6.1.2 Spatial feature extractor (SFE)

The SFE extracts an abstracted feature $\mathbf{f}_s \in \mathbb{R}^{256}$ from a spatial pixel coordinate $\mathbf{s}$:

$$\mathbf{f}_s = G_s(\mathbf{s}; \theta_s), \tag{6.3}$$
$$G_s(\mathbf{s}; \theta_s) = f_{ReLU}(\mathbf{W}_s \mathbf{s} + \mathbf{b}_s), \tag{6.4}$$

where $\theta_s = \{\mathbf{W}_s, \mathbf{b}_s\}$ is a model parameter of the SFE, and $\mathbf{W}_s \in \mathbb{R}^{256\times 2}$ and $\mathbf{b}_s \in \mathbb{R}^{256}$ are a weight matrix and bias term. The dimension of the output feature vectors is set to 256 so that they have the same dimension as the visual features. In the same way, the feature combiner (explained in the next section) can handle both the visual features and spatial features fairly.

6.1.3 Feature combiner (FC)

The function G_c of the FC converts $\mathbf{f}_v$ and $\mathbf{f}_s$ extracted by the VFE and the SFE into a single feature vector $\mathbf{f}_c \in \mathbb{R}^{256}$:

$$\mathbf{f}_c = G_c(\mathbf{f}_v, \mathbf{f}_s; \theta_c), \tag{6.5}$$
$$G_c(\mathbf{f}_v, \mathbf{f}_s; \theta_c) = f_{ReLU}(G_{cv}(\mathbf{f}_v; \theta_{cv}) + G_{cs}(\mathbf{f}_s; \theta_{cs})), \tag{6.6}$$
$$G_{cv}(\mathbf{f}_v; \theta_{cv}) = \mathbf{W}_{cv}\mathbf{f}_v + \mathbf{b}_{cv}, \tag{6.7}$$
$$G_{cs}(\mathbf{f}_s; \theta_{cs}) = \mathbf{W}_{cs}\mathbf{f}_s + \mathbf{b}_{cs}, \tag{6.8}$$

where $\theta_c = \{\theta_{cv}, \theta_{cs}\}$ is a model parameter of the FC (where $\theta_{cv} = \{\mathbf{W}_{cv} \in \mathbb{R}^{256\times 256}, \mathbf{b}_{cv} \in \mathbb{R}^{256}\}$ and $\theta_{cs} = \{\mathbf{W}_{cs} \in \mathbb{R}^{256\times 256}, \mathbf{b}_{cs} \in \mathbb{R}^{256}\}$). Network structures similar to the FC have also been introduced for combining multimodal features, such as image and audio [62, 89]. Such a network structure can determine the importance of two features by capturing correlations across the two modalities. We utilize this structure for combining the visual features and spatial features.

6.1.4 Label estimator (LE)

The function G_l of the LE estimates user stroke probability vectors $\mathbf{y}$ from feature vectors $\mathbf{f}_c$ extracted by the FC:

$$\mathbf{y} = G_l(\mathbf{f}_c; \theta_l), \tag{6.9}$$
$$G_l(\mathbf{f}_c; \theta_l) = f_{softmax}(\mathbf{W}_l \mathbf{f}_c + \mathbf{b}_l), \tag{6.10}$$

where $f_{softmax}(\mathbf{v}) = [\frac{\exp(v_1)}{\sum_i^n \exp(v_i)}, \frac{\exp(v_2)}{\sum_i^n \exp(v_i)}, ..., \frac{\exp(v_n)}{\sum_i^n \exp(v_i)}]^T$ is the soft-max function and $\theta_l = \{\mathbf{W}_l \in \mathbb{R}^{n \times 256}, \mathbf{b}_l \in \mathbb{R}^n\}$ is a model parameter of the LE. The LE determines, for each off-stroke pixel, a fractional weight for each stroke, based on the off-stroke pixel's similarity to pixels on the stroke. This is done using the soft-max layer, which acts as a standard regression function for probabilistic multi-class classification.

6.2 Learning DNN from User Strokes

This section explains how to learn the model parameters $(\theta_v, \theta_s, \theta_c, \theta_l)$ of our DNN using training data: user strokes $\mathcal{G}_\Omega$ and input features $\mathbf{x} \in X_\Omega$ on stroke region Ω.

A straightforward way for learning the model parameters is to minimize the error function E between probability vectors estimated by the DNN and binary vectors $\mathbf{g}_i \in \mathcal{G}_\Omega$ obtained from user strokes:

$$(\hat{\theta}_v, \hat{\theta}_s, \hat{\theta}_c, \hat{\theta}_l) = \underset{\theta_v, \theta_s, \theta_c, \theta_l}{\operatorname{argmin}} E(\theta_v, \theta_s, \theta_c, \theta_l), \tag{6.11}$$

$$E(\theta_v, \theta_s, \theta_c, \theta_l) = \sum_{i \in \Omega} L(G_l(G_c(G_v(\mathbf{p}_i; \theta_v), G_s(\mathbf{s}_i; \theta_s); \theta_c); \theta_l), \mathbf{g}_i), \tag{6.12}$$

where L indicates the cross-entropy loss function defined as $L(\mathbf{y}, \mathbf{g}) = -\sum_{k=1}^n \{g_k \ln y_k + (1 - g_k) \ln (1 - y_k)\}$.

Unfortunately, this naïve approach does not work in practice. Because of the structural complexity of our DNN, we need to consider the fact that many local minima can be found in an optimization process such as the one described above. In general, model parameters of feedforward neural networks are learned using the backpropagation of error with a gradient-based optimization algorithm, such as the stochastic gradient descent (SGD) [41]. However, in our case, when the model parameters of the entire network are optimized simultaneously using the backpropagation, the algorithm falls into local minima before the gradient error propagates sufficiently into the deep layers of the VFE, due to the vanishing gradient problem. That is, the learning speed of the SFE is much faster than that of the VFE because the SFE is shallow whereas the VFE is much deeper.

To address this issue, we develop a learning algorithm for optimizing the model parameters efficiently in two steps. Figure 6.4 illustrates this learning strategy. In the first step, we omit the SFE from the DNN and optimize the deep network consisting of the VFE, FC, and LE. Specifically, we pre-train visual features by minimizing a new error function E_v:

$$(\hat{\theta}_v, \hat{\theta}_{cv}, \hat{\theta}_l) = \underset{\theta_v, \theta_{cv}, \theta_l}{\operatorname{argmin}} E_v(\theta_v, \theta_{cv}, \theta_l), \tag{6.13}$$

$$E_v(\theta_v, \theta_{cv}, \theta_l) = \sum_{i \in \Omega} L_v(G_l(G_{cv}(G_v(\mathbf{p}_i; \theta_v); \theta_{cv}); \theta_l), \mathbf{g}_i), \tag{6.14}$$

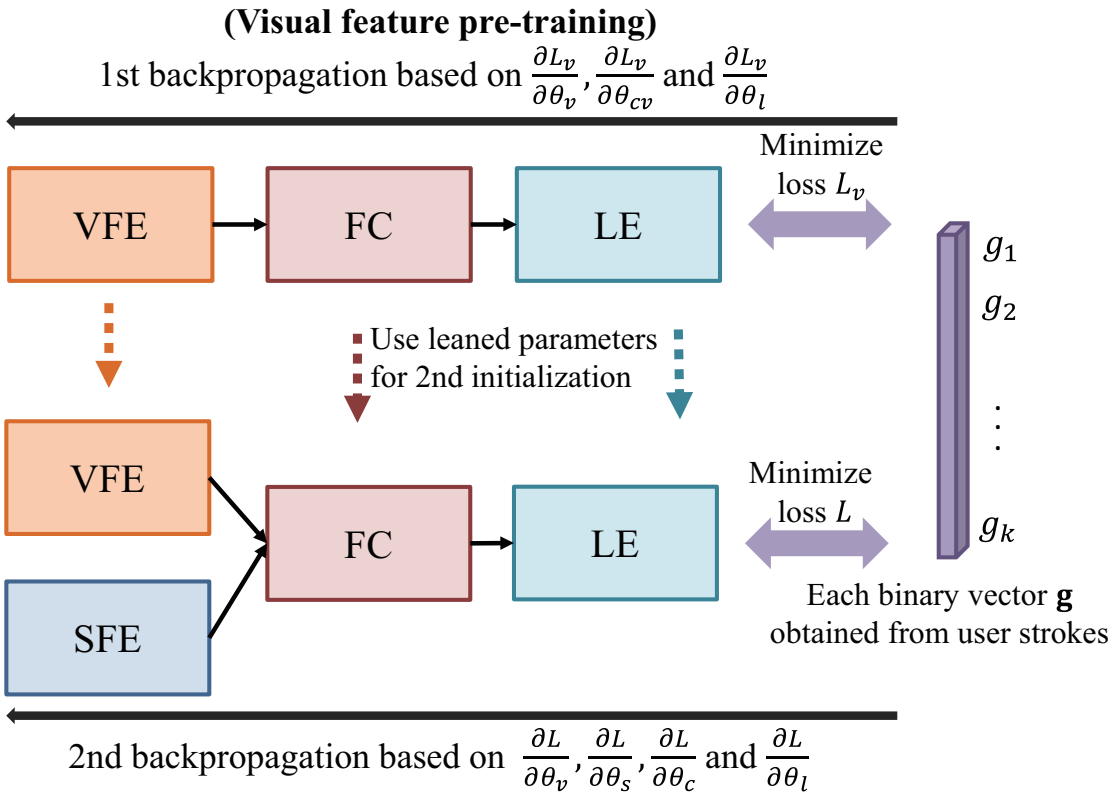


Figure 6.4: Our strategy of DNN learning. We first pre-train visual features on the network consisting of the VFE, FC, and LE using backpropagation, and then learn the entire network together with the SFE.

where L_v is the cross-entropy loss of the above network. We optimize the parameters based on gradients $\frac{\partial L_v}{\partial \theta_v}$, $\frac{\partial L_v}{\partial \theta_{cv}}$, $\frac{\partial L_v}{\partial \theta_l}$ using the backpropagation. We initialize the model parameters with random numbers following a normal distribution with zero mean and standard deviation equal to the inverse of the dimension of each model parameter. For optimization, we use the mini-batch Adam algorithm [41] because of its fast convergence. To determine convergence, we check if the error in all training data is smaller than $\epsilon|\Omega|$ or the difference between the current error and previous error is larger than $\gamma|\Omega|$ (where ϵ and γ are coefficients, and $|\Omega|$ is the number of pixels in region Ω). Note that the latter condition is used to abort optimization if it does not converge completely. In the second step, we optimize the entire network with the SFE by initializing the DNN with the model parameters $\hat{\theta}_v$, $\hat{\theta}_{cv}$, and $\hat{\theta}_l$ learned in the first step. The model parameters θ_s and θ_{cs} not learned in the first step are randomly initialized and optimized using the mini-batch Adam algorithm in the same way as above. For optimization, mini-batch size is empirically set to 10, and both ϵ and γ are set to 0.01.

As can be seen in Equations (6.12) and (6.14), the computational complexity of the

optimization w.r.t. the number of training data $|\Omega|$ is $O(|\Omega|)$. To accelerate the learning, we subsample pixels Ω' by 10% of Ω for optimization. Even with subsampling the learned result is similar to the result with all data due to inherent redundancy, as demonstrated in Section 6.4.1. Algorithm 2 summarizes our DNN learning algorithm.

Algorithm 2 Learning DNN from user strokes

Inputs: stroke region Ω, image feature vectors $\mathbf{x}_i = \{\mathbf{p}_i, \mathbf{s}_i\} \in X_\Omega$, and stroke binary vectors $\mathbf{g}_i \in \mathcal{G}_\Omega$

Outputs: $\theta_v, \theta_s, \theta_c, \theta_l$

1: $\Omega' \leftarrow$ RANDOMSAMPLING(Ω);
2: $\theta_v, \theta_s, \theta_c, \theta_l \leftarrow$ NORMALDISTRIBUTION();
3: $preE_v \leftarrow 0$;
4: **while** true **do**
5: $\quad \theta_v, \theta_{cv}, \theta_l \leftarrow$MINIBATCHADAM$(X_{\Omega'}, \mathcal{G}_{\Omega'}, \frac{\partial L_v}{\partial \theta_v}, \frac{\partial L_v}{\partial \theta_{cv}}, \frac{\partial L_v}{\partial \theta_l})$;
$\quad$ // Eq. (6.14)
6: $\quad$ **if** $E_v < \epsilon|\Omega'|$ or $E_v - preE_v > \gamma|\Omega'|$ **then**
7: $\quad\quad$ break;
8: $\quad$ **end if**
$\quad preE_v \leftarrow E_v$;
9: **end while**
10: $preE \leftarrow 0$;
11: **while** true **do**
12: $\quad \theta_v, \theta_s, \theta_c, \theta_l \leftarrow$MINIBATCHADAM$(X_{\Omega'}, \mathcal{G}_{\Omega'}, \frac{\partial L}{\partial \theta_v}, \frac{\partial L}{\partial \theta_s}, \frac{\partial L}{\partial \theta_c}, \frac{\partial L}{\partial \theta_l})$;
$\quad$ // Eq. (6.12)
13: $\quad$ **if** $E < \epsilon|\Omega'|$ or $E - preE > \gamma|\Omega'|$ **then**
14: $\quad\quad$ break;
15: $\quad$ **end if**
$\quad preE \leftarrow E$;
16: **end while**

6.2.1 Efficient DNN update per user edit

Every time the user adds or removes a stroke in order to obtain a desired result, the system has to re-learn the model parameters from the update strokes. Naïvely re-learning from scratch is costly, and thus accelerating the re-learning is crucial for interactive editing.

To update the model parameters efficiently, we reuse the model parameters learned with the previous user strokes as the initialization of the learning algorithm (line 4 in Algorithm 1). To achieve this, we need to adapt the LE structure accordingly. As shown in Figure 6.5, if the total number of different types of user strokes (e.g., strokes that edit color for colorization) is reduced, corresponding rows of the model parameters (i.e., the weight

matrix $\mathbf{W}_l$ and bias term $\mathbf{b}_l$) are deleted. If a new type of user stroke is introduced, we add new rows and initialize them with random numbers following the normal distribution.

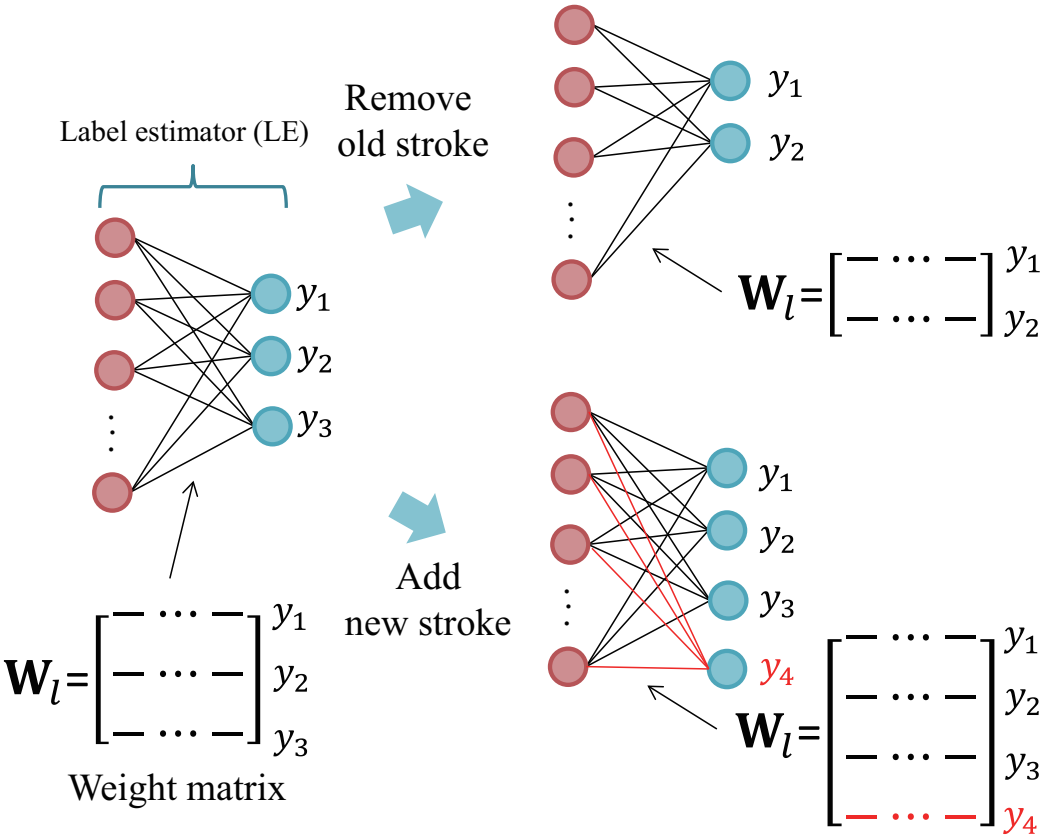


Figure 6.5: Efficient model parameter update. If user strokes are added or removed, model parameters of the LE are updated efficiently using previous parameters.

6.3 Edit Propagation Using DNN

6.3.1 Estimating probability maps

We now propagate information from the user strokes to all pixels in the input image using the learned DNN model. Given a patch feature and coordinate feature of each pixel, our feedforward neural network outputs a stroke probability vector using the learned parameters. However, processing all pixels one-by-one takes a large amount of time since the feedforward computation for one data point includes many applications of convolution filtering.

We adopt superpixel-wise propagation to reduce the computational time. For each superpixel, a center pixel is calculated, and then a user stroke probability is computed using the DNN model from features calculated only at the center pixel. Finally, the same stroke probability is assigned to all the pixels in the corresponding superpixel. For generating superpixels, we use the simple linear iterative clustering (SLIC) [2] because this algorithm can generate regularly-shaped superpixels that can reduce the gap between shapes of superpixels and patches.

6.3.2 Post-processing

We refine the estimated result by post-processing, which has mainly two roles, i.e., smoothing and interpolation. First, we can improve the result by smoothing it across superpixels. Second, interpolation can alleviate noise and halo artifacts along object boundaries [15, 94]. Similarly to [94], we obtain a final editing result (e.g., colorized or segmented images) U from the result Z estimated with the DNN model, by solving the following optimization problem:

$$\min_{\mathbf{u}_i \in U} \sum_i \beta_i ||\mathbf{u}_i - \mathbf{z}_i||^2 + \lambda \sum_i \sum_{j \in N(i)} \phi_{ij} ||\mathbf{u}_i - \mathbf{u}_j||^2, \tag{6.15}$$

$$\beta_i = \ominus\{e_i = 0\}, \tag{6.16}$$

where $\mathbf{z}_i \in Z$ denotes the quantity being post-processed, e.g., the stroke probability vector $\mathbf{y}_i$ (in case the probability map itself is the target of the post-processing) or the result computed from $\mathbf{y}_i$ in a specific application (e.g., color in a colorization application). e_i is binary edge at pixel i in input image I, and $\ominus$ is the morphological erosion operator. λ determines the degree of smoothing, and $N(i)$ is a set of neighbor pixels of pixel i. We can use several types of weighting coefficients ϕ_{ij} that represent relations between neighbor pixels, such as the affinity function [47] based on similarity of pixel values based on similarity of pixel values (when considering pixel gradient information) and matting Laplacian [48] for matting. In our experiment, λ is set to 5 for all images. To solve this linear system, we use the Gauss-Seidel method because of its simplicity of implementation. Finally, Algorithm 3 summarizes our edit propagation using the DNN model.

6.4 Experiments

We implemented our prototype system with C++ and the OpenCV2.4.9 library except for deep learning. For implementing deep learning, we used Python and the chainer library. The system was run on a PC equipped with a 2.80 GHz CPU and 8 GB of memory. The image sizes we used in our experiments, the number of user strokes (total pixels), and computational times of our method are summarized in Table 6.2. As can be seen in the table, the

Algorithm 3 Edit propagation using DNN

Inputs: input image I, image feature vectors $\mathbf{x}_i = \{\mathbf{p}_i, \mathbf{s}_i\}$, learned DNN parameters $\theta_v, \theta_s, \theta_c$, and θ_l

Output: final editing result U

1: $\Omega_s \leftarrow$ EXTRACTSUPERPIXEL(I);
2: **for** each superpixel $S \in \Omega_s$ **do**
3: $k \leftarrow$ EXTRACTCENTERPIXEL(S);
4: $\mathbf{y}_k \leftarrow G_l(G_c(G_v(\mathbf{p}_k; \theta_v), G_s(\mathbf{s}_k; \theta_s); \theta_c); \theta_l)$;
5: $Z \leftarrow$ ASSIGNRESULTTOSUPERPIXEL$(S, \mathbf{y}_k)$;
6: **end for**
7: $U \leftarrow$ POSTPROCESSUSINGOPTIMIZATION(I, Z);

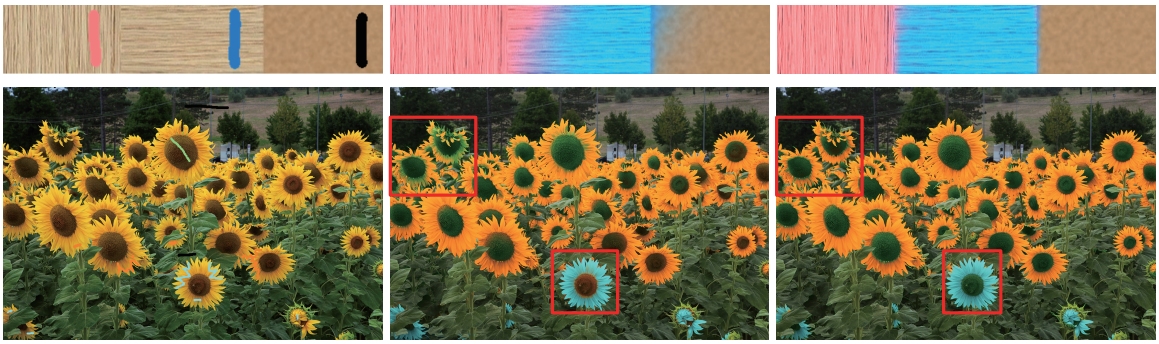


Figure 6.6: Comparisons of recolorization results without and with visual feature pre-training.

results of Figure 6.1 and the lower center of Figure 6.10 took longer than most of the others. This is because the convolutional layers require substantial learning to extract effective visual features such as textures from little available information (i.e., 1-channel grayscale values). We also tested with the high-resolution image in Figure 6.6 in our experiments. Although it took the longest, the time needed for that image is relatively short given that the number of pixels is tens of times larger than the other images. This is because only pixels on strokes are used for learning and superpixels are used for estimation.

6.4.1 Evaluation of proposed method

Before comparing our method to previous work, we evaluate our method itself.

Learning algorithm. We first evaluate effectiveness of the proposed learning algorithm for our DNN. Figure 6.6 shows comparisons of image recoloring results with and without the visual feature pre-training. Without this, the visual features (texture patterns or color)

Table 6.2: Total amount of inputs and computational times (seconds) of our method on CPU.

Images	Image sizes	# of user strokes (# of total pixels)	Times
Fig.6.1	620×413	20 (7701)	97.4
Fig.6.2	400×314	10 (1818)	23.0
Fig.6.6 top	392×70	3 (1882)	18.3
Fig.6.6 bottom	1800×1200	19 (17964)	432.4
Fig.6.10 top	480×360	4 (2457)	22.4
Fig.6.10 upper center	640×480	5 (3208)	36.5
Fig.6.10 lower center	500×375	13 (20342)	271.6
Fig.6.10 bottom	640×427	9 (3285)	40.0
Fig.6.12 top	400×285	4 (5353)	36.1
Fig.6.12 center	400×267	8 (9666)	49.9
Fig.6.12 bottom	400×280	3 (6940)	39.3

were not sufficiently learned from the image, and the image edits propagated too strongly, only relying on the spatial features. This means that the optimization process fell into the local minimum before the gradient of the error function sufficiently propagated to the network of the VFE. Specifically, in the upper-center image in the figure, the specified colors (pink and blue) are leaking across the different textures because texture features are not extracted sufficiently. Also, in the red box of the lower-center image, the color specified with the green stroke did not propagated according to the texture patterns of the center of the sunflower, due to the same reason. By contrast, we can see that using our learning algorithm with visual feature pre-training alleviates these problems in the right images.

Updating algorithm. We evaluate the effectiveness of the algorithm for updating the model parameters in Figure 6.7, where we compare the editing results and computational times of learning without and with update. When we do not use the updating algorithm, model parameters are learned from scratch every time strokes are added or removed. As can be seen in the figure, there is almost no difference between the two results, whereas the computational times with update become about half of the others.

Training data size. Figure 6.8 shows the editing results and computational times of learning with different amounts of training data. As shown in the figure, the results almost converged when we used equal to or more than 10% of the training data on the strokes, and computational times were linearly reduced according to the amount of the training data. Although it is generally known that deep learning requires a large amount of training data, in fact, a few hundreds of instances per one category are sufficient for learning models in some image recognition tasks [87]. Given this fact and the experimental results, we can confirm that our method works well with relatively little training data, and only a few types

of user strokes.

Superpixel-based estimation. Figure 6.9 shows the editing results and computational times of estimating with different numbers of superpixels. In this example, the results are visually indistinguishable even when the number of superpixels decreases to 1% of the total image pixels. However, the computational times are significantly reduced according to the number of superpixels. Only when the number of superpixels is 0.1% of the total image pixels, the result starts deteriorating. This is because the superpixels become too large to fit the object shape. On the other hand, computational times were almost unchanged when less than 1% were used: of the total reported time, 4 seconds are due to the post-processing, which can be accelerated by using more efficient solvers.

6.4.2 Comparison with previous methods

To verify the effectiveness of our feature extraction method, we compare our method with the handcrafted feature-based methods.

Compared features

We prepared the following image features used in previous work.

- **color feature**: three (or one)-dimensional RGB (or grayscale) vectors $\mathbf{c}$.
- **spatial feature**: two-dimensional pixel coordinate vectors $\mathbf{s}$.
- **patch feature**: 243 (or 81)-dimensional vectors $\mathbf{p}$ of 9×9 RGB (or grayscale) patches.
- **texture feature**: 40-dimensional vectors $\mathbf{t}$ obtained by applying the Gabor filter [21, 14].
- **dense SIFT feature**: 128-dimensional vectors $\mathbf{d}$ that are scale-invariant feature transform (SIFT) features [57] on each pixel.

For the texture features, we selected the parameters of the Gabor filter, i.e., kernel size, wavelength of the sinusoidal factor, spatial aspect ratio, phase offset, standard deviation of the gaussian envelope, and kernel orientation as 30×30, π, 1, $\pi/2$, $\{1, 2, 3, 4, 5\}$, and $\{0, \pi/8, 2\pi/8, 3\pi/8, 4\pi/8, 5\pi/8, 6\pi/8, 7\pi/8\}$, respectively, and consequently 40-dimensional vectors were obtained. We concatenated these features and used $\mathbf{x} = [\sigma_c\mathbf{c}^T, \sigma_s\mathbf{s}^T, \sigma_p\mathbf{p}^T, \sigma_t\mathbf{t}^T, \sigma_d\mathbf{d}^T]^T$ in the comparisons with previous methods. The experiments were conducted by adjusting the importance of each feature $\sigma_c, \sigma_s, \sigma_p, \sigma_t$, and σ_d. As mentioned above, the proposed method uses $\mathbf{x} = [\mathbf{p}^T, \mathbf{s}^T]^T$ as input feature and we do not manually adjust the importance of them.

Results

The state-of-the-art for edit propagation is the sparse control model (SCM) [94] except for methods that focus on acceleration and memory efficiency. Given that comparisons with other edit propagation methods have been conducted in the SCM paper [94], and the fact that we focused mainly on feature extraction, we only compare our method to SCM (denoted as [XYJ13] in the following figures) and instant propagation (IP) [53] (denoted as [LJH10] in the following figures), which is closely related to SCM, using our own implementations.

Colorization and recoloring. Figure 6.10 shows comparisons of image recoloring and colorization between our method (DNN) and the previous methods with different parameters. The color of each pixel is computed as the probability-weighted average of the ab channels of each stroke in Lab color space using the probability maps obtained by our DNN. The colorized results are then post-processed. The parameters are shown at the bottom of the figure and correspond to results in each column. First, the result of our DNN obviously outperformed the leftmost results, which were obtained via the previous methods but using the same features as ours, i.e., patch and spatial features. Second, the use of all features in the previous methods tends to deteriorate the estimated results due to the over-fitting. Third, it is difficult to handle some images even if individual features are selectively used. In contrast to these results, our method can generate better results than the previous methods overall. This means that, from low-level patch and coordinate features, our method extracted stroke-adapted high-level features that are effective for edit propagation, without manual feature selection.

Foreground segmentation. We compared results of foreground segmentation using 50 images randomly selected from MSRA 1k dataset [1]. We quantitatively evaluate the methods via a precision-recall (PR) curve, which is often used for saliency detection [85]. While precision indicates the ratio of ground-truth pixels among pixels estimated as foreground, recall indicates the ratio of pixels estimated as foreground among all the ground-truth pixels. To extract foreground regions, we segmented probability maps using a threshold. PR curves were plotted with precision and recall obtained using different thresholds. We specified foreground and background manually using strokes on each image and used the same strokes for all of the methods with the fixed feature parameters that were used in several of the results of [53], i.e., $\sigma_c = 1/0.2 = 5$ and $\sigma_s = 1$. As summarized in the PR curve in Figure 6.11 and Table 6.3, our method overall outperforms the existing methods. Figure 6.12 also shows some results that are difficult for the existing methods to segment with only color information, that is, when the foreground and background colors are very similar. Although we tried to increase the importance of the texture features in the existing methods, over-fitting problems occurred and results degenerated in some cases. While the results of the existing methods might be improved if the feature parameters are tuned intensively, our method can generate relatively good results without this process.

As shown in Figures 6.1 and 6.10, the two existing methods propagate image edits based

Table 6.3: Macro average precision, recall, and F1 $\pm$ standard deviation for binarization of probability maps using a threshold (50%) on the 50 images. Results marked with '*' show statistically significant differences between our method and the others as measured by paired t-test.

Method	Precision	Recall	F1
[53]	0.905 ± 0.113	0.900 ± 0.168	0.885 ± 0.145
[94]	0.900 ± 0.131	0.892 ± 0.115	0.887 ± 0.099
Our DNN	0.945 ± 0.060*	0.940 ± 0.065	0.940 ± 0.048**

$* : p < 0.05$, $** : p < 0.01$

on different points of view. That is, IP estimates stroke probabilities smoothly, whereas SCM estimates them discriminatively. As explained in [94], SCM is based on iterative feature discrimination and associates each pixel with only a part of the control samples. For example, SCM can clearly propagate different image edits to neighbor objects that have similar color by discriminating the spatial features, even if the importance of these features is somewhat small. However, it is difficult to take advantage of this method unless appropriate features are selected and the importance of them is appropriately determined. Our focus is to address this problem, which was demonstrated in this section. It is an interesting avenue for future work to integrate deep features into the existing edit propagation methods.

6.4.3 User study

We also conducted a user study to validate whether users can generate plausible results using our system in as few attempts as possible. We recruited eight users, six novice and two experienced users in image processing. The four images used in this study were Figures 6.1, 6.2, and 6.10 top and upper center. For each image, we showed sample results to the users and asked them to conduct the following tasks using our system and SCM [94], allowing them up to three trials until they are satisfied:

- Task 1 (Fig. 6.2): recolor groups of flower petals at different locations with the same colors.
- Task 2 (Fig. 6.10 upper center): recolor each type of cookies.
- Task 3 (Fig. 6.10 top): recolor only the peels of the pear.
- Task 4 (Fig. 6.1): colorize each object such as leaves, woods, and a cat.

Before conducting these tasks, we briefly explained how to use the systems and the image features to the users, and they practiced for about five minutes using other images such as

Figure 6.6 top. Because the users might get used to editing the same image from the second time, we divided the users into two groups and shuffled the orders of images and methods for each group. For SCM, we limited the appropriate ranges of the feature weights from zero to five in order to avoid eccentric feature weights. The number of trials were recorded, and the resulting images were evaluated by another 11 evaluators using a subjective score ranging from one to five, as summarized in Figure 6.13. Thanks to the tuning-free characteristic of our system, the number of trials for the editing was typically around two, which was less than that of SCM. Additionally, the quality of resultant images using our system consistently outperforms that of the existing method.

6.5 Discussion and Future Work

In the context of the proposed method, it is possible to consider two other approaches: (1) data-driven pre-training and (2) network expansion.

As for (1), in several applications for computer vision, DNNs are pre-trained using datasets containing a large number of general images such as ImageNet [22] before training on data for specific tasks [51, 56]. This improves the generalization ability of models and accuracy on the tasks. In contrast, our method does not use such data-driven approaches. Nevertheless, it successfully generates several image editing results. It is not clear that data-driven pre-training would improve the task handled in our task because learned features strongly depend on the variously defined user strokes. Additionally, applying such pre-trained models to our task is not straightforward because existing pre-trained models are designed for image-level classification, but our task is pixel-level classification. For example, Network in Network (NIN) [54] and AlexNet [43] require a relatively-large image as input, that is, an image of 227×227 pixels in order to classify the image as "scenery", "objects", or "animals", etc. On the other hand, our task needs to classify individual pixels in a single image, and thus we used relatively-small 9×9 image patches as input. If we were to disregard this essential difference and still integrate a pre-trained model into our framework, a naïve way would be to simply replace our VFE with a pre-trained model. We conducted such an experiment with NIN, which is a lightweight pre-trained model using ImageNet and whose performance is sometimes slightly better than AlexNet in image classification tasks. Specifically, we used NIN without its classification layer, which outputs 1000-dimensional feature vectors from input images, and fixed its model parameters during a learning session in order to leverage pre-trained features. Because NIN (and also AlexNet) requires images of 227×227 pixels, we magnified 9×9 input patches to 227×227 patches and fed these large patches to NIN. As clearly shown in Figure 6.14, the NIN model cannot appropriately capture visual features. Even worse, this approach took a significantly longer time (five minutes on average) despite the fact that the NIN model is lightweight one, as this network has a too deep structure for our task. Because such data-driven approaches are out of the scope of our research question, we consider such problems

as future work.

As for (2), it is interesting to use more wide-scale networks, and we tried to use DNNs with deeper layers and larger kernels. However, there were no significant changes in results while the computational time increased when we used images with up to 2M pixels in our experiments. If we use larger images, a network that automatically changes its scale according to image sizes could possibly improve editing results.

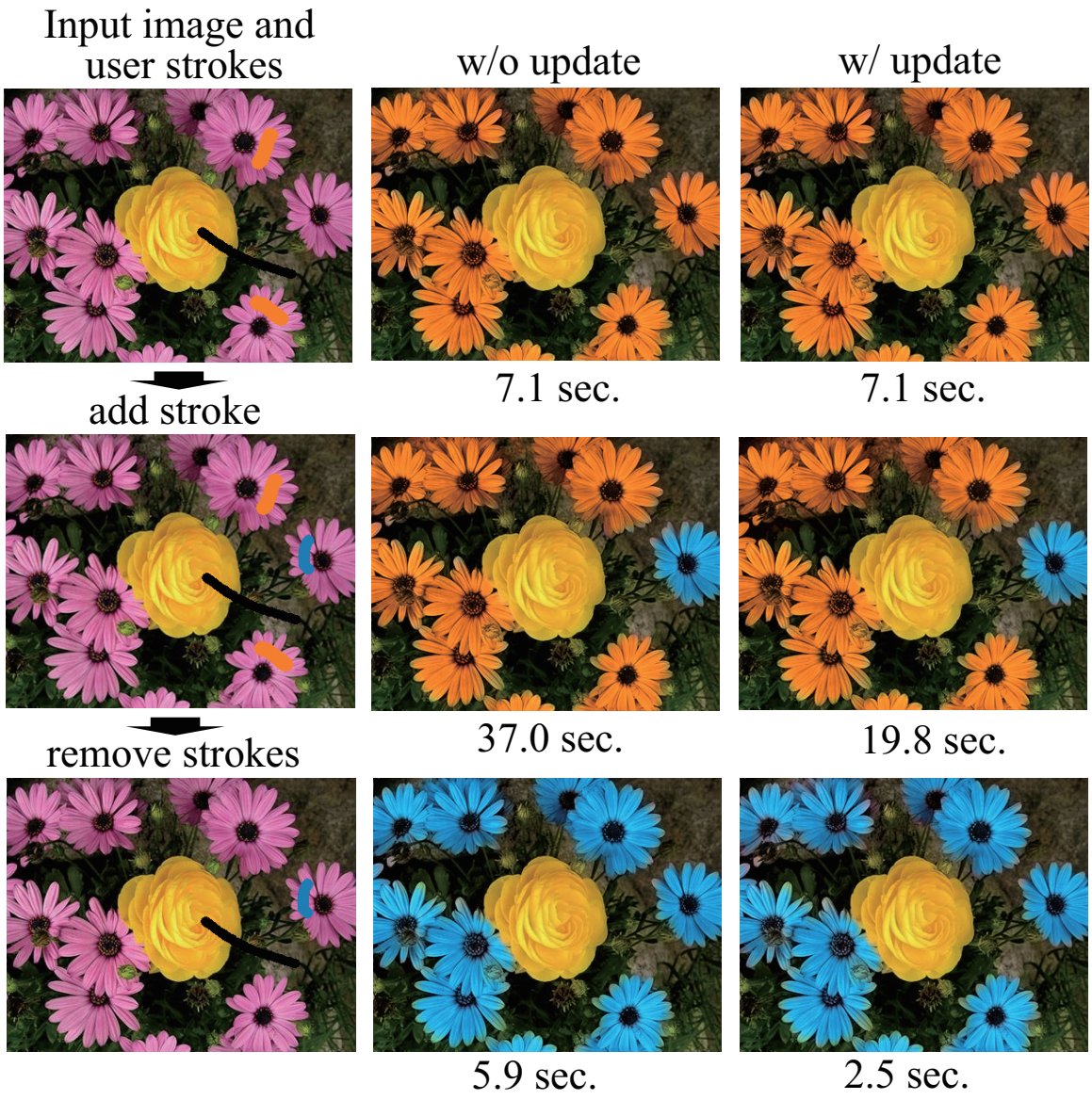


Figure 6.7: Comparison of colorization results without (w/o) and with (w/) updating parameters. The caption under each result shows computational time of learning.

Figure 6.8: Results with training data of different ratio. The input image and user strokes are the same as Figure 6.1. The caption under each image shows the percentage of samples of training data (left) and computational time of learning (right).

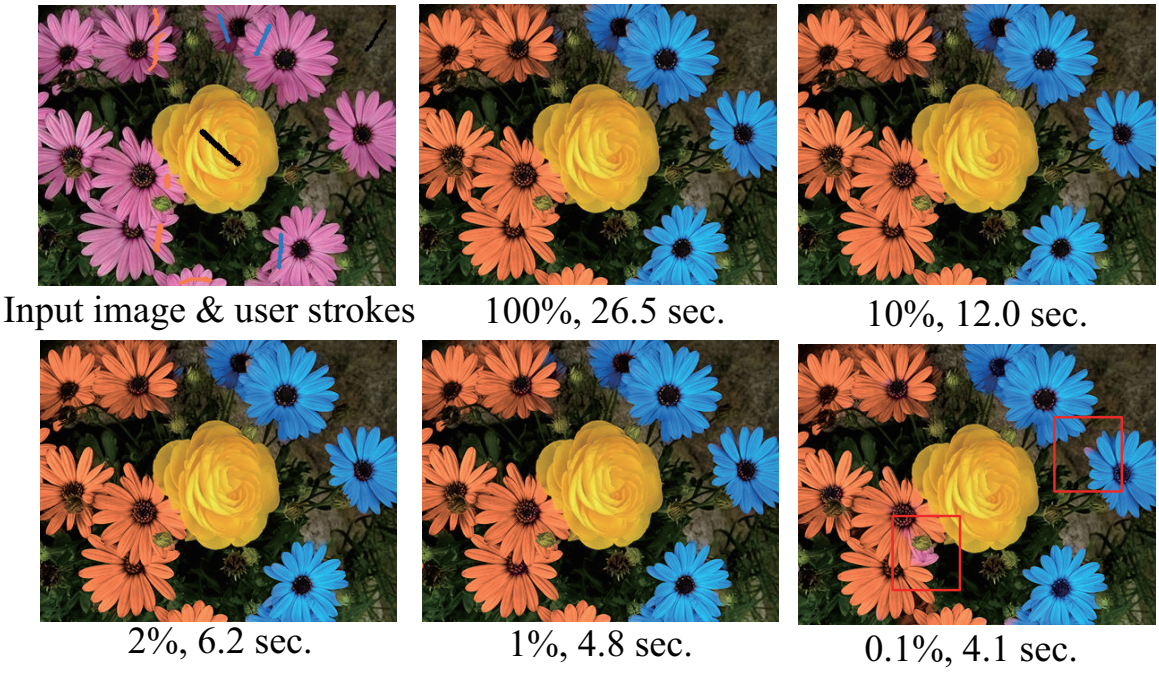


Figure 6.9: Results with different number of superpixels. The caption under each image shows the percentage of the number of superpixels to that of the original image pixels (left) and computational time of estimation and post-processing (right).

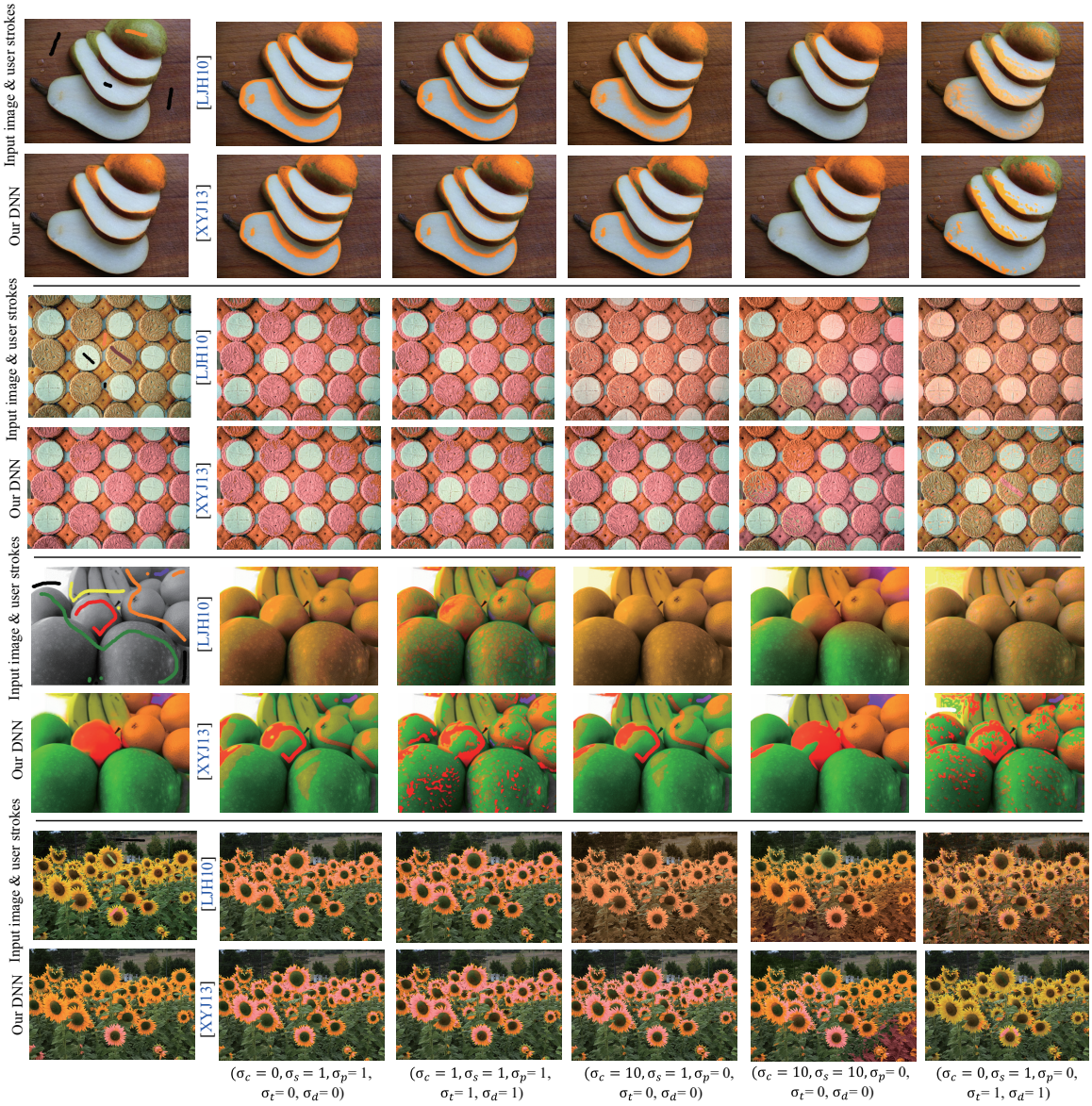
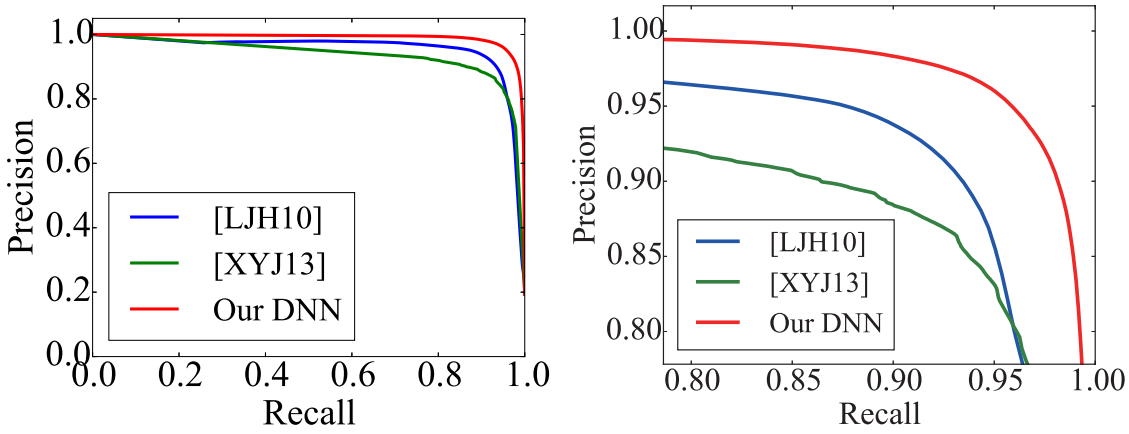


Figure 6.10: Comparisons of color image recoloring and grayscale image colorization. For the existing methods [53] [94] (denoted as [LJH10] and [XYJ13], the feature parameters used in each column are shown in the bottom.



(a) (b)

Figure 6.11: (a) Micro average PR curve that shows comparisons of foreground segmentation using 50 images randomly selected from MSRA 1k dataset [1]. (b) Magnified PR curve.

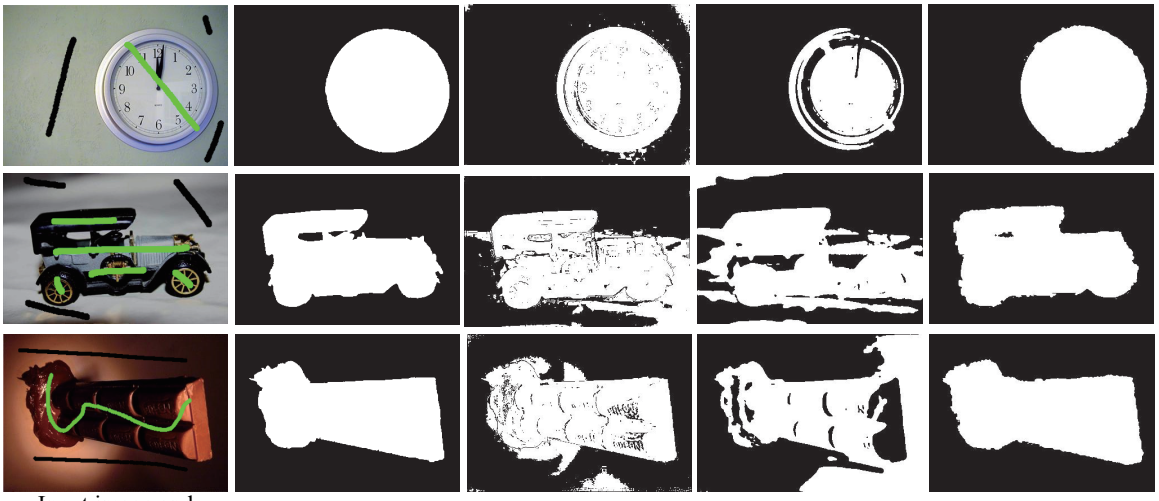


Figure 6.12: Comparisons of several results of foreground segmentation selected from MSRA 1k dataset [1]. Each segmented result is visualized by binarizing a probability map using a threshold (50%).

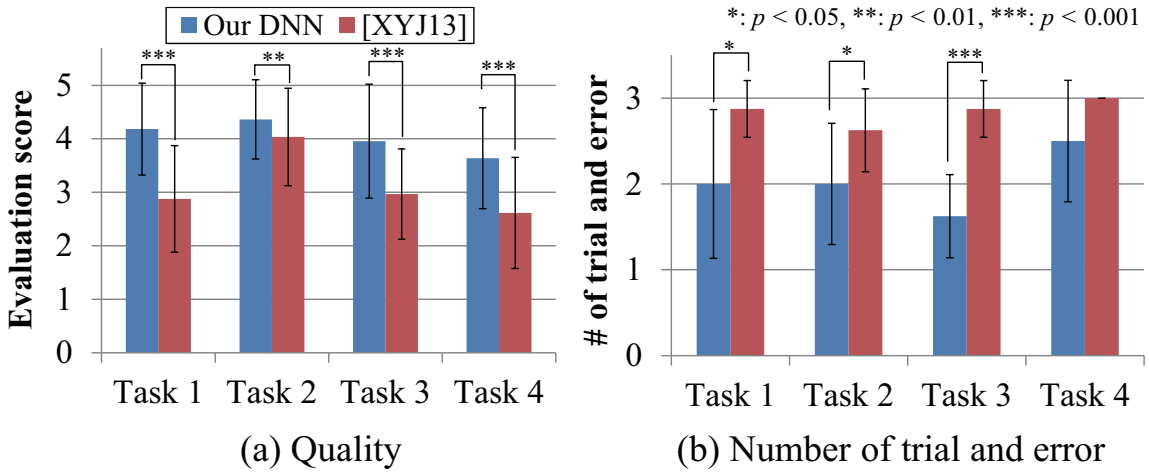


Figure 6.13: Results of user study. Error bars show the standard deviation, and results marked with '*' show statistically significant differences as measured by paired t-test.

Figure 6.14: Results with NIN model instead of our VFE. The same input image and user strokes are used for each image in Figures 6.1, 6.2, 6.10 top and upper center.

Chapter 7

Conclusion and Future work

In this thesis, we focused on the goal of establishing efficient framework that can reproduce realistic real-world appearance variations on 2D images. To achieve this, we proposed weathering and reflection models for images. Our methods enabled users to more easily generate realistic phenomena on images than not only existing methods and but also general image editing softwares. As a pre-processing in our framework, we also provided an efficient method of edit propagation for the ROI extraction. Additionally, we applied our edit propagation method to other image editing such as colorization. We verified effectiveness of our methods in the extensive experiments including the various applications and user studies.

7.1 Summary of Contributions

We summarize the contributions of this thesis as follows:

- **Reproduction of weathering effects:** We have proposed an interactive system that helps users design water flow stains on outdoor images. Based on a particle simulation modified from Dorsey et al.'s model, our system allows the user to specify the initial and terminal positions of particles by drawing a few control lines. Additionally, the user can adjust the simulation to the perspective in the input image by using a control mesh. Regarding the simulation scheme, we reduced the parameters used in Dorsey et al.'s model to improve the usability, and ignored the interaction between particles to accelerate simulations. Our system automatically estimates the bumpiness of the surface where particle flows, using the luminance variations in the input image. A user test demonstrated that our system yields better results more quickly than a generic paint tool in the task of synthesizing water flow stains.

  Additionally, we have presented a technique for reproducing weathering effects taking into account the geometric details caused by weathering. Because our method

can reproduce geometric variations caused by weathering effects by assuming that they are high-frequency patterns, our method cannot handle weathering effects not extracted as high-frequency patterns. Nevertheless, we have demonstrated that our approach can achieve more realistic results through various results. Moreover, we have introduced a user interface for editing weathering effects with a brush tool. Using the brush tool, the user could edit weathering effects while considering where weathering tends to proceed.

- **Reproduction of surface reflection:** We have presented a method for matting and compositing reflection in images. Our system allows the user to easily edit reflection through only a few user interactions. Based on our assumptions, our matting algorithm solves the reflection matting problem that is difficult to handle with existing image matting techniques. Despite the simplification by our assumptions, we have demonstrated results with visually-plausible composited reflection.

- **ROIs extraction for efficient image editing:** We have proposed DeepProp, which achieves various image edits from only simple user strokes using deep leaning. As for our research question, we conclude that, without manual feature selection, effective features for edit propagation can be automatically extracted by (i) deep learning from sparse user inputs in a single image and (ii) efficiently learning the DNN in order to avoid falling into undesirable local solutions due to the vanishing gradient problem. Our edit propagation system using deep features has generated better results than previous work in several applications such as grayscale image colorization, image recoloring, and foreground segmentation.

7.2 Future Work

We state our future work for each chapter and our long-term view.

- **Reproduction of weathering effects:** Future work for Chapter 3 is to shorten the computational time for simulation. If input images are large, the simulation parameter of the particle size should be increased. This fact increases the computational time and reduce the usability of the system. We believe that parallel computation on GPU is effective. Another future work is to improve our simulation scheme, which simulates the particle movements using the displacement map based on the variations of intensity between neighbor pixels. Luminance artifacts (e.g., specular highlights, shadows) in the input image might affect the simulation, which will be avoided by removing such artifacts [32] or by manually editing the displacement map.

  As we described in Chapter 4, in the deweathering process, our approach cannot restore object shapes if they are inherently complicated. For future work, we would like to develop a better deweathering technique that can restore the shapes of objects.

The data-driven approach that utilizes other images containing unweathered objects may be effective for solving this problem.

- **Reproduction of surface reflection:** In Chapter 5, our work is the first attempt of matting for complicated reflection, and we believe it has opened a new avenue of research topics; handling of more complicated (e.g., curved) reflection surfaces, geometries of real objects, and separation of a transmission component with surfaces of shallow water such as a pond, river or pool, where complicated patterns of bottoms will be seen.

- **Efficient ROIs extraction:** As for future work based on Chapter 6, the data-driven approaches may be effective for learning DNNs as we mentioned before. Besides, more acceleration is needed for interactive editing. Most of the modules in our system can be accelerated using parallel computation on GPU. To this end, the GPU-based SLIC algorithm for generating superpixels was proposed [70], linear solvers for the post-processing was also accelerated [39], and learning and estimation algorithms for DNNs were parallelized [46]. The chainer library we used makes GPU implementation easy, and thus we tried to use GPU acceleration for deep learning with our method. The algorithm on PC with NVIDIA GeForce GTX 760 was about five times faster than CPU implementation. For high resolution images, our system may provide instant feedback with a low resolution result as a preview during editing sessions. We would like to accelerate our system for more interactive editing in the future.

Finally, we envision our long-term goal. The proposed methods in this thesis are mainly based on two approaches, that is, theoretical approach and data-driven approach. Although the former approach makes steady progress, there are still many challenges for various appearance variations. The latter approach holds the promise of applications to wide range of problems but it requires enormous data. Recently, we can easily obtain a large amount image and video data through the Internet. Additionally, using crowd sourcing, we may be able to collect data of image editing operations, and can more deeply understand users' intent during editing session. By analyzing these data and using machine learning approaches, we believe that we can create more intelligent and user-friendly systems that synthesize more realistic and diverse effects.

References

[1] Achanta, R., Hemami, S.S., Estrada, F.J., Susstrunk, S.: Frequency-tuned salient region detection. In: CVPR, pp. 1597–1604. IEEE Computer Society (2009)

[2] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., Susstrunk, S.: SLIC superpixels compared to state-of-the-art superpixel methods. IEEE Trans. Pattern Anal. Mach. Intell. **34**(11), pp. 2274–2282 (2012). DOI 10.1109/TPAMI.2012.120. URL http://dx.doi.org/10.1109/TPAMI.2012.120

[3] An, X., Pellacini, F.: AppProp: All-pairs appearance-space edit propagation. In: ACM SIGGRAPH '08, pp. 40:1–40:9 (2008). DOI 10.1145/1399504.1360639. URL http://doi.acm.org/10.1145/1399504.1360639

[4] Ashikhmin, M.: Synthesizing natural textures. In: Proceedings of the 2001 Symposium on Interactive 3D Graphics, I3D '01, pp. 217–226. ACM, New York, NY, USA (2001). DOI 10.1145/364338.364405. URL http://doi.acm.org/10.1145/364338.364405

[5] Bandeira, D., Walter, M.: Synthesis and transfer of time-variant material appearance on images. In: 2009 XXII Brazilian Symposium on Computer Graphics and Image Processing, pp. 32–39 (2009). DOI 10.1109/SIBGRAPI.2009.38

[6] Bandeira, D., Walter, M.: Highlights on weathering effects. The Visual Computer **26**(6), pp. 965–974 (2010). DOI 10.1007/s00371-010-0495-1. URL http://dx.doi.org/10.1007/s00371-010-0495-1

[7] Bengio, Y.: Learning deep architectures for AI. Found. Trends Mach. Learn. **2**(1), pp. 1–127 (2009). DOI 10.1561/2200000006. URL http://dx.doi.org/10.1561/2200000006

[8] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., Montreal, U.D., Quebec, M.: Greedy layer-wise training of deep networks. In: In NIPS. MIT Press, pp. 153–160 (2007)

[9] Bie, X., Huang, H., Wang, W.: Real time edit propagation by efficient sampling. Comput. Graph. Forum **30**(7), pp. 2041–2048 (2011)

[10] Bousseau, A., Paris, S., Durand, F.: User-assisted intrinsic images. In: ACM SIGGRAPH Asia '09, pp. 130:1–130:10 (2009). DOI 10.1145/1661412.1618476. URL http://doi.acm.org/10.1145/1661412.1618476

[11] Bousseau, A., Paris, S., Durand, F.: User-assisted intrinsic images. In: ACM SIGGRAPH Asia 2009 Papers, SIGGRAPH Asia '09, pp. 130:1–130:10. ACM, New York, NY, USA (2009). DOI 10.1145/1661412.1618476. URL http://doi.acm.org/10.1145/1661412.1618476

[12] Boyadzhiev, I., Bala, K., Paris, S., and Adelson, E.: Band-shifting decomposition for image-based material editing, *ACM Trans. Graph*, 34, 5, 163:1-164:16, 2015.

[13] Bronstein, A.M., Bronstein, M.M., Zibulevsky, M., Zeevi, Y.Y.: Sparse ICA for blind separation of transmitted and reflected images. International Journal of Imaging Systems and Technology **15**(1), pp. 84–91 (2005). URL http://dx.doi.org/10.1002/ima.20042

[14] Chen, X., Zou, D., Li, J., Cao, X., Zhao, Q., Zhang, H.: Sparse dictionary learning for edit propagation of high-resolution images. In: CVPR 2014, pp. 2854–2861 (2014). DOI 10.1109/CVPR.2014.365. URL http://dx.doi.org/10.1109/CVPR.2014.365

[15] Chen, X., Zou, D., Zhao, Q., Tan, P.: Manifold preserving edit propagation. ACM Trans. Graph. **31**(6), pp. 132:1–132:7 (2012). DOI 10.1145/2366145.2366151. URL http://doi.acm.org/10.1145/2366145.2366151

[16] Cho, H., Lee, H., Kang, H., Lee, S.: Bilateral texture filtering. ACM Trans. Graph. **33**(4), pp. 128:1–128:8 (2014). DOI 10.1145/2601097.2601188. URL http://doi.acm.org/10.1145/2601097.2601188

[17] Chuang, Y.Y., Goldman, D.B., Curless, B., Salesin, D.H., Szeliski, R.: Shadow matting and compositing. ACM Trans. Graph. **22**(3), pp. 494–500 (2003)

[18] Chuang, Y.Y., Goldman, D.B., Zheng, K.C., Curless, B., Salesin, D.H., Szeliski, R.: Animating pictures with stochastic motion textures. In: ACM SIGGRAPH 2005 Papers, SIGGRAPH '05, pp. 853–860 (2005)

[19] Chuang, Y.Y., Zongker, D.E., Hindorff, J., Curless, B., Salesin, D.H., Szeliski, R.: Environment matting extensions: Towards higher accuracy and real-time capture. In: Siggraph 2000, Computer Graphics Proceedings, pp. 121–130, (2000)

[20] Criminisi, A., Sharp, T., Rother, C., P'erez, P.: Geodesic image and video editing. ACM Trans. Graph. **29**(5), pp. 134:1–134:15 (2010). DOI 10.1145/1857907.1857910. URL http://doi.acm.org/10.1145/1857907.1857910

[21] Daugman, J.G.: Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. J. Opt. Soc. Am. A **2**(7), pp. 1160–1169 (1985). URL http://josaa.osa.org/abstract.cfm?URI=josaa-2-7-1160

[22] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR '09 (2009)

[23] Deng, L., Li, J., Huang, J., Yao, K., Yu, D., Seide, F., Seltzer, M.L., Zweig, G., He, X., Williams, J., Gong, Y., Acero, A.: Recent advances in deep learning for speech research at microsoft. In: IEEE ICASSP '13, pp. 8604–8608 (2013). DOI 10.1109/ICASSP.2013.6639345. URL http://dx.doi.org/10.1109/ICASSP.2013.6639345

[24] Desbenoit, B., Galin, E., Akkouche, S.: Simulating and modeling lichen growth. Computer Graphics Forum **23**(3), pp. 341–350 (2004). DOI 10.1111/j.1467-8659.2004.00765.x. URL http://dx.doi.org/10.1111/j.1467-8659.2004.00765.x

[25] Dorsey, J., Edelman, A., Jensen, H.W., Legakis, J., Pedersen, H.K.: Modeling and rendering of weathered stone. In: ACM SIGGRAPH 2005 Courses, SIGGRAPH '05, pp. 225–234, ACM, New York, NY, USA (2005). DOI 10.1145/1198555.1198697. URL http://doi.acm.org/10.1145/1198555.1198697

[26] Dorsey, J., Hanrahan, P.: Modeling and rendering of metallic patinas. In: ACM SIGGRAPH 2005 Courses, SIGGRAPH '05, pp. 387–396, ACM, New York, NY, USA (2005). DOI 10.1145/1198555.1198695. URL http://doi.acm.org/10.1145/1198555.1198695

[27] Dorsey, J., Pedersen, H.K., Hanrahan, P.: Flow and changes in appearance. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96, pp. 411–420. ACM, New York, NY, USA (1996). DOI 10.1145/237170.237280. URL http://doi.acm.org/10.1145/237170.237280

[28] Eisenacher, C., Lefebvre, S., Stamminger, M.: Texture Synthesis From Photographs, Computer Graphics Forum, 27, 2, pp.419-428, April 2008.

[29] Fang, H., Hart, J.C.: Textureshop: Texture synthesis as a photograph editing tool. ACM Trans. Graph. **23**(3), pp. 354–359 (2004). DOI 10.1145/1015706.1015728. URL http://doi.acm.org/10.1145/1015706.1015728

[30] Farbman, Z., Fattal, R., Lischinski, D.: Diffusion maps for edge-aware image editing. ACM Trans. Graph. **29**(6), pp. 145:1–145:10 (2010). DOI 10.1145/1882261.1866171. URL http://doi.acm.org/10.1145/1882261.1866171

[31] Fattal, R.: Single image dehazing. In: ACM SIGGRAPH 2008 papers, SIGGRAPH '08, pp. 72:1–72:9 (2008)

[32] Finlayson, Graham D. and Hordley, Steven D. and Drew, Mark S.: Removing Shadows from Images. In: ECCV '02, pp. 823–836 (2002)

[33] Gai, K., Shi, Z.W., Zhang, C.S.: Blindly separating mixtures of multiple layers with spatial shifts. In: CVPR, pp. 1–8 (2008). URL http://dx.doi.org/10.1109/CVPR.2008.4587343

[34] Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: G.J. Gordon, D.B. Dunson (eds.) AISTATS '11, vol. 15, pp. 315–323 (2011). URL http://www.jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf

[35] Gu, J., Tu, C.I., Ramamoorthi, R., Belhumeur, P., Matusik, W., Nayar, S.: Time-varying surface appearance: Acquisition, modeling and rendering. In: ACM SIGGRAPH 2006 Papers, SIGGRAPH '06, pp. 762–771. ACM, New York, NY, USA (2006). DOI 10.1145/1179352.1141952. URL http://doi.acm.org/10.1145/1179352.1141952

[36] He, K.M., Sun, J., Tang, X.: Single image haze removal using dark channel prior. In: CVPR, pp. 1956–1963 (2009). URL http://dx.doi.org/10.1109/CVPRW.2009.5206515

[37] Hirota, K., Tanoue, Y., Kaneko, T.: Simulation of three-dimensional cracks. The Visual Computer **16**(7), pp. 371–378 (2000). DOI 10.1007/s003710000069. URL http://dx.doi.org/10.1007/s003710000069

[38] Hsu, S.c., Wong, T.t.: Simulating dust accumulation. IEEE Comput. Graph. Appl. **15**(1), pp. 18–22 (1995). DOI 10.1109/38.364957. URL http://dx.doi.org/10.1109/38.364957

[39] Jost, T., Contassot-Vivier, S., Vialle, S.: An efficient multi-algorithms sparse linear solver for GPUs. In: ParCo2009. Lyon, France (2009). URL https://hal.inria.fr/inria-00430520

[40] Khan, E.A., Reinhard, E., Fleming, R., Buelthoff, H.: Image-based material editing. ACM Transactions on Graphics (Proceedings of SIGGRAPH) **25**(3) pp. 654–663 (2006)

[41] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014). URL http://arxiv.org/abs/1412.6980

[42] Kratz, L., Nishino, K.: Factorizing scene albedo and depth from a single foggy image. In: ICCV, pp. 1701–1708. IEEE (2009). URL http://dx.doi.org/10.1109/ICCV.2009.5459382

[43] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25, pp. pp. 1097–1105 (2012). URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[44] Lane, N.D., Georgiev, P., Qendro, L.: DeepEar: Robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In: UbiComp '15, pp. 283–294 (2015). DOI 10.1145/2750858.2804262. URL http://doi.acm.org/10.1145/2750858.2804262

[45] Lang, M., Wang, O., Aydin, T., Smolic, A., Gross, M.: Practical temporal consistency for image-based graphics applications. ACM Trans. Graph. **31**(4), pp. 34:1–34:8 (2012). DOI 10.1145/2185520.2185530. URL http://doi.acm.org/10.1145/2185520.2185530

[46] Le, Q.V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Ng, A.Y.: On optimization methods for deep learning. In: ICML, pp. 265–272 (2011)

[47] Levin, A., Lischinski, D., Weiss, Y.: Colorization using optimization. ACM Trans. Graph. **23**(3), pp. 689–694 (2004). DOI 10.1145/1015706.1015780. URL http://doi.acm.org/10.1145/1015706.1015780

[48] Levin, A., Rav-Acha, A., Lischinski, D.: Spectral matting. In: CVPR. IEEE Computer Society (2007)

[49] Levin, A., Weiss, Y.: User assisted separation of reflections from a single image using a sparsity prior. IEEE Trans. Pattern Analysis and Machine Intelligence **29**(9), pp. 1647–1654 (2007). URL http://dx.doi.org/10.1109/TPAMI.2007.1106

[50] Levin, A., Zomet, A., Weiss, Y.: Separating reflections from a single image using local features. In: CVPR (1), pp. 306–313 (2004). URL http://doi.ieeecomputersociety.org/10.1109/CVPR.2004.226

[51] Li, G., Yu, Y.: Visual saliency based on multiscale deep features. In: CVPR 2015 (2015)

[52] Li, Y., Adelson, E., Agarwala, A.: Scribbleboost: Adding classification to edge-aware interpolation of local image and video adjustments. In: EGSR '08, pp. 1255–1264 (2008). DOI 10.1111/j.1467-8659.2008.01264.x. URL http://dx.doi.org/10.1111/j.1467-8659.2008.01264.x

[53] Li, Y., Ju, T., Hu, S.M.: Instant propagation of sparse edits on images and videos. Comput. Graph. Forum **29**(7), pp. 2049–2054 (2010)

[54] Lin, M., Chen, Q., Yan, S.: Network in network. CoRR **abs/1312.4400** (2013). URL http://arxiv.org/abs/1312.4400

[55] Lischinski, D., Farbman, Z., Uyttendaele, M., Szeliski, R.: Interactive local adjustment of tonal values. ACM Trans. Graph. **25**(3), pp. 646–653 (2006). DOI 10.1145/1141911.1141936. URL http://doi.acm.org/10.1145/1141911.1141936

[56] Liu, F., Shen, C., Lin, G.: Deep convolutional neural fields for depth estimation from a single image. In: CVPR 2015 (2015)

[57] Lowe, D.G.: Object recognition from local scale-invariant features. In: ICCV '99, pp. 1150–1157 (1999). URL http://dl.acm.org/citation.cfm?id=850924.851523

[58] Luan, Q., Wen, F., Cohen-Or, D., Liang, L., Xu, Y.Q., Shum, H.Y.: Natural image colorization. In: EGSR '07, pp. 309–320 (2007)

[59] Mérillou, S., Ghazanfarpour, D.: Technical section: A survey of aging and weathering phenomena in computer graphics. Comput. Graph. **32**(2), pp. 159–174 (2008). DOI 10.1016/j.cag.2008.01.003. URL http://dx.doi.org/10.1016/j.cag.2008.01.003

[60] Motoyoshi, I., Nishida, S., Sharan, L. and Adelson, E.H.: Image statistics and the perception of surface qualities, Nature, May 10; 447(7141): 2006-2009, 2007.

[61] Musialski, P., Cui, M., Ye, J., Razdan, A., Wonka, P.: A framework for interactive image color editing. Vis. Comput. **29**(11), pp. 1173–1186 (2013). DOI 10.1007/s00371-012-0761-5. URL http://dx.doi.org/10.1007/s00371-012-0761-5

[62] Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., Ng, A.Y.: Multimodal deep learning. In: ICML, pp. 689–696 (2011)

[63] Paquette, E., Poulin, P., Drettakis, G.: The simulation of paint cracking and peeling, Proceedings of Graphics Interface 2002, pp. 59–68, Calgary, Alberta, Canada, 27 - 29 May 2002

[64] Park, J., Heo, N., Choi, S., Shin, S.Y.: Tour into the picture with water surface reflection and object movements: Research articles. Comput. Animat. Virtual Worlds **17**(3-4), pp. 315–324 (2006)

[65] Peers, P., Dutre, P.: Wavelet environment matting. In: Proceedings of the 14th Eurographics Workshop on Rendering, pp. pp. 157–166 (2003)

[66] Pellacini, F., Lawrence, J.: AppWand: Editing measured materials using appearance-driven optimization. ACM Trans. Graph. **26**(3) (2007). DOI 10.1145/1276377.1276444. URL http://doi.acm.org/10.1145/1276377.1276444

[67] Pérez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Transactions on Graphics **22**(3), pp. 313–318 (2003)

[68] Qu, Y., Wong, T.T., Heng, P.A.: Manga colorization. ACM Trans. Graph. **25**(3), pp. 1214–1220 (2006)

[69] Reinhard, E., Ashikhmin, M., Gooch, B., Shirley, P.: Color transfer between images. IEEE Computer Graphics and Applications **21**(5), pp. 34–41 (2001)

[70] Ren, C.Y., Reid, I.: gslic: a real-time implementation of slic superpixel segmentation. Tech. rep., University of Oxford, Department of Engineering Science (2011)

[71] Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., Rott, P.: A perceptually motivated online benchmark for image matting. In: CVPR 2009 (2009)

[72] Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., Rott, P.: A perceptually motivated online benchmark for image matting. In: CVPR '09, pp. 1826–1833 (2009)

[73] Sawayama, M. and Nishida, S. (2015). Visual perception of surface wetness. Vision Sciences Society 2015, St. Pete Beach, Florida, USA, May 15-20, 2015.

[74] Schlick, C.: An inexpensive BRDF model for physically-based rendering. Computer Graphics Forum **13**(3), pp. 233–246 (1994)

[75] Shen, L., Tan, P., Lin, S.: Intrinsic image decomposition with non-local texture cues. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pp. 1–7 (2008). 00044

[76] Shen, W., Wang, X., Wang, Y., Bai, X., Zhang, Z.: Deepcontour: A deep convolutional feature learned by positive-sharing loss for contour detection. In: CVPR '15 (2015)

[77] Shih, Yichang and Paris, Sylvain and Durand, Frédo and Freeman, William T.: Data-driven Hallucination of Different Times of Day from a Single Outdoor Photo. ACM Trans. Graph., 32, 6, pp. 200:1–200:11 (2013).

[78] Subr, K., Soler, C., Durand, F.: Edge-preserving multiscale image decomposition based on local extrema. In: ACM SIGGRAPH Asia 2009 Papers, SIGGRAPH Asia '09, pp. pp. 147:1–147:9. ACM, New York, NY, USA (2009). DOI 10.1145/1661412.1618493. URL http://doi.acm.org/10.1145/1661412.1618493

[79] Sun, J., Cao, W., Xu, Z., Ponce, J.: Learning a convolutional neural network for non-uniform motion blur removal. In: CVPR '15 (2015)

[80] Tan, R.T.: Visibility in bad weather from a single image. In: CVPR, pp. 1–8 (2008). URL http://dx.doi.org/10.1109/CVPR.2008.4587643

[81] Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: ICCV, pp. 839–846 (1998)

[82] Wang, J., Cohen, M.F.: Optimized color sampling for robust matting. In: CVPR, pp. 1–8 (2007). URL http://dx.doi.org/10.1109/CVPR.2007.383006

[83] Wang, J., Cohen, M.F.: Image and video matting: a survey. In: Foundations and Trends in Computer Graphics and Vision, vol. 3 pp. 97–175 (2008)

[84] Wang, J., Tong, X., Lin, S., Pan, M., Wang, C., Bao, H., Guo, B., Shum, H.Y.: Appearance manifolds for modeling time-variant appearance of materials. In: ACM SIGGRAPH 2006 Papers, SIGGRAPH '06, pp. 754–761. ACM, New York, NY, USA (2006). DOI 10.1145/1179352.1141951. URL http://doi.acm.org/10.1145/1179352.1141951

[85] Wang, L., Lu, H., Ruan, X., Yang, M.H.: Deep networks for saliency detection via local estimation and global search. In: CVPR '15 (2015)

[86] Wexler, Y., Fitzgibbon, A., Zisserman, A.: Image-based environment matting. In: Proceedings of the 13th Eurographics Workshop on Rendering (RENDERING TECHNIQUES-02), pp. 279–290 (2002)

[87] Wu, J., Yu, Y., Huang, C., Yu, K.: Deep multiple instance learning for image classification and auto-annotation. In: CVPR 2015 (2015)

[88] Wu, T.P., Tang, C.K., Brown, M.S., Shum, H.Y.: Natural shadow matting. ACM Trans. Graph **26**(2) (2007). URL http://doi.acm.org/10.1145/1243980.1243982

[89] Wu, Z., Jiang, Y.G., Wang, J., Pu, J., Xue, X.: Exploring inter-feature and inter-class relationships with deep neural networks for video classification. In: ACM MM '14, pp. 167–176 (2014). DOI 10.1145/2647868.2654931. URL http://doi.acm.org/10.1145/2647868.2654931

[90] Xiao, T., Xu, Y., Yang, K., Zhang, J., Peng, Y., Zhang, Z.: The application of two-level attention models in deep convolutional neural network for fine-grained image classification. In: CVPR '15 (2015)

[91] Xu, K., Li, Y., Ju, T., Hu, S.M., Liu, T.Q.: Efficient affinity-based edit propagation using k-d tree. In: ACM SIGGRAPH Asia '09, pp. 118:1–118:6. New York, NY, USA (2009). DOI 10.1145/1661412.1618464. URL http://doi.acm.org/10.1145/1661412.1618464

[92] Xu, K., Wang, J., Tong, X., Hu, S.M., Guo, B.: Edit propagation on bidirectional texture functions. Computer Graphics Forum **28**(7), pp. 1871–1877 (2009)

[93] Xu, L., Lu, C., Xu, Y., Jia, J.: Image smoothing via l0 gradient minimization. ACM Trans. Graph. **30**(6), pp. 174:1–174:12 (2011). DOI 10.1145/2070781.2024208. URL http://doi.acm.org/10.1145/2070781.2024208

[94] Xu, L., Yan, Q., Jia, J.: A sparse control model for image and video editing. ACM Trans. Graph. **32**(6), pp. 197:1–197:10 (2013). DOI 10.1145/2508363.2508404. URL http://doi.acm.org/10.1145/2508363.2508404

[95] Xuey, S., Wang, J., Tong, X., Dai, Q., Guo, B.: Image-based material weathering. Computer Graphics Forum **27**(2), pp. 617–626 (2008). DOI 10.1111/j.1467-8659.2008.01159.x. URL http://dx.doi.org/10.1111/j.1467-8659.2008.01159.x

[96] Yan, Z., Zhang, H., Wang, B., Paris, S., Yu, Y.: Automatic photo adjustment using deep neural networks. ACM Trans. Graph. **35**(2) (2015)

[97] Yatagawa, T., Yamaguchi, Y.: Sparse pixel sampling for appearance edit propagation. The Visual Computer **31**(6-8), pp. 1101–1111 (2015)

[98] Yatziv, L., Sapiro, G.: Fast image and video colorization using chrominance blending. IEEE Transactions on Image Processing **15**(5), pp. 1120–1129 (2006)

[99] Yeung, S.K., Tang, C.K., Brown, M.S., Kang, S.B.: Matting and compositing of transparent and refractive objects. ACM Transactions on Graphics **30**(1), pp. 2:1–2:13 (2011). DOI http://dx.doi.org/10.1145/1899404.1899406

[100] Yin, X., Fujimoto, T., Chiba, N.: Cg representation of wood aging with distortion, cracking and erosion. The Journal of the Society for Art and Science **3**(4), pp. 216–223 (2004). DOI 10.3756/artsci.3.216

[101] Zelinka, S., Fang, H., Garland, M., Hart, J.C.: Interactive material replacement in photographs. In: Proceedings of Graphics Interface 2005, GI '05, pp. 227–232. Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada (2005). URL http://dl.acm.org/citation.cfm?id=1089508.1089546

[102] Zongker, D.E., Werner, D.M., Curless, B., Salesin, D.: Environment matting and compositing. In: SIGGRAPH, pp. 205–214 (1999). URL http://doi.acm.org/10.1145/311535.311558