Expressive image manipulations for a variety of visual representations

Adrien Bousseau

Université Joseph Fourier de Grenoble (UJF)

EXPRESSIVE IMAGE MANIPULATIONS FOR A VARIETY OF VISUAL REPRESENTATIONS

Manipulations d'image expressives
pour une variété de représentations visuelles

Adrien BOUSSEAU

Thèse présentée pour l'obtention du titre de
Docteur de l'Université Joseph Fourier
Spécialité Informatique
Préparée au sein du laboratoire Jean Kuntzmann (LJK)
Dans le cadre de l'École Doctorale Mathématique,
Sciences et Technologies de l'Information, Informatique

Soutenance prévue le 15 octobre 2009 devant le jury composé de:

James L. CROWLEY	Président
George DRETTAKIS	Rapporteur
Doug DECARLO	Rapporteur
David SALESIN	Examinateur
François SILLION	Directeur de thèse
Joëlle THOLLOT	Co-Directrice de thèse

Abstract

Visual communication greatly benefits from the large variety of appearances that an image can take. By neglecting spurious details, simplified images focus the attention of an observer on the essential message to transmit. Stylized images, that depart from reality, can suggest subjective or imaginary information. More subtle variations, such as change of lighting in a photograph can also have a dramatic effect on the interpretation of the transmitted message.

The goal of this thesis is to allow users to manipulate visual content and create images that corresponds to their communication intent. We propose a number of manipulations that modify, simplify or stylize images in order to improve their expressive power.

We first present two methods to remove details in photographs and videos. The resulting simplification enhances the relevant structures of an image. We then introduce a novel vector primitive, called Diffusion Curves, that facilitates the creation of smooth color gradients and blur in vector graphics. The images created with diffusion curves contain complex image features that are hard to obtain with existing vector primitives. In the second part of this manuscript we propose two algorithms for the creation of stylized animations from 3D scenes and videos. The two methods produce animations with the 2D appearance of traditional media such as watercolor. Finally, we describe an approach to decompose the illumination and reflectance components of a photograph. We make this ill-posed problem tractable by propagating sparse user indications. This decomposition allows users to modify lighting or material in the depicted scene.

The various image manipulations proposed in this dissertation facilitates the creation of a variety of visual representations, as illustrated by our results.

Keywords Expressive Rendering, Non Photorealistic Rendering, Image Simplification, Vector Graphics, Image Editing.

Résumé

La communication visuelle tire profit de la grande variété d'apparences qu'une image peut avoir. En ignorant les détails, les images simplifiées concentrent l'attention de l'observateur sur le contenu essentiel à transmettre. Les images stylisées, qui diffèrent de la réalité, peuvent suggérer une information subjective ou imaginaire. Des variations plus subtiles, comme le changement de l'éclairage dans une photographie, ont également un impact direct sur la façon dont le message transmis va être interprété.

Le but de cette thèse est de permettre à un utilisateur de manipuler le contenu visuel et créer des images qui correspondent au message qu'il cherche à transmettre. Nous proposons plusieurs manipulations qui modifient, simplifient ou stylisent des images pour augmenter leur pouvoir d'expression.

Nous présentons d'abord deux méthodes pour enlever les détails d'une photographie ou d'une vidéo. Le résultat de cette simplification met en valeur les structures importantes de l'image. Nous introduisons ensuite une nouvelle primitive vectorielle, nommée Courbe de Diffusion, qui facilite la création de dégradés de couleurs et de flou dans des images vectorielles. Les images créées avec des courbes de diffusion présentent des effets complexes qui sont difficiles à reproduire avec les outils vectoriels existants. Dans une seconde partie, nous proposons deux algorithmes pour la création d'animations stylisées à partir de vidéos et de scènes 3D. Ces deux méthodes produisent des animations qui ont l'apparence 2D de média traditionnels comme l'aquarelle. Nous décrivons enfin une approche pour décomposer l'information d'illumination et de réflectance dans une photographie. Nous utilisons des indications utilisateurs pour résoudre ce problème sous-contraint.

Les différentes manipulations d'image proposées dans ce mémoire facilitent la création d'une variété de représentations visuelles, comme illustré par nos résultats.

Mots Clefs Rendu expressif, rendu non-photoréaliste, simplification d'image, dessin vectoriel, edition d'image.

Remerciements

Beaucoup de personnes méritent des remerciements pour cette thèse, et la liste qui suit n'est pas exhaustive. Un grand merci tout d'abord à mes superviseurs. En premier lieu Joëlle Thollot pour son encadrement, son écoute et sa disponibilité; Joëlle m'a offert une grande liberté qui m'a encouragé à prendre des initiatives et proposer mes propres solutions. Merci également à François Sillion pour m'avoir accueilli dans l'équipe Artis et m'avoir présenté à David Salesin avant même mon début de thèse. J'en profite pour remercier David Salesin pour son accueil et encadrement lors de mes stages à Adobe; c'est lors de mon premier séjour à Seattle que j'ai véritablement appris ce qu'était une deadline SIGGRAPH! Merci enfin à Frédo Durand, aussi bien pour son expertise scientifique que pour ses conseils pratiques sur le "comment" de la recherche. Je ne pouvais pas espérer de meilleurs mentors pour guider mes premiers pas dans la communauté de la synthèse d'image.

Merci ensuite à mes "grands frères", vieux thésards ou jeunes chercheurs, pour leur aide et leurs conseils. Merci à Pascal Barla pour son enthousiasme et les longues discussions autour d'un tableau blanc ou d'un verre de rouge (Bordeaux bien entendu). Merci à Holger Winnemöller pour sa bonne humeur contagieuse et ses quelques mots de français toujours bien placés. Et enfin merci à Sylvain Paris pour avoir partagé avec moi son expérience et pour m'avoir hébergé lorsque j'étais sans-abri à Boston. Je remercie également tous ceux qui m'ont guidé et conseillé lors de ma scolarité, et en particulier Laurent Joyeux qui m'a suggéré de candidater dans une école d'ingénieur après ma maitrise: c'était une fameuse idée! Je suis également reconnaissant envers Doug DeCarlo et George Drettakis pour avoir accepté d'être les rapporteurs de cette thèse, et pour leurs retours constructifs sur mon manuscrit et ma soutenance.

Merci à toute l'équipe Artis pour les pauses café, les randos (et la liste de diffusion correpondante tenue par Fabrice Neyret) et les sorties ski. Je remercie en particulier Pierre Bénard pour sa gentillesse et ses tartes meringuées, et Kartic Subr pour ses conseils culinaires épicés, et pour avoir corrigé les nombreuses fautes d'anglais que j'avais laissé trainer dans mon manuscrit. Merci également à Laurence Boissieux pour sa patience lors de l'utilisation de nos prototypes de recherche buggés.

Merci à mes parents pour leurs constants encouragements tout au long de mes études, pour leur intéret dans mon travail, et pour supporter mon éloignement géographique. Grand merci enfin à Alexandrina Orzan pour tous les bons moments que nous passons ensemble.

Contents

Part I

Introduction

Ce manuscrit est en anglais, un résumé de chaque chapitre en français est proposé en annexe A.

Visual content is often used in our modern society to convey ideas and information. Road signs, websites, television, advertisement boardings, packaging and comic books are only a few examples of images that populate our environment and transmit visual messages. The variety of these examples illustrates that different types of images are needed to convey different messages. Photographs are nowadays one of the simplest visual representation to produce and capture the **reality** the photographer sees at the moment of the take. Variations in the capture setup, such as change of viewpoint, change of lighting, or change of material, can have a dramatic impact on the perception of the resulting image (Figure 1(a) and (b)). Because the visual clutter of realistic images can distract the observer from the relevant information to transmit, **simplified** representations are used to focus the attention of the observer and allow a rapid and unequivocal interpretation of the message (Figure 1(c)). Illustrators also use **stylized** depictions that depart from reality to transmit subjective information and engage the observer's imagination (Figure 1(d)).

(a) Random picture
ⓒ63vwdriver@flickr.com

(b) Sunset lighting
ⓒMr.Photo@flickr.com

(c) Blueprint
ⓒwww.the-blueprints.com

(d) Stylized drawing
ⓒKevin Kidney, Disney

Figure 1: *Different images, different meaning. Compared to a random picture of a VW Beetle (a), a sunset lighting (b) emphasis the image of freedom associated with this economy car. A blueprint (c) retains the essential intrinsic components of the car while suppressing distracting visual information. This simplification improves shape depiction. A stylized drawing (d) suggests subjective information, such as the cute aspect of the little car.*

In his seminal study of comic books, McCloud [McC94] positions these different types of images (realistic, simplified, stylized) in a triangle (Figure 2) delimited on the bottom by the "representational edge" that corresponds to *what* is depicted, and on the left by the "retinal edge" that corresponds to *how* it is depicted. The right side is the "language edge" where images become symbols and eventually text.

We take inspiration from this triangle to organize the different types of images we are interested in. In this dissertation we only study representational images, and do not consider the right

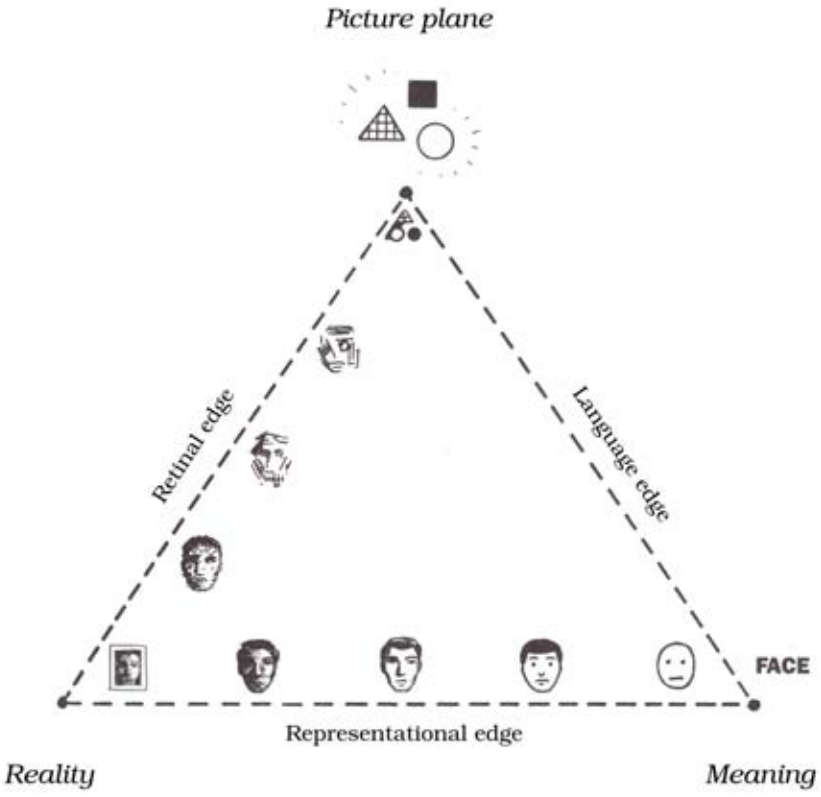


Figure 2: *McCloud's triangle [McC94]. ©Scott McCloud.*

corner (text) and top corner (pure shapes) of McCloud's triangle. We propose the 2D space illustrated in Figure 3 composed of one axis going from realistic to stylized images, and one going from complex to simple images.

Given this organization, the goal of this thesis is to **allow the user to express her intended message in an image by navigating in this space of visual representations**. With this goal in mind, image manipulations are used to navigate from one image to another. An image manipulation can modify the content of an image while preserving its position in the 2D space, or simplify or stylize the content in order to move along the *what* and *how* axis of our 2D space. However, it is important to note that the borders between the three main areas of the 2D space (realistic, stylized and simplified) remain fuzzy. For example, simplification is inherent to many styles, either because of technical constraints (the use of a coarse brush, the finite time involved in the drawing) or because of artistic choices. Even if the photographs of Ansel Adams depict natural scenes realistically, they are also stylized by the use of highly contrasted black and white tones (Figure 4). Our categorization aims at differentiating various types of depiction and image manipulations, but should not be considered as a unique classification. In the following sections, we describe in more details the three main types of images we are interested in, and motivate their manipulation.

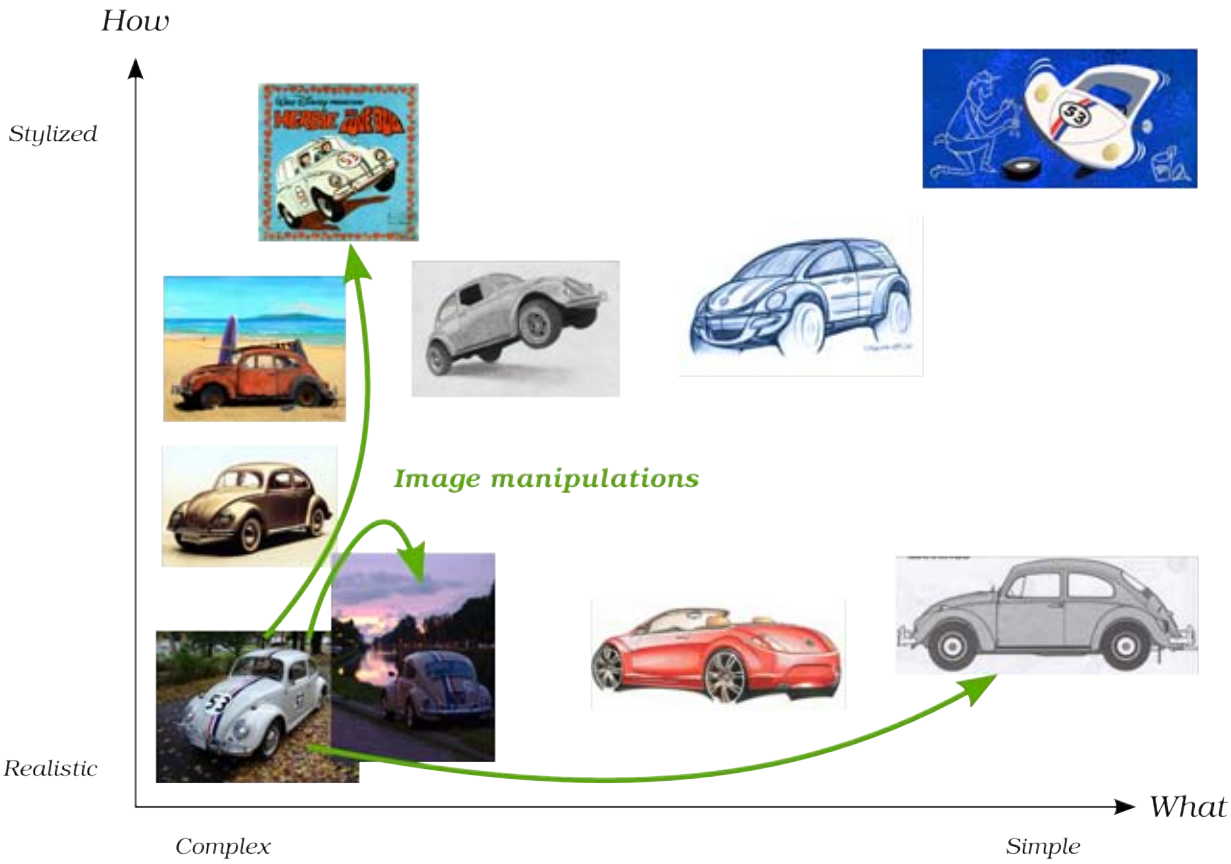
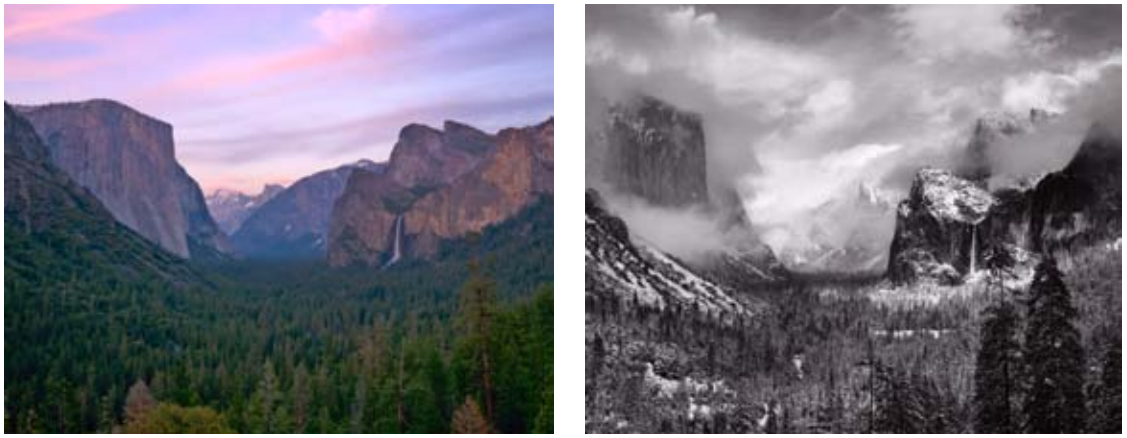


Figure 3: *We organize images in a 2D space of stylization and simplification. Image manipulations are used to navigate in this space.*



(a) Yosemite, ©Ivan Makarov@flickr.com (b) Yosemite, ©Ansel Adams

Figure 4: *The photographs of Ansel Adams (b) display realistic content with distinctive highly contrasted tones, proper to his style.*

1 Realistic Images

Photographs are widely produced visual representations. Digital cameras make the capture of photographs a simple and fast process, and post-processing edits such as cropping or red eye correction are now commonplace. However, a given picture is only a biased representation of reality that results from the capture of a particular scene under one specific setup [Dur00]. The choice of focus, exposure, lighting has a direct impact on the image and the way it will be interpreted afterwards. For example, a long exposure can be used to suggest movement via motion blur (Figure 5(a)), while a short exposure will "freeze" the moment that cannot be perceived by a naked eye (Figure 5(b)). Most of these choices need to be done during the capture, which restricts photographs to only transmit the message intended at the particular time of the shot. The ability to vary the settings after the capture would represent a powerful tool to adapt a picture to changes of intent.

(a) Motion blurred image
ⓒi am bad@flickr.com

(b) High speed image
ⓒA.Connah@flickr.com

Figure 5: *Motion blur depicts motion in still images (a), while high speed photography can capture moments invisible to a naked eye (b).*

Even with the desired capture setup in place, photographers have rarely complete control over what falls in the viewpoint: undesired objects, material or lighting can dramatically alter the original intention. For instance, a sunset lighting may be preferable to daylight in order to convey the warm atmosphere of a place (Figure 6(a) and (b)), while the presence of haze can produce a more mysterious and oppressive mood (Figure 6(c)). It takes the skills and patience of a talented artist to master such photographic effects and for many point-and-shoot photographers the resulting pictures do not match the image they had in mind.

Photograph manipulation is as old as photography itself, and aims at removing the restriction of "what you see is what you get" by modifying the content of a photograph after its capture. Figure 7 illustrates early image manipulations where objects or characters were added or removed from the photograph to accord with political preferences. The main challenge in photograph manipulation is to preserve the complexity and realism of the image while modifying its content. The research field of *Computational Photography* aims at facilitating this process by means of

(a) Taj Mahal under sunlight
ⓒironmanixs@flickr.com

(b) Taj Mahal at sunset
ⓒbetta design@flickr.com

(c) Taj Mahal through haze
ⓒfoxypar4@flickr.com

Figure 6: *Variations in the scene, such as lighting (b) or haze (c), alter the atmosphere of a picture.*

additional information about the captured scene. This information can be obtained from novel camera designs, computer vision algorithms or user interaction. In that spirit, Chapter 7 of this dissertation proposes a user assisted method to decorrelate the lighting and material component of a single photograph in order to edit these pieces of information independently.

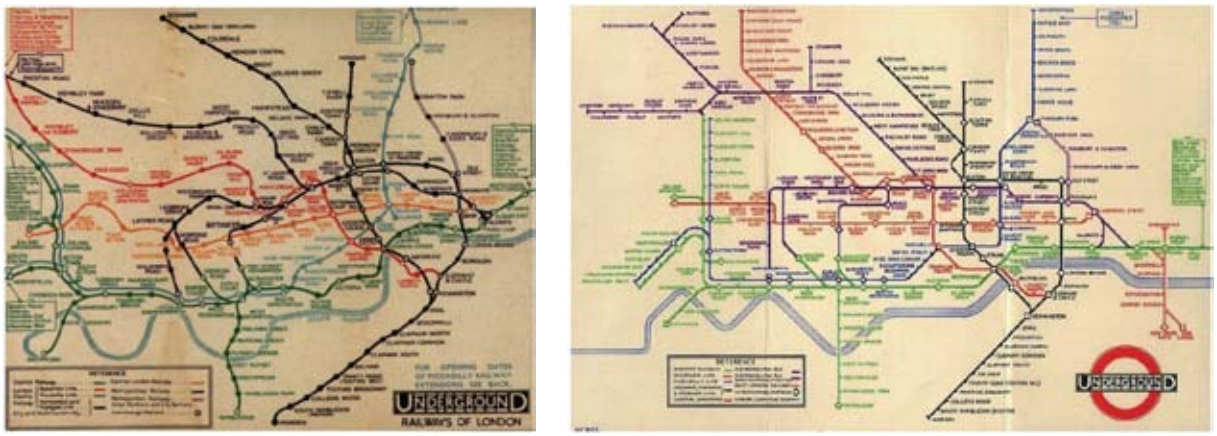
(a) General Sherman and colleagues, 1865

(b) Stalin, 1930

Figure 7: *Manipulation of photographs is as old as photography itself. Original pictures on left. In (a), a general is added to the picture, while in (b), a commissar is removed. Images from http://www.cs.dartmouth.edu/farid/research/digitaltampering/.*

2 Simplified Images

Simple representations communicate better than photographs. When starting from complex images, *simplification* (which is a form of *abstraction*) retains only the information to transmit while neglecting unnecessary details. As pointed by McCloud [McC94], simplified images amplify the essential meaning and focus the attention of the observer on *what* is represented. Symbols are used for road signs or instructions because they are quickly understandable and unambiguous. A good illustration of this concept of simplification is the London Underground Map created by Harry Beck (Figure 8). In contrast to accurate geographical maps that reproduce exact distances and angles, the simplified map only retains the information needed to easily navigate in the underground (station order and line junctions). Introduced in 1933, this concept has proved to be very effective and has been adopted by many transportation agencies over the world.



(a) London Underground Map 1928 (b) London Underground Map 1933

Figure 8: *Compared to the geographically accurate London Underground Map from 1928 (a), the simplified map by Harry Beck from 1933 only retains the relevant information necessary to navigate around the London Underground. Example from [Kra07].*

Kramer [Kra07] notes that the relevance of a simplification strongly depends of the task at hand: the London Underground Map can be misleading for tourists who aim at finding their way in the streets of London. Frorer et al. [FMH97] explain that the existing knowledge of the observer also affects his understanding of a given simplification. For example, the concept of stations and lines need to be known to understand an underground map. In order to deal properly with these two observations, semantic knowledge about the information to transmit is required [AS01, GASP08]. In this dissertation, we focus instead on simplification of images without any a priori knowledge of the underlying content. We rely on a user to provide semantic guidance when needed.

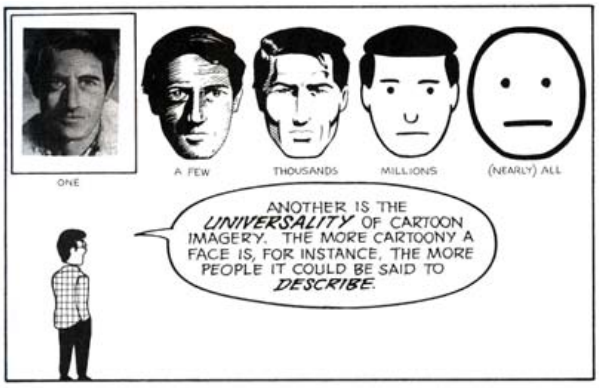


Figure 9: *As pointed by McCloud [McC94], simplified images transmit more general concepts than realistic ones. ©Scott McCloud.*

A second property of simplified images pointed by McCloud is their universality: while a photograph depicts one specific subject, a simplified image can depict any similar subjects (Figure 9). In this context, simplification can be considered as a factorization process, that illustrates general concepts by extracting the common features shared by multiple instances of a class [Kra07]. As a result, observers tend to interpret simplified images in a subjective manner and relate the depicted subject to themselves. For example, while a photograph of a car is perceived as "the car of somebody else", the silhouette of a car on a road sign is interpreted as "my car".

Finally, varying the amount of detail over an image is also an effective way to attract the attention of the observer to specific parts of the scene to depict. This effect is well known by photographers who employ shallow depth of field to blur the background and isolate the object of interest in a picture (Figure 10(a)). Similar spatially varying simplifications can be produced in traditional paintings, although more freedom is let to the artist by means of simplification. Common simplifications include the use of coarse brush strokes, desaturated colors or line drawings (Figure 10(b) and (c)). McCloud also notes that comic book authors often use various level of simplification over an image so that readers identify themselves to simplified characters that evolves in realistic environments. This process can also suggest the dynamism of the characters in contrast to their static surroundings (Figure 11).

We present in this document two methods to remove details in images and videos without a priori knowledge of the visual content (Chapter 1 and 2). The user can guide the resulting simplifications to emphasize specific objects in the image, or to mimic the use of coarse brush strokes in stylized rendering.

(a) Shallow depth of field
©carlosluis@flickr.com

(b) Self portrait
Claude Monet

(c) Le portrait d'Olga,
Pablo Picasso

Figure 10: *In photography (a), shallow depth of field is an effective way to attract attention on a specific subject. Similar attention grabbing effect can be obtained in painting by varying the size of brush strokes (b) or reducing the amount of colors (c).*

Figure 11: *Hergé draws simplified characters to suggest their dynamism over a static background. ©Moulinsart S.A.*

3 Stylized Images

In this dissertation the term "stylized" applied to images designates images with visual characteristics that differ from photographs, such as painting or drawing. In our 2D space of visual representations, stylization refers to *how* information is depicted in an image. While photorealistic images reproduce reality objectively, as if it was directly seen through the image plane, stylized images rely on primitives such as paint strokes or pigments to transmit the visual infor-

mation in a more subjective way. Stylization offers an additional dimension to artists to express their message. Taking a look at Figure 12, we can recognize Van Gogh's style through the use of heavy brush strokes and vivid colors, even when he mimics existing master pieces. Compared to his realistic inspiration, Van Gogh's stylized paintings suggest more dynamism in the depicted scene.

(a) La Méridienne
Jean-François Millet

(b) La Méridienne
Vincent Van Gogh

(c) Le Bon Samaritain
Eugène Delacroix

(d) Le Bon Samaritain
Vincent Van Gogh

Figure 12: *Van Gogh introduces his own style when reproducing realistic master pieces.*

It is this departure from reality that provides stylized images with their evocative power. By representing information unrealistically, stylization engages the imagination of the observers who can more easily interpret the image according to their own perspective. Graphic novels authors heavily rely on this process to serve the storytelling [McC94]. Stylized images are used in architecture or archeology to convey the idea that the depicted information is imaginary, and corresponds to either a proposal for something that doesn't exist yet, or an hypothesis for something that doesn't exist anymore (Figure 13).

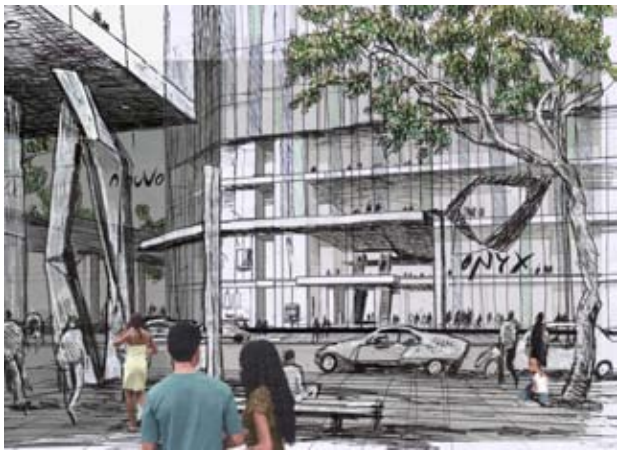


Figure 13: *In this architectural illustration, a pen and ink style is used to convey the idea that the building is only a proposal, while photographs of pedestrians link this proposal to reality.* ©*http://zlgdesign.files.wordpress.com/.*

In this dissertation we focus on the problem of creating *animations* in a watercolor style, from 3D scenes (Chapter 5) and videos (Chapter 6). This research contributes to the field of *Non Photorealistic Rendering* or *Expressive Rendering* that aims at facilitating and accelerating the production of stylized images. Our work on stylized animations is motivated by the benefits a greater style variety can bring to applications such as movies and video games.

The hegemony of realistic rendering in video games and movies leads to a uniformity in their appearance. This prevents a novice observer from distinguishing one production from another. Figure 14(a) and (b) illustrates this trend with two different driving simulations that are visually very similar. In this context, a game that adopts a stylized rendering is appreciated for its original look (Figure 14(c)). The use of stylized images in movies emphasizes the uniqueness of the story compared to previous movies that have the same actors or use the same locations (Figure 15).

(a) Gran Turismo©

(b) Project Gotham Racing©

(c) Auto Modellista©

Figure 14: *Compared to the numerous games rendered realistically (a) and (b), a stylized game (c) offers a fresh distinctive look.*

(a) Constantine© (b) The Matrix© (c) A Scanner Darkly©

Figure 15: *While an actor tends to have the same appearance in every movie (a) and (b), stylized rendering broadens the range of possible look (c).*

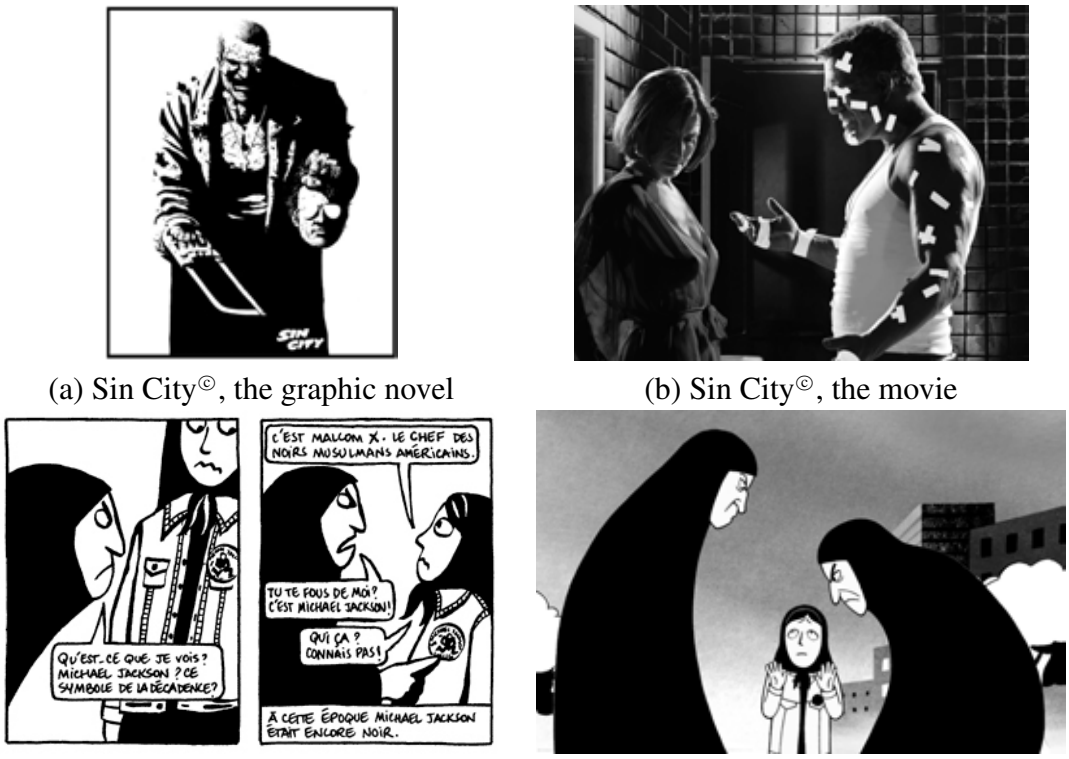
A wide variety of styles are equally valuable for effective storytelling. Movies and video games can thus adapt their style to a story in order to transmit subtle information, after the fashion of graphic novels or children's book. Figure 16 illustrates various styles that transmit information ranging from the culture of a character to the commercial target of a product.

(a) Kill Bill© (b) Renaissance© (c) IPod©

Figure 16: *The rendering style can greatly serve the storytelling. In Kill Bill (a), Quentin Tarentino illustrates the story of a Japanese character using a manga style. In Renaissance (b), a black and white style is used to enhance the oppressive atmosphere of the movie. In opposition, the colorful watercolor style of the IPod advertisement (c) enforces the festive image of the product.*

Finally, a major application of stylized rendering is in the adaptation of existing stylized work on new media. Figure 17 gives example of the adaptation of graphic novels into movies, where the style of the original work has been preserved. Our work on watercolor rendering has been initiated by this last application of stylization via a collaboration with the French studio *Studio Broceliande*[1]. Watercolor was identified by the people of Studio Broceliande as a major contributor to the appreciated style of French and Belgium graphic novels. To target the same audience with animations, they wished to obtain a similar watercolor look in animated 3D rendering.

[1] http://www.studio-broceliande.fr/



(a) Sin City©, the graphic novel (b) Sin City©, the movie

(c) Persepolis©, the graphic novel (d) Persepolis©, the movie

Figure 17: *Stylized rendering allows the preservation of style when adapting existing stories (here graphic novels) to movies.*

4 Our Contributions

The space of visual representations described in the previous sections includes all possible types of representational images. We believe that this large variety of potential images prevents the use of a generic approach to continuously navigate in this space. Our approach consists of **instantiating particular image manipulations** that allow the navigation from one specific point of the space to another. All the methods proposed in this manuscript follow a common three step pipeline. The first step identifies the image features required to perform the manipulation. Images features provide knowledge about the depicted scene, although in many cases partial knowledge is sufficient. For example, video stylization only requires knowledge of the motion in the scene. Image features can be provided by the user, obtained from a 3D scene, or, as in most of the methods proposed in this document, extracted from a photograph or video using computer vision algorithms. The second step manipulates the features through *operators* that can *create*, *remove* or *modify* visual content. Finally, the resulting images are obtained from the manipulated features via dedicated rendering algorithms. We visualize in Figure 18 these three steps of image manipulation.

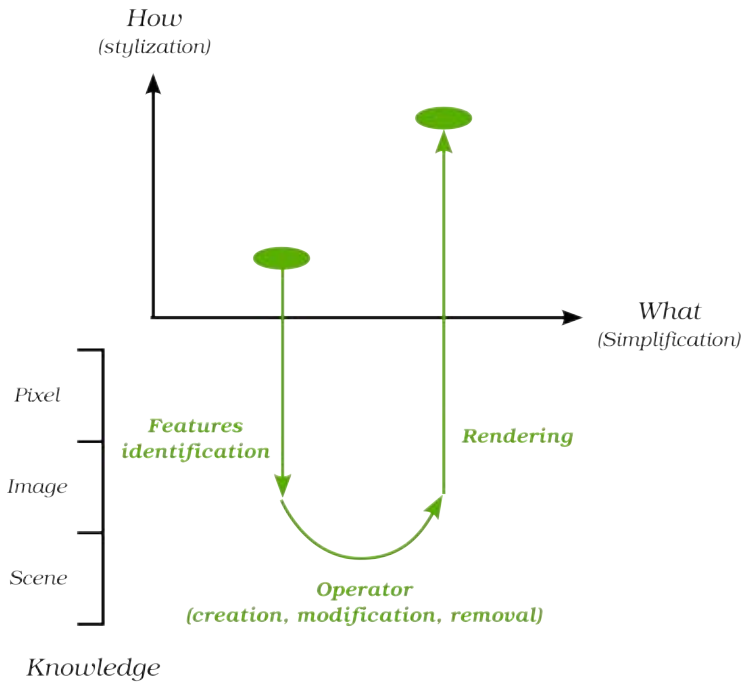


Figure 18: *We visualize image manipulations as a three steps pipeline that navigates between two types of visual representations. Each image manipulation relies on image features identified in different level of knowledge about the visual content (pixel level, image level and scene level).*

In this manuscript, three levels of knowledge are distinguished (Figure 18): the *pixel level* corresponds to the recorded color values, the *image level* encodes images features such as edges or color regions, finally the *scene level* contains information about the 3D scene like surface depth, orientation, reflectance or illumination. The definition of these levels of knowledge is inspired by theories of perception and computer vision that model the human visual system as a succession of stages. These stages separate the visual information in variations due to geometry, reflectance, illumination and viewpoint [BT78, Mar82, Pal99]. Given this new "knowledge" dimension, an image manipulation can be expressed as the transformation of an image from one position in the space of visual representations to another position in this space, using features of the image lying in one specific level of image understanding. Each image manipulation in this manuscript is illustrated based on this definition. The manipulations differ in their input (photographs, videos, 3D scenes, drawing) and their goal (stylization, simplification, enrichment).

This dissertation is structured around the three types of images we manipulate. The first part is dedicated to simplified images, the second part to stylized images and finally the third part to realistic images. The image manipulations instanced in each part sample the major areas of the space of visual representations. The proposed methods facilitate the creation of a variety of images, and contribute to some of the main challenges of each area, as detailed in the following paragraphs.

Simplified Images From a computer graphics perspective, the main challenge of image simplification is to identify the main structures in an image and discriminate them from details. Chapter 1 aims at simplifying shapes in a video without introducing flickering and popping. Based on the assumption that shape details correspond to small geometrical features in an image, we show that **shape simplification of videos can be performed with simple low level image filters**, without requiring the complex shape extraction of previous methods. These filters are extended to the temporal domain to maintain the stability of the animation.

In addition to this geometrical approach to detail removal, Chapter 2 proposes a perceptually based definition of details that allows image simplification without a priori knowledge on the image content. The main intuition behind the proposed method is to **consider edges as a powerful and flexible encoding of the image structures**, and to estimate the importance of each edge in an image based on a model of visual perception. This approach is motivated by the fact that edges encode most of the visual information, and that the human visual system is very sensitive to contrast variations [Pal99].

Based on the same edge structure, Chapter 3 introduces a novel vector primitive that facilitates the enrichment of simple line drawings with complex color variations. This allows the depiction of realistic image features in a vectorial form, which remains a challenge for most existing vector primitives. Moreover, this edge-based primitive is a natural drawing tool because artists are used to represent the main structures of an image as a line drawing before colorizing it. Finally, this approach allows the automatic vectorization of photographs.

These three image manipulations provide a continuum in the navigation between complex and simple images.

Stylized Images Two methods for stylizing videos and 3D animations are introduced. Both approaches address the problem of *temporal coherence* that remains one of the main challenge of Non Photorealistic Rendering. Temporal incoherence traditionally occurs in stylized animations because of random phenomenons that are inherent to many styles and vary from one frame to another. These phenomenons include variations of paper grain and pigment density in watercolor, of stroke size in oil painting, or pencil width in line drawing. Although the popping and flickering produced by these random variations can be appreciated as part of the style (see for example the short movies of *Bill Plympton*), they can quickly become disturbing for the observer. Above all, artists have almost no control on these side effects. In computer generated animations, style marks such as pigments or brush strokes are not placed randomly on each frame, but animated from one frame to another. However, this raises an inherent contradiction: style marks should follow the *3D* motion of the objects they depict but preserve their *2D* appearance.

The common claim behind the two approaches presented in this dissertation is that **resolving this contradiction locally in time is sufficient for a convincing illusion of temporal coherence**. With a limited temporal support, the stylization follows the 3D motion during enough frames to produce an accurate perception of movement, but is smoothly regenerated to avoid deviation from a 2D appearance. We apply this principle to the two algorithms we propose, each of which takes advantage of the specific input data to stylize. For 3D scenes, Chapter 5 introduces

a simple and fast *infinite zoom mechanism* that produces stylization textures with a constant size in screen space. This technique conceals most of the 3D clues induced by perspective projection, and is very well suited to real time applications such as video games. For videos, Chapter 6 presents a novel *texture advection scheme* that minimizes texture distortion when animating a stylization texture along lines of optical flow. The resulting stylization preserves a strong 2D appearance at any frame.

Realistic Images In the area of realistic image editing, Chapter 7 proposes a novel method to estimate the illumination and reflectance of a scene from a single photograph. Estimating this information is an under-constrained problem and automatic methods are challenged by complex natural scenes. We show that this problem can be made tractable via a combination of **user indications** and simple assumptions on the **color distributions of natural images**. The resulting additional knowledge about the depicted scene allows advanced editing such as re-lighting and re-texturing.

The three parts of this manuscript show a variety of navigations between the three main areas of the space of visual representations. However many other image manipulations could be proposed to cover this space. In our conclusion we present a description of a few potential research directions that we plan to explore in the future.

Part II

Image Simplification and Enrichment

We investigate in this part the navigation from a complex photograph to a simple illustration, and from a simple line drawing to a complex colorized image.

We first introduce two methods for removing details from an image. These two methods have been developed with a different goal in mind. In the first method, small features of a video are removed to mimic the simplification observed in watercolor painting. In that case, the simplification reflects a technical constraint of the image formation process: artists often use a coarse brush that prevents the drawing of thin features, especially with the additional difficulty induced by the fluid nature of watercolor. The main contribution of our approach resides in the extension of simple image filters to the temporal domain to allow temporally coherent simplification of shapes. The second simplification we propose relies on a model of the low level human visual system to identify the relevant structures of an image, along with their importance. We use this importance measure to drive various image manipulations such as detail removal that focus the attention of the observer on areas specified by the user.

In the third chapter of this part we propose a novel vector primitive, called *diffusion curve*, that allows a user to enrich line drawings with rich color gradients. Diffusion curves facilitate the creation of artworks with complex features such as soft shading or defocus blur, that are hard to produce with existing vector drawing tools. This primitive is motivated by the traditional “sketch and colorize” workflow of many artists.

chapter 1

Temporally Coherent Detail Removal for Video Simplification

The work presented in this chapter and in Chapter 6 was done during an internship at Adobe Advanced Technology Labs in Seattle in 2006. This research has been published as part of a SIGGRAPH 2007 paper [BNTS07], in collaboration with Fabrice Neyret, Joëlle Thollot and David Salesin.

In this chapter we describe a simplification filter that removes thin features in an video. We use this simplification as part of watercolor stylization of videos. One of the major characteristics of watercolor is the medium's ability to suggest detail with abstract washes of color. Examples of real watercolor paintings in Figure 1.1 illustrates how subtle details, such as faces of the characters or tree leaves, are only suggested through abstraction. In this context, "detail" correspond to thin shape features that could not be painted using a coarse brush. Additional watercolor effects such as distinctive pigment and paper texture will be described and reproduced in part III.

Figure 1.1: *Real watercolor, such as these paintings by Max Cabanes©, often exhibits a limited amount of detail.*

We examine how raw video footage can be abstracted into shapes that are more characteristic of watercolor renderings, as well as temporally coherent. We extend mathematical morphology to the temporal domain, using filters whose temporal extents are locally controlled by the degree of distortions in the optical flow. We show that morphological filters effectively simplify shapes without requiring the explicit shape extraction of previous methods. Figure 1.2 visualizes this simplification in our 2D space of visual representation. The operation relies on features at two different levels of knowledge: the spatiotemporal filter in the pixel level, and the optical flow in the scene level.

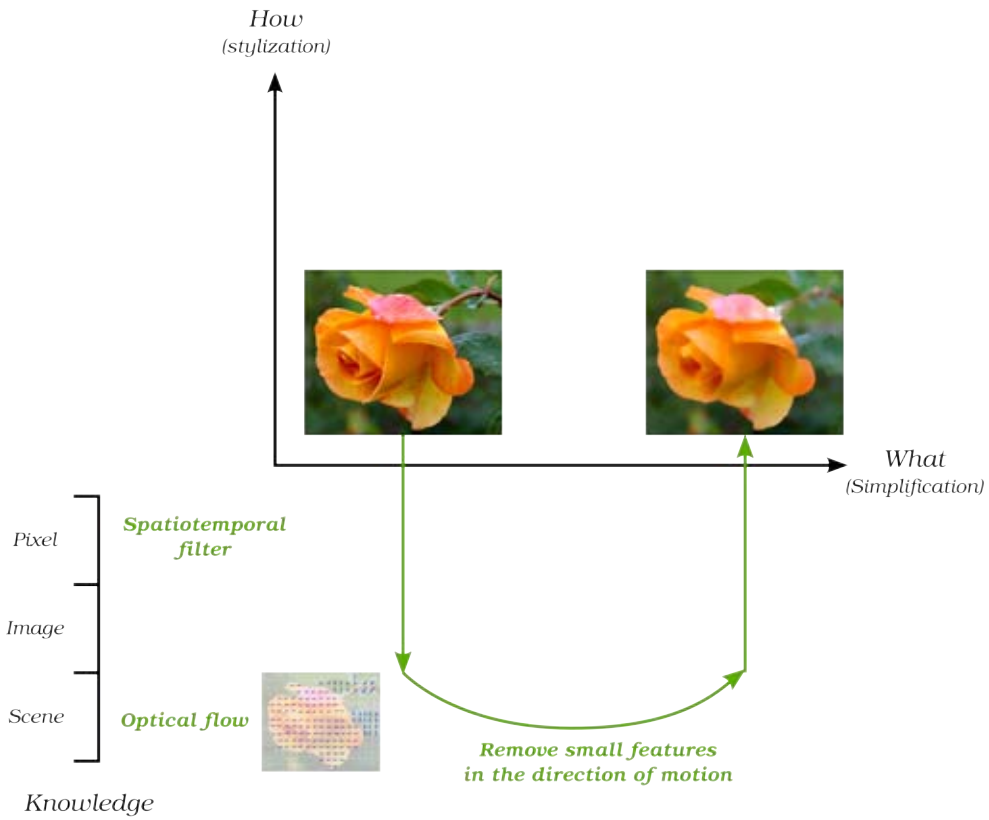


Figure 1.2: *We create a simplified video from a complex input. Small features are removed according to the pixel's neighborhood. We use optical flow estimation to orient the filter in the temporal domain and ensure temporal coherence.*

1 Related Work

A significant body of research has been concerned with the issue of abstraction of video. Winnemöller et al. [WOG06] presented a method to smooth a video using a bilateral filter, which reduces the contrast in low-contrast regions while preserving sharp edges. We can use a similar approach to produce large uniform color areas, and we go a step further in simplifying not just

the color information but the 2D scene geometry as well. Kang and Lee [KL08] describe how to remove image details and simplify shapes using the mean curvature flow, but it is unclear how such an approach behaves on image sequences. In *Video Tooning*, Wang et al. [WXSC04] use a mean-shift operator on the 3D video data to cluster colors in space and time. To smooth geometric details, they employ a polyhedral reconstruction of the 3D clusters followed by mesh smoothing to obtain coarser shapes. Collomosse et al. [CRH05] use a similar type of geometric smoothing involving fitting continuous surfaces to voxel objects. Such high-level operations are usually computationally expensive and may sometimes require user input to produce convincing results. Our approach, by contract, uses simple low-level image processing for geometric as well as color simplification. In essence, we build upon 2D morphological filters [SV92] to abstract the shapes and mimic the characteristic roundness of watercolor. By extending a morphological filter to the 3D spatiotemporal volume, we obtain a temporally coherent abstraction filter.

Many image filtering operations have been extended to the spatiotemporal domain to process video: the median filter [AHJ+90], average filter [OST93], and Wiener filter [Kok98] are all examples. Moving objects describe slanted trajectories in the temporal domain, which leads to ghosting if an axis-aligned filter is applied. A motion compensation is usually applied before filtering to avoid such ghosting artifacts in regions of high motion. These filters have been developed in the context of noise removal, which requires kernels of only small support. Our application is more extreme in that it targets the removal of significant image features, potentially several time larger than a single pixel. To avoid "popping" artifacts, we propose a type of adaptive kernel that smoothes the appearance and disappearance of significant image features.

2 2D Morphological Operators

Real watercolor are typically composed of large color areas with few thin structures. We propose to remove thin structures by applying morphological filters. Such filters have long been described as having the ability to "simplify image data, preserving their essential shape characteristics and eliminating irrelevancies" [HSZ87].

Morphological operators for gray-level images are defined as follow. Let I be an image and B a *structuring element*, that is an array that contains the relative coordinates of a pixel's neighborhood. The morphological *dilation* δ of I by B at a pixel $\mathbf{x}$ and its dual, the morphological *erosion* ε, are defined as:

$$\delta_B(I)(\mathbf{x}) = \max_{b\in B}\{I(\mathbf{x}-b)\} \quad \varepsilon_B(I)(\mathbf{x}) = \min_{b\in B}\{I(\mathbf{x}+b)\}$$

The dilation spreads the light features of the image whereas the erosion spreads the dark features (see Figure 1.3(b,c)). The morphological *opening* is then defined as a sequence of one erosion followed by one dilation, $\delta_B \circ \varepsilon_B(I)$, and the morphological *closing* as one dilation followed by one erosion $\varepsilon_B \circ \delta_B(I)$. Opening removes light features of the image (Figure 1.3(d)), whereas closing removes dark features (Figure 1.3(e)). We apply a sequence of one closing followed by one opening to remove both the small dark and bright details of the image (Figure 1.3(f)). For

further details on morphological filtering, see the overview of Serra and Vincent [SV92] or the work of Haralick et al. [HSZ87].

Figure 1.3: *Mathematical morphology operators. We apply one closing followed by one opening to remove the small details of the image (f).*

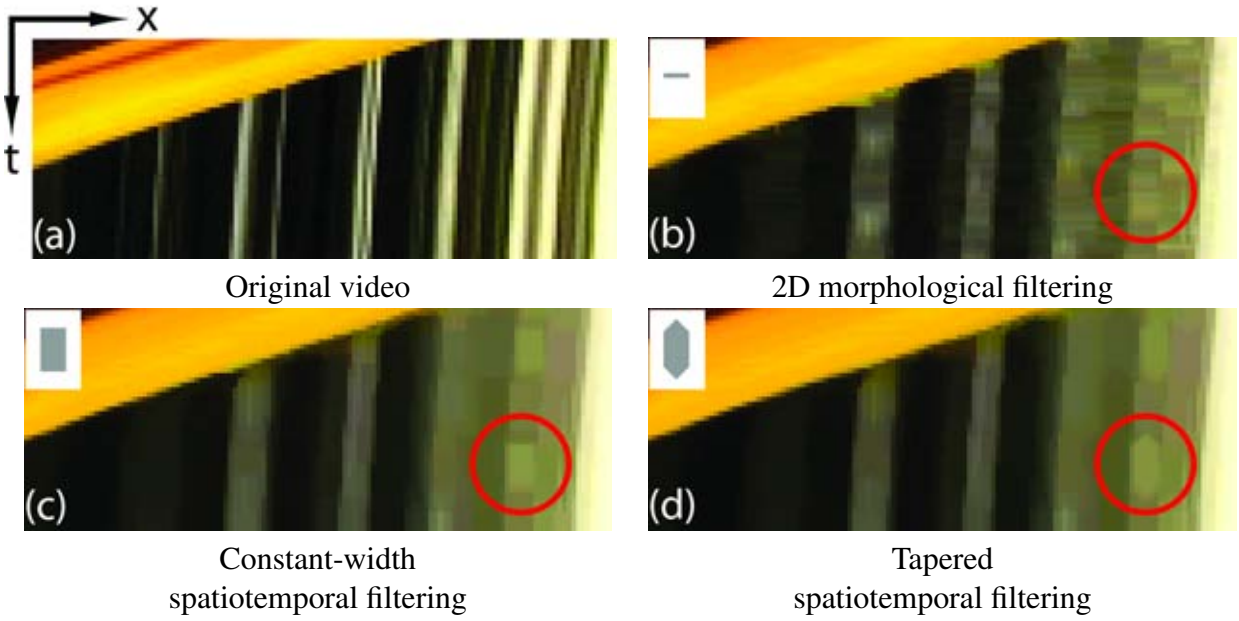
The size of the morphological structuring element defines the size of the smallest elements that will be preserved by the filtering. This size can be seen as an analogous to the size of the paint brush used in real painting. The shape of the structuring element defines the shape of the filtered objects in the resulting image. By applying a disk-shaped morphological filter we obtain rounded shapes in the image, very similar to the one encountered in real watercolors (Figure 1.3(f)).

For simplicity, the operators are applied on the three color channels separately. Although this independent filtering of the channels produces color ghosting on dilation and erosion (Figure 1.3(b,c)), it becomes unnoticeable when these dual operators are applied in sequence (Figure 1.3(d,e,f)).

3 Spatiotemporal Morphological Operators

Applying morphological filtering on each frame of a video sequence separately produces a great deal of flickering, as many features appear and disappear at every frame (See Figure 1.4-(b)). Moreover, as every feature is at least as large as the 2D structuring element, the features' appearances and disappearances produce noticeable "popping" artifacts. To reduce temporal artifacts,

we extend the structuring element to the temporal domain. The resulting 3D structuring element can be thought of as a stack of neighborhoods in k successive frames. The 3D element reduces flickering as it guarantees that each new feature remains visible during all k frames. However, care must be taken to correctly orient the filter. Indeed, moving objects correspond to spatiotemporal regions that are not necessarily perpendicular to the time axis. As in previous work [OST93, Kok98], we compensate for this motion by translating the neighborhood at each pixel according to the optical flow for each frame. We extract the optical flow field of the video using a classical gradient-based method available in Adobe After Effects. The resulting filtering produces uniform temporal regions, resulting in a higher temporal coherence (See Figure 1.4-(c)).



Original video | 2D morphological filtering

Constant-width spatiotemporal filtering | Tapered spatiotemporal filtering

Figure 1.4: *A visualization of the effects of various filters on the spatiotemporal volume (filter represented in gray). These figures show a portion of a single horizontal scanline of video as a function of time, which appears on the vertical axis. A 2D morphological filter (b) results in vertical noise, which corresponds to flickering in the filtered sequence. A constant-width spatiotemporal filter (c) produces sudden onset of features, corresponding to popping in the filtered sequence. With our tapered filter (d), features appear and disappear gradually rather than suddenly.*

Unlike previous methods in video denoising, which used small filter kernels (usually $3 \times 3 \times 3$ pixels), we would like to use kernels with much larger spatiotemporal support (generally, $7 \times 7 \times 9$ pixels) to abstract away significant details. In order to reduce the popping produced by the abrupt appearance or disappearance of these details, we design the structuring element in such a way that visibility events occur gradually. As such, we taper the structuring element at its extremities (see Figure 1.5). The features in the resulting video appear and disappear gradually. Their shapes in the spatiotemporal volume, visualized in Figure 1.4(d), mirror precisely the shape

of the tapered 3D structuring element that we use.

Figure 1.5: *A visualization of how our tapered morphological filters follow the optical flow from frame to frame.*

Finally, proper spatiotemporal filtering assumes that every pixel has a correct motion trajectory throughout the temporal support of each filter. In practice, this assumption fails for several reasons: First, a pixel can become occluded by another object in the video. Second, optical flow is usually just an approximate estimation of the motion flow, especially in disocclusion areas where no real correspondence can be found between two successive frames. A common approach to increase the robustness for spatiotemporal filtering in such cases is to make the filter adaptive [OST93]: the idea is to ignore the contribution of outliers in the filtered value. However, a direct application of this method to morphological operators would create holes in the structuring element, which would have a direct impact on the shapes in the resulting image. Instead, we simply truncate the structuring element in the temporal domain as soon as an "outlier" is detected. A pixel is flagged as an outlier when its estimated motion accuracy is below a defined threshold. Similarly to Chuang et al. [CAC+02], we estimate the accuracy of our optical flow assuming a constant intensity along motion trajectories. This is done by comparing the mean color on a 3×3 neighborhood of a pixel between two frames. In practice we use the L_2 norm of the distance in RGB.

4 Results and Discussion

Figure 1.6 shows the result of our approach on one frame of a video with complex motion boundaries. We have implemented our method as a non optimized Adobe After Effects plug-in. Morphological filtering operations are notoriously expensive, and in our current implementation the spatiotemporal filter takes about 30 seconds per frame. However, some of the clever implementation strategies devised for 2D processing [vDT96, Wei06] may be generalizable to 3D, which could greatly improve the performance.

Input video Simplified video

Figure 1.6: *Result of the morphological simplification on one frame of a video. Note that occlusion boundaries along the silhouette are correctly handled.*

Our filtering is stable over time and offers large color regions that appear and disappear gradually over the course of the animation. All the examples presented in our videos[1] have been abstracted with a $7 \times 7 \times 9$ structuring element. Increasing the spatial extent of the structuring element generally requires increasing its temporal extent as well, in order to allow time for the width of the structuring element to increase and decrease gradually.

Our choice of using morphological filters is motivated by our definition of "detail" in the context of watercolor rendering. We define detail as thin features, that could not be drawn with a coarse brush (modeled by the structuring element). This definition differs from the notion of detail in terms of bilateral filtering used in existing methods [WOG06], that smoothes out features of small contrast. We believe that the choice of a particular simplification filter depends on the task at hand. For applications such as contrast reduction or enhancement, a pure contrast-based approach is valid and makes the bilateral filter or related methods good candidates [DD02, FAR07]. On the other hand, in the context of stylization, our shape simplification approach allows the removal of small highly contrasted features that cannot be removed by the bilateral filter (Figure 1.7). Nothing prevents the use of both filters to first create large smooth color areas and then remove the remaining small details. Note however that while the rounded shapes produced by morphological filters is an attractive side effect for watercolor rendering, it can be seen as a limitation if a shape preserving approach is required.

5 Conclusions

In this chapter we have presented a simplification method that only relies on local observations in the image to discard small features. By extending 2D filters to the temporal domain, our approach abstracts videos without introducing flickering and popping. Our definition of details

[1] Videos are available on the project webpage: http://artis.imag.fr/Publications/2007/BNTS07/

(a) Original image (b) Bilateral filter

(a) Morphological filter (b) Bilateral filter, followed by morphological filter

Figure 1.7: *While a bilateral filter removes details based on contrast (b), a morphological filter removes details based on geometry (c). As a result, a morphological filter is able to remove highly contrasted features, but also simplifies shapes. For watercolor stylization, a combination of a bilateral filter followed by a morphological filter create large smooth color areas with few details (d)*

as small features mimics the use of a coarse brush in watercolor rendering. Note however that this definition is purely geometric and does not take into account the way humans perceive details in an image. We explore in the next chapter a different definition of details, based on a model of low-level visual perception. This model allows the estimation of a perceptually motivated measure of importance on the image structures. This measure offers fine control on the amount of visual information, while preserving the perceptually important shapes of the original image.

chapter 2

Structure-Preserving Simplification of Photographs

The research described in this chapter is the result of a collaboration with Alexandrina Orzan, Pascal Barla and Joëlle Thollot. The work of implementation was equally shared between Alexandrina Orzan and myself. This work has been published at NPAR 2007 [OBBT07].

As discussed in Chapter I, effective visual communication is not always best achieved by the "real-world like" images. Simplified objects or exaggerated features can often improve perception and facilitate comprehension by grabbing visual attention. In this chapter, we offer a tool for creating such enhanced representations of photographs in a way consistent with the original image content. Compared to previous chapter that only relies on a geometric definition of detail to simplify shapes, this chapter explores the discrimination of detail from structure based on a model of low level human perception.

Taking a look at Figure 2.1 that represents a hand-made scientific illustration, it is clear that the main subject of the image is the butterfly: it is depicted with many details, while plants around are more or less suggested. However, while abstracted, secondary elements of the image retain their look and are easily identified; in other words, their relevant *structural* components are preserved through the abstraction process.

The main goal of this chapter is to give insights into "what structure means" and to provide the user with image manipulation tools to create enhanced or abstracted representations of photographs in accordance with their structural information. To do so, we develop a method to identify the relevant image structures and their importance. We define *edges* at multiple scales as the basic structural unit, which is motivated by the fact that most color variations in an image can be assumed to result in edges (material and depth discontinuities, texture edges, shadow boundaries, etc.) [KD79, MH80]. Based on this observation, it has been demonstrated that edges constitute a near-complete and natural primitive to encode and edit images [Car88, Eld99, EG01]. In this work, we rely on scale space theory to define relevant structures as edges that persist at

Figure 2.1: *"Le Papillon" (The Butterfly), watercolor by Eric Alibert. From "Leman, mon ile", © 2000 by Editions Slatkin. As seen in the guidebook of scientific illustration [Hod03]. The detailed butterfly over a simplified background attracts attention.*

multiple scales. Edges together with their importance form a hierarchy of structures in the image level (Figure 2.2). This hierarchy can be easily manipulated by the user to reflect what is important to her (Figure 2.3). The final image is then rendered from the "cropped" gradient information using Poisson reconstruction.

This edge-based approach to image processing is made feasible by two new techniques we introduce: an addition to the Gaussian scale space theory to compute a perceptually relevant hierarchy of structures, and a contrast estimation method necessary for faithful gradient-based reconstructions. We present various applications that manipulate image structure in different ways, as illustrated in Figure 2.3.

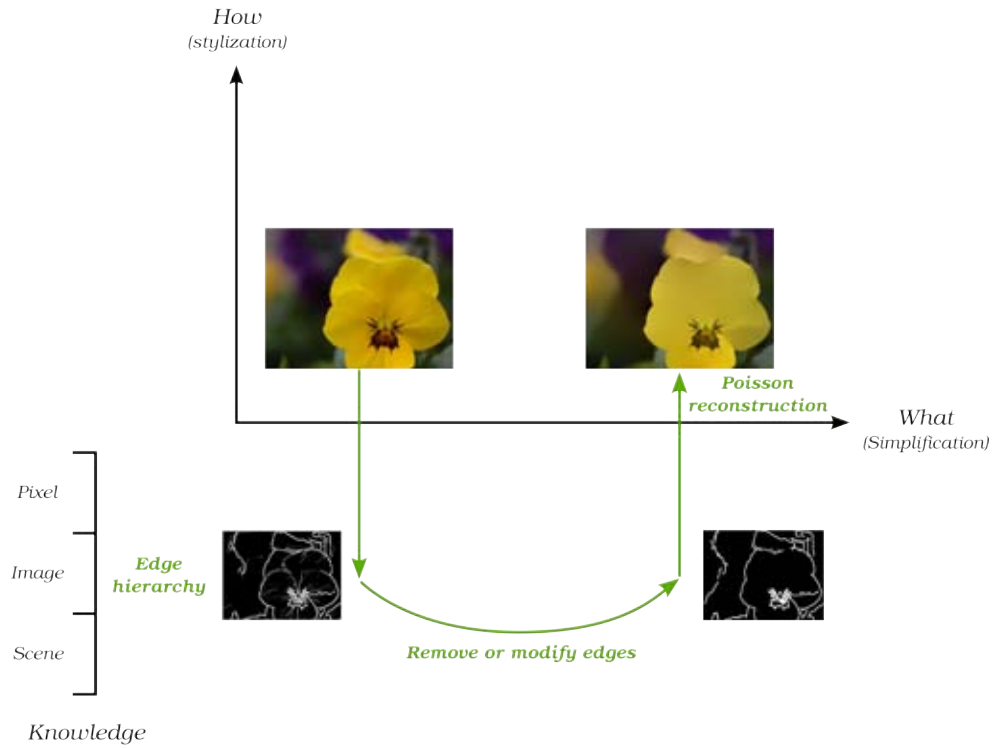


Figure 2.2: *Our approach manipulates edges in the image level to create simplified representations. The final image is reconstructed from the manipulated edges by solving a Poisson equation.*

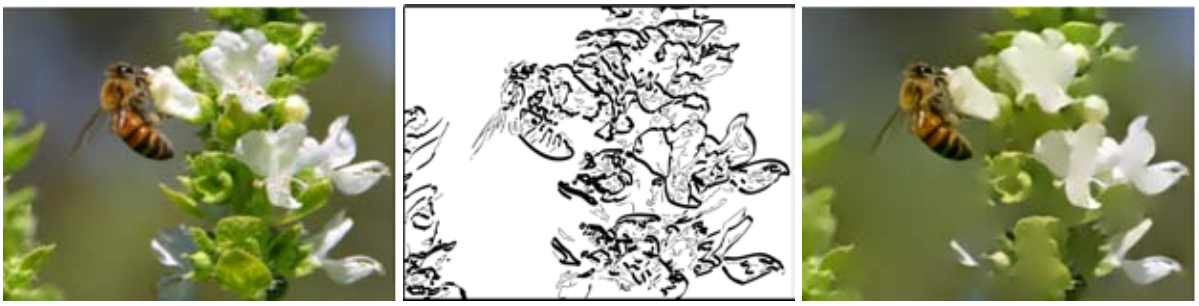


Figure 2.3: *Our approach takes as input an image (left), and allows a user to manipulate its structure in order to create abstracted or enhanced output images. Here we show a line drawing with line thickness proportional to their structural importance (middle), and a reconstruction of color information that focuses on the bee and removes detail around it (right). Bee image from www.pdphoto.org.*

1 Related Work

A number of previous techniques focused on creating enhanced or abstracted renderings from arbitrary photographs. Most of the previous methods manipulate an image globally without using the image structure [WOG06], or rely on the user to define what is important [WXSC04, KCC06, WLL+06]. As a result, the content either cannot be controlled, or its control involves tedious user interactions. We are interested in automatically extracting the relevant structural information to enrich automatic systems or assist the user in her task.

Previous work made use of Gaussian scale space [Her98] or saliency maps [CH05, CH03] in order to guide painterly stylizations. However, saliency maps identify image regions that already grab visual attention in the original image, and using them to guide stylization will only preserve these attention-grabbing regions. In contrast, our goal is to extract a structure that allows the user to *intentionally* manipulate the image, possibly modifying its attention focus (i.e. changing its subject, see Figure 2.3 - right), and hence conveying a particular message.

DeCarlo and Santella [DS02] were the first to use a visual structure in photo abstraction. They use color regions as structural units and create their hierarchy of regions from a pyramid of down-sampled versions of the image. Bangham et al. [BGH03] extend DeCarlo and Santella's work by improving the region segmentation. Their region hierarchy is based on a morphological scale-space that is designed to preserve region shapes. But since only the region size is considered, and not its contrast, they tend to eliminate visually important cues that have a high contrast but small size. In general, multi-scale region approaches have the inconvenience of associating a solid color to each region, which results in a posterization of the final rendering. In contrast, our edge-based structural hierarchy allows us to avoid the problems generated by region-based methods. Our edges are not required to be closed contours, as opposed to region boundaries, and hence they do not create erroneous color discontinuities.

Edge representations of images have been used in previous work, although not with the same purpose. Elder et al. [EG01] use the edge domain to ease image editing operations (crop, delete, paste), but have no concept of edge importance. Perez et al. [PGB03] suggest using gradient information only at edge locations as input for a Poisson solver, in order to obtain a texture flattening effect. We improve on this method with the aim of manipulating an image for abstraction and/or enhancement purposes by (a) giving insights into how image structure can be manipulated, and (b) by providing a new reconstruction method that extends [PGB03].

2 Background

In order to manipulate images in a structure-preserving way, our method relies on two image processing tools: Gaussian scale space and gradient domain image manipulation. We provide a quick overview of both tools and the reasons for choosing them for our purpose. Gaussian scale space will be responsible for extracting the structure of edges, while gradient domain processing will be used for reconstruction.

2.1 Gaussian scale space

Scale space methods base their approach on representing the image at multiple scales, ensuring that fine-scale structures are successively suppressed and no new elements are added (the so-called "causality property" [Koe84][1]).

The motivation for constructing scale-space representations originates from the basic fact that real-world objects are composed of different structures at different scales of observation. Hence, if no prior information is available about the image content, the state-of-the-art approach for deriving the image structure is to use the successive disappearance of scale features to create a hierarchy of structures [Rom03]. A relevant structure is defined as an element that is *invariant* to scale; other elements can be considered "accidental", and of less importance.

Gaussian scale space is the result of two different research directions: one looking for a scale-space that would fit the axiomatic basis stating that "we know nothing about the image" and the other searching for a model for the front-end human vision [FF87, Wan95, Rom03]. Since our purpose is to define a human-vision-like importance measure for an image content we have no a priori on, this scale-space fits our needs.

A scale-space is a stack of images at increasing scales. The basic Gaussian scale space is thus a stack of images convolved by Gaussian kernels of increasing standard deviation [2]. In general, Gaussian derivatives of any order can be used to build the stack, allowing the creation of scale-spaces of edges, ridges, corners, laplacians, curvatures, etc.

Edge representations, as discontinuities in image brightness, retain important data about objects in the image (shape, surface orientation, reflectance) [Lin98]. We thus settle on studying the image structures represented by a hierarchy of edges in the Gaussian scale space. As edges are defined by gradient information, we only need to convolve the original image with Gaussian derivatives of order 1, one for each image dimension. These Gaussian derivatives G_x and G_y are computed as follows

$$G_y(x,y;\sigma) = g(x)\cdot g'(y)$$

with

$$g(x) = \frac{e^{-\frac{x^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} \quad \text{and} \quad g'(y) = -\frac{e^{-\frac{y^2}{2\sigma^2}}y}{\sqrt{2\pi}\sigma^3}$$

where the standard deviation σ of the kernel corresponds to scale. Given an input image I, we thus build two different scale spaces: an horizontal gradient $I_x = I \otimes G_x$ and a vertical gradient $I_y = I \otimes G_y$.

2.2 Gradient domain image manipulation

Many recent works introduced gradient manipulations as an efficient tool for image processing. The main reason is that gradients represent image variation independently of the origi-

[1] Note that the causality property holds in 1D, but may not hold in some degenerate cases of 2D signals

[2] For numerical stability, one usually starts with a standard deviation $\sigma_0 = 1$ pixel

nal colors, allowing more flexibility in image manipulations. Handling the image variations directly makes possible applications such as seamless image editing [PGB03] and image fusion [ADA+04, RIY04]. Gradient domain is also an intuitive representation for image contrast [FLW02]. We propose to associate the flexibility of gradient domain manipulations to the high level control provided by the Gaussian scale space. This allows us to seamlessly combine information from multiple scales.

Working in the gradient domain implies one can reconstruct an image I from its gradient field $\mathbf{w} = (\mathbf{w}_x, \mathbf{w}_y)$. As a manipulated gradient is unlikely to be conservative and integrable, a common approach is to compute an estimation of the image whose gradient field best fits $\mathbf{w}$ in a least-square minimization sense:

$$\arg\min_I \int_\Omega (\nabla I - \mathbf{w})^2 d\Omega$$

This estimation corresponds to the unique solution of the Poisson equation $\Delta I = \operatorname{div} \mathbf{w}$, where Δ and div are the Laplace and divergence operators [PGB03, FLW02].

3 Our approach

We first apply Gaussian scale-space analysis to the input image I to get gradient values at multiple scales $(I_x, I_y)_\sigma$; then we manipulate this rich information in a way that preserves the structure of the image, giving rise to a gradient field $\mathbf{w} = (\mathbf{w}_x, \mathbf{w}_y)$; finally, the output image O is built from $\mathbf{w}$ using Poisson reconstruction. Figure 2.4 illustrates our approach.

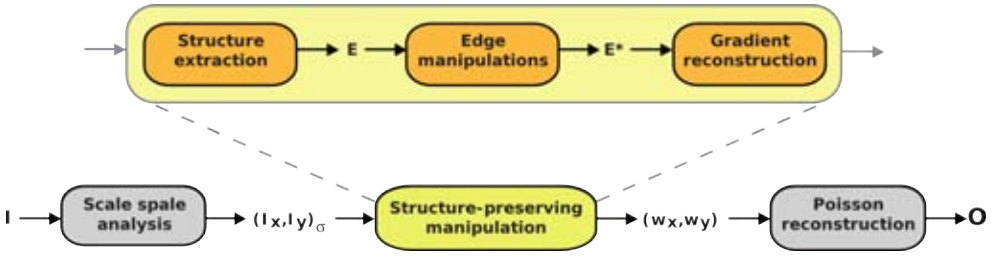


Figure 2.4: *Overview of our method.*

Our structure-preserving manipulation represents the heart of the approach and is composed of three steps:

1. **Structure extraction:** starting from the raw multi-scale gradient values, we extract the image structure S corresponding to the edges, their importance and profile.

2. **Edge manipulations:** We then use the structure S as a high-level control for user-defined image manipulations, and output a manipulated structure S^*.

3. **Gradient reconstruction:** We finally reconstruct a gradient field from the set of manipulated edges with their profile.

In the following section we mainly present the two technical steps of the method: structure extraction and gradient reconstruction. Several edge manipulation techniques are presented in Section 4.

3.1 Structure extraction

3.1.1 Edge extraction

From the first-order Gaussian derivative scale spaces, we want to build a hierarchy of edges holding structural importance. Before defining what we mean by structural importance, we first extract edges at all the available scales in order to get the richest possible information. For this task we use a Canny edge detector [Can86]: it is a state-of-the-art edge detection method that processes the Gaussian derivative information at each scale to give thin, binary edges. Its main advantage resides in using hysteresis thresholding that results in long connected paths and avoids small noisy edges (see Figure 2.5). When using a color image as input, we apply the Canny detector to the multi-channel color gradient described by Di Zenzo [Zen86].

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

Figure 2.5: *Edge importance. (a) The input image. (b-d) Canny edges at increasing scales. (e) The lifetime measure reflects the importance of edges: "older" edges correspond to more stable and important structures.*

After applying the Canny detector, we are left with a multi-scale binary mask C_σ that indicates at each scale the edges locations. Figure 2.6 illustrates such a typical edge scale-space for a simple 1D example. Due to the nature of Gaussian scale-space, three different cases can occur: (a) an edge exists and suddenly drops off at a higher scale; (b) two edges approach each other and collapse at a higher scale; (c) some "blurry" edges only appear at a higher scale. To simplify further computations, we "drag" edges corresponding to case (c) down to the minimum scale and note C^*_σ the resulting multi-scale edge mask.

3.1.2 Edge importance

As shown in Figure 2.6, there is a great deal of coherence along the scale dimension in the multi-scale edge representation. The main idea behind scale-space techniques is to try to extract

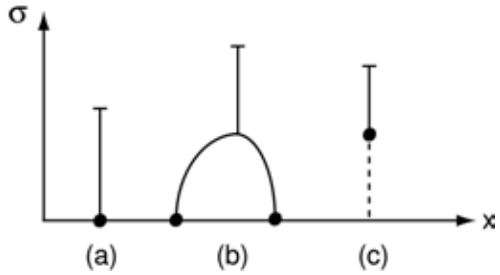


Figure 2.6: *Three different events in a 1D Gaussian scale-space: (a) an edge drops off at a high scale; (b) two edges collapse ; (c) a blurry edge is created. In the last case, we drag the edge down to the finest scale for convenience.*

this coherent *deep structure*, by linking edges at different scales. In particular, because of the causality property of Gaussian scale-space, an edge that disappears at a given scale will not reappear at a higher scale; hence an important measure of structure along scale is *lifetime*, as edges that live longer will correspond to more important structures.

Unfortunately, extracting an edge lifetime is not trivial, since edges move in Gaussian scale-space (this corresponds to case (b) above). This motivated edge focusing techniques, that track edges at increasing scales. We take an alternative approach which revealed simpler to implement: instead of considering each pixel p belonging to an edge, we consider its projected point $\mathcal{P}_\sigma(p)$ onto the closest edge at scale σ (we use a distance field for this purpose). We can then define the membership of any pixel $m_\sigma(p)$ as the binary function that indicates whether p can be considered to belong to an edge at scale σ:

$$m_\sigma(p) = \begin{cases} 1 \text{ if } ||\mathcal{P}_\sigma(p) - p|| < T_\sigma \\ 0 \text{ otherwise} \end{cases}$$

The choice of the threshold distance T_σ is essential to get a good approximation for our membership function. Bergholm [Ber87] proved that the edge shifting is less than a pixel when the scale σ varies by less than 0.5. Therefore, we increase our σ values by $\Delta\sigma = 0.4$ at each scale and use $T_\sigma = \sigma/\Delta\sigma$. This approach is similar in spirit to the morphological linking method of Papari et al. [PCPN07].

Finally, using membership for linking purpose, we compute the lifetime $L(p)$ at each edge pixel p in the finest scale by summing up membership values. Considering the successive scale values $\sigma_i, i \in 1..N$, where N is the size of our scale-space stack, we write lifetime as:

$$L(p) = \arg\min_i \{\sigma_i | m_{\sigma_i}(p) = 0\}$$

This can be seen as a simpler version of Lindeberg edge strength measure [Lin98] that requires the extraction of edge surfaces in scale space. We can now use lifetime as a measure of structural importance to manipulate edges in a structure-preserving way, as shown in Section 4.

3.1.3 Edge profile

In the previous section, we mainly relied on edge locations and their persistence along scale. Another concern is to deal with their *profile* (contrast value and degree of blur) . Similarly to previous work [Lin98, EG01], we rely on a simple assumption: an edge profile is modeled as the convolution of a Dirac (its location and contrast) with a spatially varying Gaussian kernel (its blur). For instance, in a photograph with depth-of-field, out-of-focus edges are blurry (with a wide profile) while in-focus edges are sharp (with a thin profile).

Our second measure of structure then consists, for each edge, in finding the *best scale* that locally corresponds to its blur. This is also the scale where we measure the contrast.

The best scale search is another form of *deep structure* that has been studied by Lindeberg [Lin98]. Following his approach, we first compute a normalized gradient magnitude scale-space by $||\nabla I|| = \sqrt{\sigma(I_x^2 + I_y^2)}$. The best scale $B(p)$ at an edge pixel p is then identified as the one which gives the first local maxima along the scale axis in this normalized gradient magnitude stack. But as with lifetime computation, we need to link "moving edges" at different scales using the projection operator $\mathcal{P}_\sigma$ again: $||\nabla I(p)|| = ||\nabla I(\mathcal{P}_{\sigma_i}(p))||$. Figure 2.7 shows how best scales can be well estimated for edges of increasing blur.

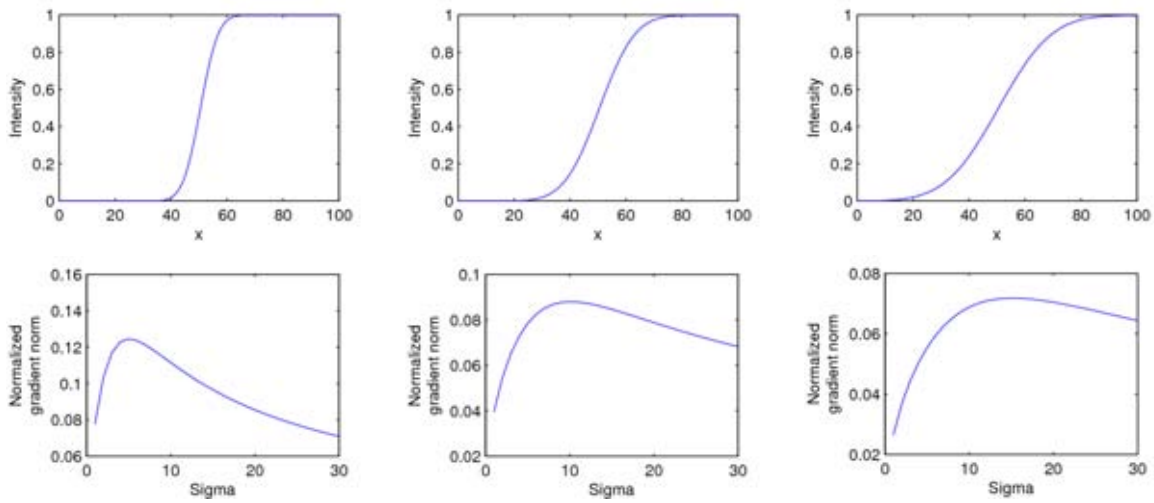


Figure 2.7: *Edges profile. Top: 1D edges blurred with* $\{\sigma_i\} = \{5, 10, 15\}$. *Bottom: normalized gradient magnitude scale space proposed by Lindeberg. The best-scale measures (the local maxima) are at the* $\{\sigma_i\}$ *used for blurring, hence they represent well each edge profile.*

We are now able to "re-blur" the edges using the best scale. Moreover, we will also make use of this measure to find a correct contrast in order to get edge profiles back into the output image.

3.2 Edge manipulations

The multi-scale Canny edges, together with their lifetime and best scale finally constitute the structure $S = \{C^*_\sigma, L, B\}$ we extracted from the input image. This structure can be manipulated in various ways depending on the application (see Section 4). The main idea is to select a subset E of the multi-scale Canny edges C^*_σ according to lifetime L. After manipulation, we are thus left with a new, simpler structure $S^* = \{E, B\}$.

3.3 Gradient reconstruction

We wish to reconstruct the corresponding image by solving a Poisson equation, i.e. we want to build a vector field $\mathbf{w}$ that corresponds to our new edges.

We propose to use the scale space information to estimate the original gradient profiles and correctly reproduce the contrast and blur of the input image. However, taking the original gradient values at edge locations as suggested by Perez et al. [PGB03] results in a gradient field that does not capture the whole original contrast, nor the original blur (Figure 2.8, (a) and (b)). This is because we only consider the central value of the profile, loosing all its surrounding information.

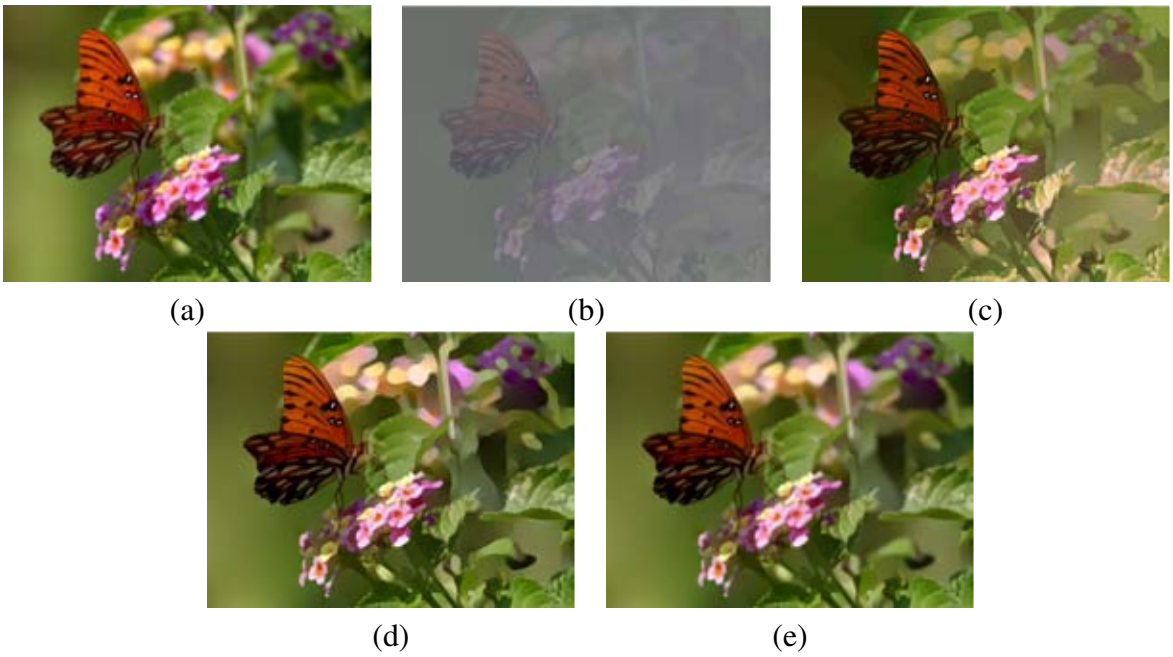


Figure 2.8: *Gradient reconstruction. (a) Input image. (b) Reconstructed image using only the original gradient values at edge positions. (c) Reconstructed image with histogram equalization. Note the quantization artifacts. (d) Reconstructed image using contrast correction. Note that blurry edges become sharp if the profile is not taken into account. (e) Full reconstruction using contrast correction and re-blurring. Butterfly image from www.pdphoto.org.*

A simple solution to the contrast problem would be to apply a histogram equalization on the reconstructed image to match the original contrast. However the very low dynamic range of the reconstructed image leads to strong quantization artifacts (Figure 2.8(c)).

We thus need to take into account our knowledge of edge profiles to compute the correct contrast. Our model of an edge represents blurry edges that appear in the input image I as the convolution of a step function H by a 2D Gaussian kernel G_B, where B is the local best scale. When we measure I_x (resp. I_y) at scale B on edge locations, we get the following values:

$$I_x = H \otimes G_B \otimes \frac{\partial G_B}{\partial x} = H \otimes \frac{\partial G_{B_2}}{\partial x} = \frac{\partial H}{\partial x} \otimes G_{B_2}$$

with $B_2 = \sqrt{2B^2}$. However, to recover the original contrast, we are precisely interested in the value of $\frac{\partial H}{\partial x}$. This corresponds to the deconvolution of I_x (resp. I_y) by G_{B_2}. Unfortunately, deconvolution is known as an ill-posed problem, particularly sensitive to noise and quantization [Rom03]. To avoid this problem, we propose to simplify our model for the sake of contrast correction: we replace the 2D Gaussian derivative by a 1D Gaussian derivative $\widetilde{G}_x = g'(x)$. This way, we can derive an analytical solution for the correction problem: we model a directional edge gradient $I_{\{x,y\}}$ as the 1D convolution of a step function H of amplitude A by a gaussian kernel g_σ and a gaussian derivative g'_σ, resulting in:

$$\begin{aligned} I_x(0) &= \left(H \otimes g_\sigma \otimes g'_\sigma\right)(0) = \left(H \otimes g'_{\sqrt{2\sigma^2}}\right)(0) \\ &= \int_{-\infty}^{+\infty} H(t) \cdot g'_{\sqrt{2\sigma^2}}(-t)dt = \int_0^{+\infty} A \cdot g'_{\sqrt{2\sigma^2}}(-t)dt \\ &= A \cdot g_{\sqrt{2\sigma^2}}(0) = \frac{A}{2\sigma\sqrt{\pi}} \end{aligned}$$

As a result, for each edge pixel p, we only need to multiply the gradient value found in I_x (resp. I_y) by $2B(p)\sqrt{\pi}$. This correction gives a final contrast close to the original one, and we find that our approximation works well in practice, with no visible artifacts (see Figure 2.8(d)).

Finally, even if using edge locations and correcting their contrast does give a convincing result, blurry edges become sharp in the reconstructed image. Therefore, we also re-blur the edges, as seen in Figure 2.8(e). This process remains optional as the sharp result provides an interesting cartoon style.

4 Applications

Most of the image manipulations presented in this section can be seen as variations of recently proposed methods that take advantage of the flexibility of the gradient domain. Our contribution is to use the high-level structural information provided by our approach to guide these gradient manipulations.

4.1 Detail removal

We use the lifetime information as a threshold value to simplify the image by seamlessly removing details, while preserving important structures. Such image editing operations are similar to the seamless cut and paste operations proposed by Perez et al. [PGB03] and Elder et al. [EG01], except that we provide a high level control to the user, who has only to select the desired level of detail (Figure 2.9).

Figure 2.9: *Detail removal: (a) original image, and (b-d) several levels-of-detail automatically generated by our method.*

4.2 Multi-scale shape abstraction

We propose a shape abstraction method that adapts the level of abstraction to the scale of the features in order to preserve the informative content of the picture. In practice, we select for each edge its last available version in the scale space using lifetime. As shapes become more and more smoothed along scales due to the Gaussian filter, relevant structures will have increasingly rounded shapes while details will keep their original silhouettes.

In opposition to previous approaches [DS02] that remove texture details and abstract shapes at the same time, our approach selects for each edge (including edges belonging to texture details or other small elements) the shape of its last scale. Hence, our approach still keeps most of the relevant structural information, while simplifying its shape, as seen in Figure 2.10.

This application can be seen as a fusion of multi-scale images, similar in spirit to other image fusion methods like the ones of Agarwala et al. [ADA+04] and Raskar et al. [RIY04].

(a) (b)

Figure 2.10: *Shape abstraction: (a) original image, and (b) our shape abstraction result. Notice how the thin details are kept, while shapes of bigger objects are abstracted (e.g. the poles).*

4.3 Line drawing

The edge lifetime information offers a powerful high-level parameter for any line drawing algorithm. Figure 2.11 presents the rendering of vectorized edges with a different width to enhance important structures from details. Figure 2.3 - middle also shows an example of this application.

4.4 Local control

In order to offer a local control to the user, each image manipulation can be weighted by a gray-level map indicating the desired amount of abstraction (Figure 2.3(right) and 2.12). This mechanism is essential to be able to focus on a given zone of the input image, and efficiently grabs visual attention. We take advantage of the Poisson reconstruction to obtain seamless transitions between regions of different weight.

Figure 2.11: *Vectorized edges, with a larger width for relevant structures (i.e. those having greater lifetime).*

Figure 2.12: *Local control: original image of DeCarlo et al. [DS02] and our results for two different user-specified control maps.*

5 Implementation

In our approach, we did not focus on performance, rather on how to extract and use image structure: Our current implementation[3] is in Matlab, with performance times of approximately 10 minutes for the whole process, considering an 800×600 input image and a scale-space depth of $N = 30$. However, most of this time is spent in the structure extraction, and the Poisson reconstruction takes only about 2 seconds; once structure has been computed, it can be manipulated rather efficiently.

To solve the Poisson equation on the manipulated gradient field, we use the sine transform based Poisson solver of Simchony et al. [SCS90] with Dirichlet conditions. We use the Matlab implementation provided by Agrawal et al. [4]

[3] http://artis.imag.fr/Publications/2007/OBBHT07/

[4] http://www.umiacs.umd.edu/users/aagrawal/software.html

6 Discussion

We now discuss subtle arguments related to Gaussian scale-space and Poisson reconstruction that we omitted until now for the sake of clarity.

In our exploration of deep structure, we mainly took inspiration from Lindeberg [Lin98]; indeed, he has a notion similar to lifetime, and the best scale measure is directly borrowed from his approach. One alternative for measuring the best scale is the method introduced by Elder et al. [EZ98, EG01], based on local signal-to-noise ratios. But this approach is not easy to combine with our importance measure, making Lindeberg's method better suited to our purposes. However, there is a main difference between his work and ours: we separate the importance of edges from their contrast and profile, while he deals with all this information at once. Our approach has the advantage of being easier to manipulate: one can modify any property without affecting the others.

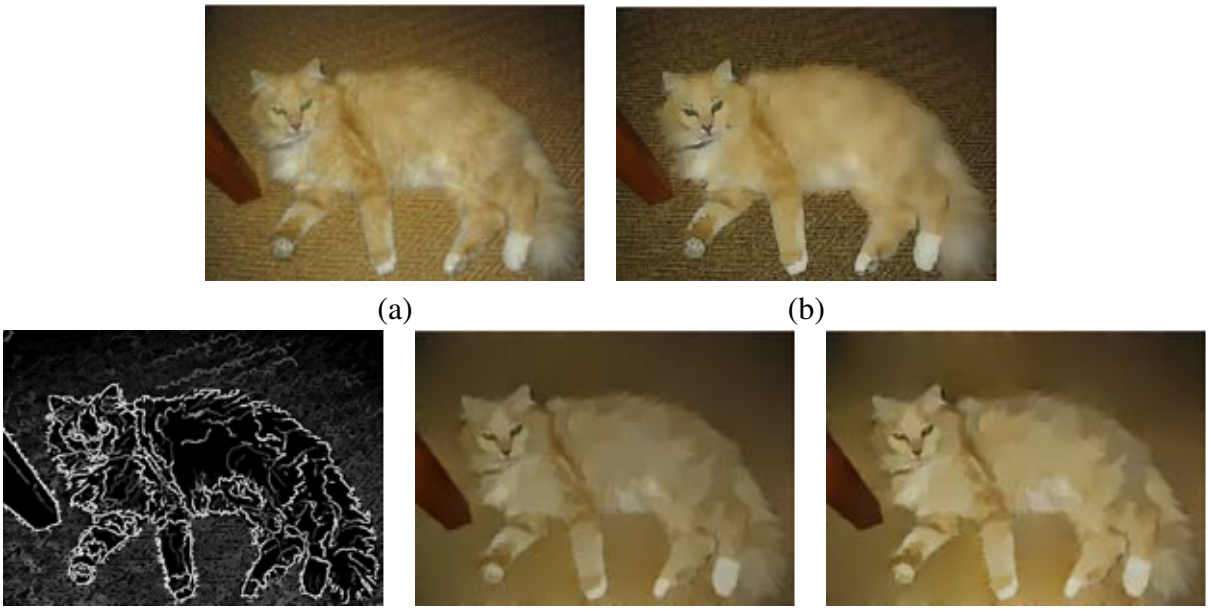


Figure 2.13: *Comparison with the failure case of Winnemöller et al. [WOG06]. (a) Original picture. (b) Winnemöller et al. abstraction failure: note how the carpet details are preserved while the fur is abstracted away. (c) Our lifetime map. (d) Our detail removal abstraction preserves the cat structure and abstract the carpet. (e) We apply histogram equalization as a post-process to fine tune contrast.*

The main difficulty when analysing the Gaussian scale space resides in the induced edge motion that requires edge focusing. Edge preserving smoothing such as anisotropic diffusion [PM90] or bilateral filtering [TM98] is often used to avoid edge motion along scale. However, these approaches favor high contrasted features over low contrasted ones independantly of their

scale. This is well illustrated in Figure 2.13: here we show a failure case of Winnemöller et al.'s abstraction approach [WOG06] that is based on a bilateral filter. Although their method gives convincing results in many cases, this specific example shows how they cannot get rid of high-contrast texture lines without abstracting the cat too far; in contrast, our approach allows us to simply remove detail edges regardless of their contrast.

(a) (b) (c)

Figure 2.14: *Comparison with the DeCarlo et al. [DS02]. (a) Original picture. (b) DeCarlo et al. results exhibit flat color regions. (c) Our result simplifies the image while keeping smooth color variations.*

Another choice we made is to use Poisson reconstruction methods. Compared to diffusion approaches, this body of techniques has the advantage of effectively removing the visual content instead of diffusing it in the image. For instance, while a diffusion method will try to blur an unwanted detail, a Poisson approach will simply ignore it in the reconstruction. This is again well illustrated by the example in Figure 2.13, since the texture lines simply do not appear in our image. Another advantage is that it gives smooth results: when compared to the stylized image of DeCarlo et al. in Figure 2.14, our method preserves smooth color variations and avoids the introduction of arbitrary color discontinuities. However, these advantages come at a cost: it is hard to reconstruct an image with a correct contrast. This is the reason why we introduced our contrast correction method. We can also perform histogram equalization as a post-process, as shown in Figure 2.13 (d) and (e). In the next chapter we show how adding color constraints on either side of the edges also contribute to a better restitution of colors.

Finally, one may wonder why we have not used Elder et al.'s contour domain approach [EG01] instead of the Poisson reconstruction. Although their approach could be used for most of the stages of our method, the fact that they need to handle colors on both side of edges makes the manipulation stage less flexible. For example, in the shape abstraction application it is unclear how the colors could be "attached" to the modified edge without additional parameterization. In the next chapter of this manuscript, we describe such a parameterization that unifies these two approaches.

7 Conclusions

We have presented in this chapter an image simplification method that defines edges as the structural unit of an image and relies on scale space analysis to discriminate between details and relevant structures.

This edge-based image representation proved to be a flexible encoding of visual information, motivated by the fact that most intensity variations occur at the discontinuities of an image. In the next chapter, we explore how a similar edge-based approach represents a powerful vector primitive that allows both automatic vectorization of bitmap images and the creation of original artworks from scratch.

chapter 3

Diffusion Curves for Smooth Vector Graphics

The work presented in this chapter was initiated during an internship at Adobe Advanced Technology Labs in Seattle in 2007. This research was done in collaboration with Alexandrina Orzan, Holger Winnemöller, Pascal Barla, Joëlle Thollot and David Salesin, and has been published at SIGGRAPH 2008 [OBW+08]. The work of implementation was equally shared between Alexandrina Orzan and myself.

We have shown in chapter 2 how edges provide a powerful representation of an image structure. Based on this observation, we describe in this chapter a novel edge-based vector primitive, called *Diffusion Curve*, that facilitates the creation of complex color gradients in vector graphics. This novel representation retains the editability of vector graphics while increasing their realism. In addition to the flexibility of edge-based representations described previously, our approach is motivated by the fact that artists often rely on lines to first sketch the structure of a drawing before adding colors (Figure 3.1).

A diffusion curve partitions the space through which it is drawn, defining different colors on either side. By encoding an image via its discontinuities, diffusion curves naturally support traditional freehand drawing techniques of sketching and coloring. In a typical drawing session with our tool, the artist first draws curves corresponding to (potentially open) color boundaries. The user can then specify colors on either side of the curves, and a diffusion process propagates these colors to fill-in the empty space and create smooth color variations. By specifying blur values along the curve, the artist can also create smooth color transitions across the curve. In that sense, Diffusion Curves can be seen as a representation that allows the creation of complex images from line drawings (Figure 3.2). In addition, by adding a vectorization step to the edge structure extraction described in chapter 2, a given bitmap can be automatically converted into its diffusion curves representation.

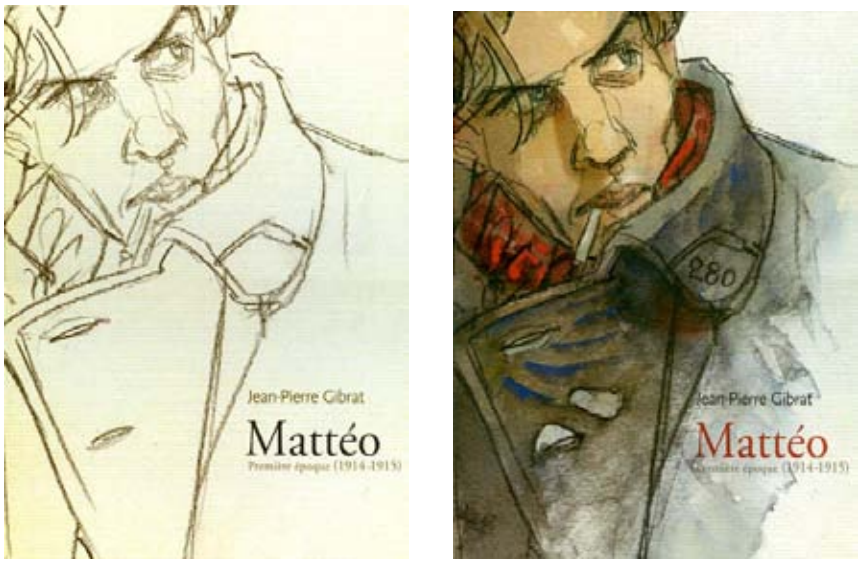


Figure 3.1: *Artists often use line drawings (left) to build the structure of their final colorized creation (right). ©Jean-Pierre Gibrat.*

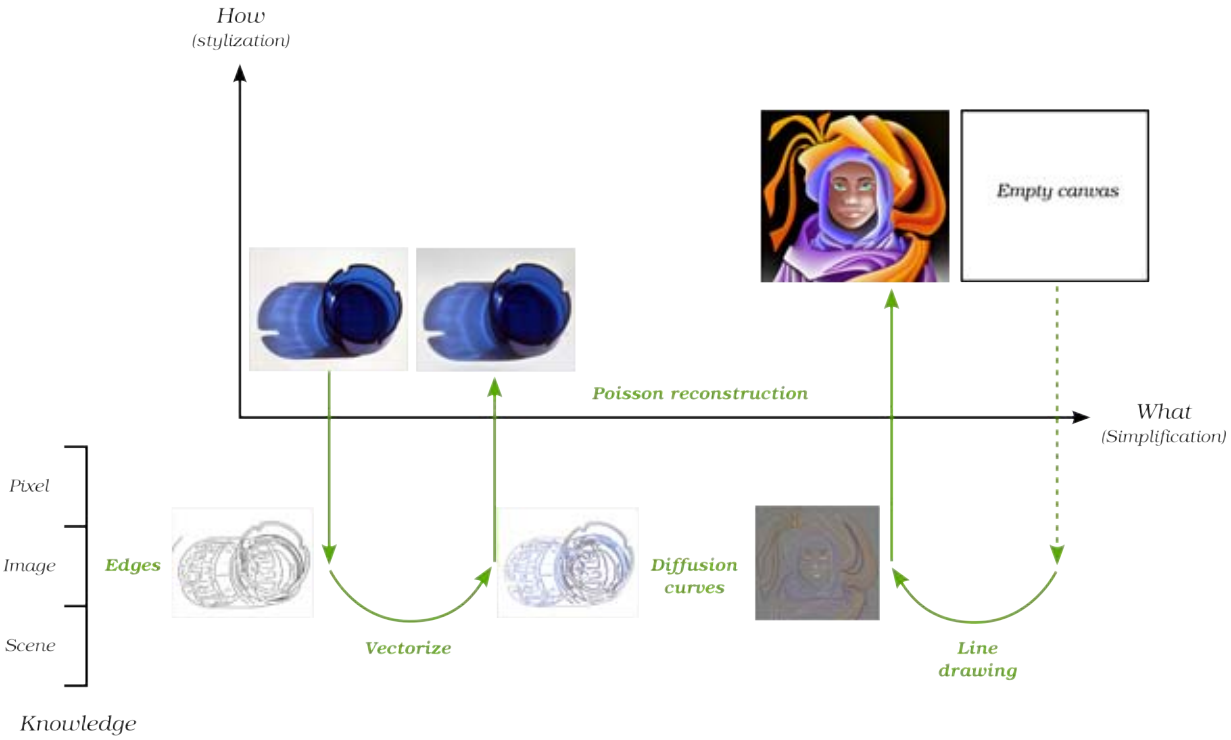


Figure 3.2: *Diffusion Curves can be used to create rich vector graphics via a line drawing interface, or to vectorize existing images.*

Compared to existing vector primitives that offer only limited support for complex color gradients, diffusion curves facilitate the creation of smooth color variations (Figure 3.3) that are usually better represented by raster graphics. This ability to efficiently encode color gradients broaden the range of artistic styles that can be produced in vector graphics. For example, the art movement of photorealism relies on smooth gradients to achieve soft shadows, defocus blur, diffuse shading, glossy reflections, and various material effects. The *airbrush* technique, widely used in design and urban art, is fundamentally based on (physical) color diffusion. The *Art Deco* painting movement, various comic styles, as well as numerous painting styles also heavily feature color gradients. Like all vector-based primitives, diffusion curves conveniently support a variety of operations, including geometry and color editing, keyframe animation and ready stylization. Moreover, their representation is compact and inherently resolution-independent.

Figure 3.3: *Diffusion curves (left), and the corresponding color image (right). Note the complex shading on the folds.* ©*Laurence Boissieux.*

The main contribution of this chapter is to define diffusion curves as a fundamental vector primitive to represent complex color gradients, along with two types of tools: (1) A prototype allowing manual creation and editing of diffusion curves. Thanks to an efficient GPU implementation, the artist benefits from instant visual feedback despite the heavy computational demands of a global color diffusion. (2) A fully automatic conversion from a bitmap image based on scale-space analysis. The resulting diffusion curves faithfully represent the original image and can then be edited manually.

1 Related Work

We review existing techniques to create complex color gradients and blur with modern vector graphic tools.

For a long time, vector graphics have been limited to primitives (paths, polygons) filled with uniform color or linear and radial gradients. Although skillful artists can create rich vector art with these simple tools, the resulting images often present flat or limited shading due to the lack of support for complex gradients and blur. Commercial tools such as Adobe Live Trace© assist the user in creating complex vector graphics from input bitmap images. They operate by segmenting an input image into regions of constant or slowly varying color, and fitting polygons onto these primitives. Although this class of methods produces convincing results in uniform areas, the segmentation typically generates a prohibitive number of primitives in smooth regions.

The ArDeco system of Lecot et al. [LL06] allows vectorization of more complex gradients using existing linear or radial gradient primitives. It is based on a segmentation of the input image into regions of slowly varying color, and an approximation of color variations within each region with linear or quadratic gradients. The resulting primitives are fully compatible with the SVG standard, but the approximation tends to produce sharp color transitions between segmented regions. A simpler solution to bypass these limitations, adopted by the SVG format and modern tools (Adobe Illustrator©, Corel CorelDraw©, Inkscape©), is to reblur the image once vector primitives have been rasterized. However, they only allow for a uniform blur for each given primitive, which, similar to the limitations of flat colors or simple gradients, necessitates an impractical large number of primitives to approximate complex image content.

Gradient meshes have been recently introduced (Adobe Illustrator©, Corel CorelDraw©) to address all of these issues by allowing a user to specify color values on the vertices of a quad mesh and smoothly interpolating these values over the mesh faces. However, creating a mesh from scratch requires much skill and patience, because the artist needs to accurately anticipate the mesh resolution and topology necessary to embed the desired smooth features. Consequently, most users rely on an example bitmap to drive the design of realistic gradient meshes [1]. The users first decompose an input photograph into several sub-objects and then draw meshes over each sub-object following their topology; finally, they sample colors in the original photograph, assigning them to the mesh vertices. Many tutorials describing this approach are available on the Web. Still, drawing effective meshes and performing accurate manual color sampling is very time consuming in practice (several hours or even days for detailed images) and requires a good appreciation of the image complexity to adopt an adequate mesh resolution.

Sun et al [SLWS07] propose to assist the user by automatically fitting an input gradient mesh to an input image. The fitting is achieved by minimizing the reconstruction error between the resulting image and an input photograph. Their semi-automatic method greatly reduces the time required to draw a precise mesh and sampling colors, although the user still has to manually specify the sub-objects of the picture and draw the initial meshes with an adequate resolution. This limitation is addressed by Lai et al. [LHM09] who show how to automatically generate a

[1] www.khulsey.com/masters_yukio_miyamoto.html

gradient mesh from a bitmap image using segmentation algorithms. Price and Barret [PB06] proposed a similar approach to gradient mesh for object vectorization, using recursive subdivisions until the reconstruction error falls below a fixed threshold. Their method produces faithful results but also generates many small patches in smooth regions.

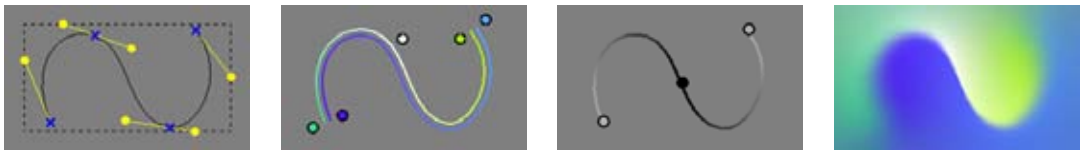
Yet, with both approaches, it remains unclear how to efficiently manipulate the resulting meshes for further editing. We believe this is due to the unnecessary constraints imposed by the use of a mesh: using a predefined topology, employing specific patch subdivision schemes, and choosing a global orientation. In practice, this translates into a dense net of patches that are not readily connected to the depicted content. Hence, the manipulation of such a set of primitives quickly becomes prohibitive for the non-expert.

The new representation described in this chapter offers the same level of visual complexity as that achieved by gradient meshes, but has two main advantages: it is *sparse*, and corresponds to *meaningful* image features. Indeed, the newly introduced diffusion curves are intuitive to create, as each primitive corresponds to an image feature; they are easy to manipulate and animate, as no constraint is imposed on connectivity, and no superfluous subdivision is required; and they are well adapted for stylization, which would be non-trivial with a gradient mesh approach. Moreover, our representation naturally lends itself to automatic extraction from a bitmap image: primitive locations are found completely automatically, and primitive attributes (color and blur) are extracted via computer vision algorithms.

In other words, compared to regions used in classic vector representations, or patches used in gradient meshes, our approach is motivated by the fact that most of the color variation in an image is caused by, or can be modeled with edges; and that (possibly open) regions or patches are implicitly defined in between. As reported in the previous chapter, such a sparse image representation is strongly motivated by the work of Elder [Eld99], who demonstrated that edges are a near-complete representation for images. Elder [EG01] also suggested the possibility of using edges to efficiently manipulate images with basic operations (edge delete, copy and paste). For this reason, our conversion algorithm starts from the same premises as Elder's system. But by vectorizing edges and their attributes, we greatly extend its manipulation capabilities to include shape, color, contrast, and blur operations. This way, we provide the user with more intuitive editing tools, and also support resolution-independence, stylization and key-frame animation.

2 Diffusion Curves

In this section we introduce the basic primitive of our representation, called a *diffusion curve*, and describe how to efficiently render an image from such primitives. Specification and manipulation of diffusion curves are discussed in subsequent sections.



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

Figure 3.4: *A Diffusion curve is composed of (a) a geometric curve described by a Bézier spline, (b) arbitrary colors on either side, linearly interpolated along the curve, (c) a blur amount linearly interpolated along the curve. The final image (d) is obtained by diffusion and reblurring. Note the complex color distribution and blur variation defined with a handful of controls.*

2.1 Data structure

The basic element of a diffusion curve is a geometric curve defined as a cubic Bézier spline (Figure 3.4(a)) specified by a set of control points P. The geometry is augmented with additional attributes: two sets of color control points C_l and C_r (Figure 3.4(b)), corresponding to color constraints on the *right* and *left* half space of the curve; and a set of *blur* control points (Σ) that defines the smoothness of the transition across the curve (Figure 3.4(c)). Intuitively, the curves diffuse color on each side with a soft transition across the curve given by its blur (Figure 3.4(d)).

Color and blur attributes can vary along a curve to create rich color transitions. This variation is guided by an interpolation between the attribute control points in attribute space. In practice, we use linear interpolation and consider colors in RGB space throughout the rendering process, because they map more easily onto an efficient GPU implementation and proved to be sufficient for the artists using our system. Controls points for geometry and attributes are stored independently, since they are generally uncorrelated. This leads to four independent arrays in which the control points (geometry and attribute values) are stored together with their respective parametric position t along the curve:

DiffusionCurve:
- P$[n_{pos}]$: array of $(x, y,$ tangent$)$;
- $C_l[n_l]$: array of (r,g,b,t);
- $C_r[n_r]$: array of (r,g,b,t);
- $\Sigma[n_\sigma]$: array of (σ,t);

The diffusion curves structure encodes data similar to Elder's edge-based representation [Eld99]. However, the vectorial nature of a diffusion curve expands the capabilities of Elder's discrete edges by allowing precise control over both shapes — via manipulation of control points and tangents — and appearance attributes — via color and blur control points (small circles on the Figures). This fine-level control, along with our realtime rendering procedure, facilitates the drawing and editing of smooth-shaded images.

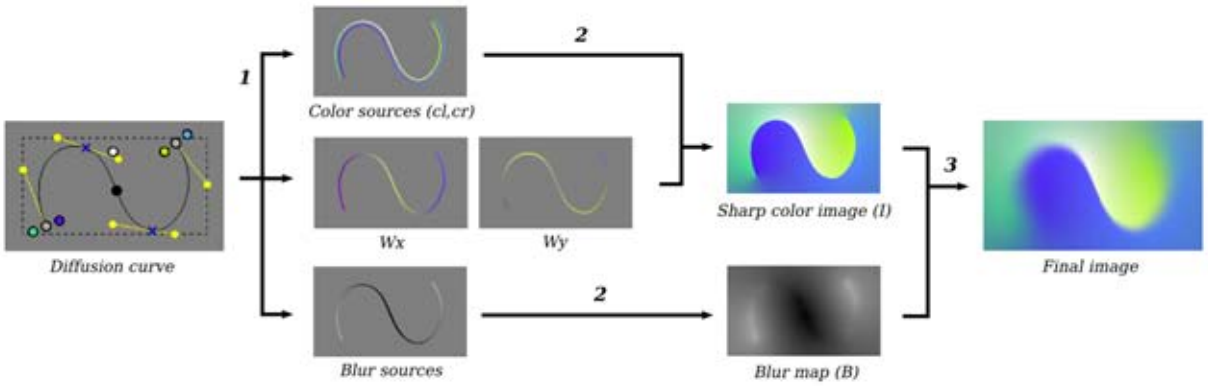


Figure 3.5: *Rendering diffusion curves requires (1) the rasterization of the color and blur sources, along with the gradient field* $\mathbf{w} = (\mathbf{w}_x, \mathbf{w}_y)$*, (2) the diffusion of colors and blur, and (3) the reblurring of the color image.*

2.2 Rendering smooth gradients from diffusion curves

Three main steps are involved in our rendering model (see Figure 3.5): (1) rasterize a *color source* image, where color constraints are represented as colored curves on both sides of each Bézier spline, and the rest of the pixels are uncolored; (2) *diffuse* the color sources similarly to heat diffusion — an iterative process that spreads the colors over the image; we implement the diffusion on the GPU to maintain realtime performance; and (3) *reblur* the resulting image with a spatially varying blur guided by the blur attributes. Technical details about these three steps are explained in the following sections.

2.2.1 Color sources

Using the interpolated color values, the first step renders the left and right color sources $cl(t), cr(t)$ for every pixel along the curves. An alpha mask is computed along with the rendering to indicate the exact location of color sources versus undefined areas. For perfectly sharp curves, these color sources are theoretically infinitely close. However, rasterizing pixels separated by too small a distance on a discrete pixel grid leads to overlapping pixels. In our case, this means that several color sources are drawn at the same location, creating visual artifacts after the diffusion. Our solution is to distance the color sources from the curve slightly, and to add a color gradient constraint directly on the curve. The gradient maintains the sharp color transition, while the colors, placed at a small distance d in the direction normal to the curve, remain separate.

More precisely, the gradient constraint is expressed as a gradient field $\mathbf{w}$ which is zero everywhere except on the curve, where it is equal to the color derivative across the curve. We decompose the gradient field into a gradient along the x direction $\mathbf{w}_x$ and a gradient along the y direction $\mathbf{w}_y$. For each pixel on the curve, we compute the color derivative across the curve from the curve normal N and the left (cl) and right (cr) colors as follows (we omit the t parameter for clarity): $\mathbf{w}_{x,y} = (cl - cr)N_{x,y}$

We rasterize the color and gradient constraints in 3 RGB images: an image C containing colored pixels on either side of each curve, and two images (W_x, W_y) containing the gradient field components. In practice, the gradient field is rasterized along the curves using lines of one pixel width. Color sources are rasterized using triangle strips of width $2d$ with a special pixel shader that only draws pixels that are at the correct distance d (Figure 3.5(1)). In our implementation d is set at 3 pixels. Pixel overlap can still occur along a curve in regions of high curvature (where the triangle strip overlaps itself) or when two curves are too close to each other (as with thin structures or intersections). A simple stencil test allows us to discard overlapping color sources before they are drawn, which implies that solely the gradient field $\mathbf{w}$ dictates the color transitions in these areas. An example of such a case can be seen in Figure 3.3, where the eyebrows are accurately rendered despite their thin geometry.

2.2.2 Diffusion

Given the color sources and gradient fields computed in the previous step, we next compute the color image I resulting from the steady state diffusion of the color sources subject to the gradient constraints (Figure 3.5(2)). Similarly to previous methods [Car88, Eld99, PGB03], we express this diffusion as the solution to a Poisson equation, where the color sources impose local constraints:

$$\begin{aligned} \Delta I &= \operatorname{div} \mathbf{w} \\ I(i,j) &= C(i,j) \text{ if pixel } (i,j) \text{ stores a color value} \end{aligned}$$

where Δ and div are the Laplace and divergence operators, discretized with finite differences:

$$\begin{aligned} \Delta I(i,j) &= -4I(i,j) + I(i-1,j) + I(i+1,j) + I(i,j-1) + I(i,j+1) \\ \operatorname{div} \mathbf{w}(i,j) &= \frac{\mathbf{w}_x(i+1,j) - \mathbf{w}_x(i-1,j)}{2} + \frac{\mathbf{w}_y(i,j+1) - \mathbf{w}_y(i,j-1)}{2} \end{aligned}$$

From this discretization, the image values can be expressed as:

$$I(i,j) = \frac{I(i+1,j) + I(i-1,j) + I(i,j+1) + I(i,j-1) - \operatorname{div} \mathbf{w}(i,j)}{4}$$

Computing the Poisson solution requires solving a large, sparse, linear system, which can be very time consuming if implemented naively. To offer interactive feedback to the artist, we solve the equation with a GPU implementation of the multigrid algorithm [BHM00, GWL+03]. McCann and Pollard [MP08] give a detailed description of a realtime Poisson solver very similar to ours. The idea behind multigrid methods is to use a coarse version of the domain to efficiently solve for the low frequency components of the solution, and a fine version of the domain to refine the high frequency components.

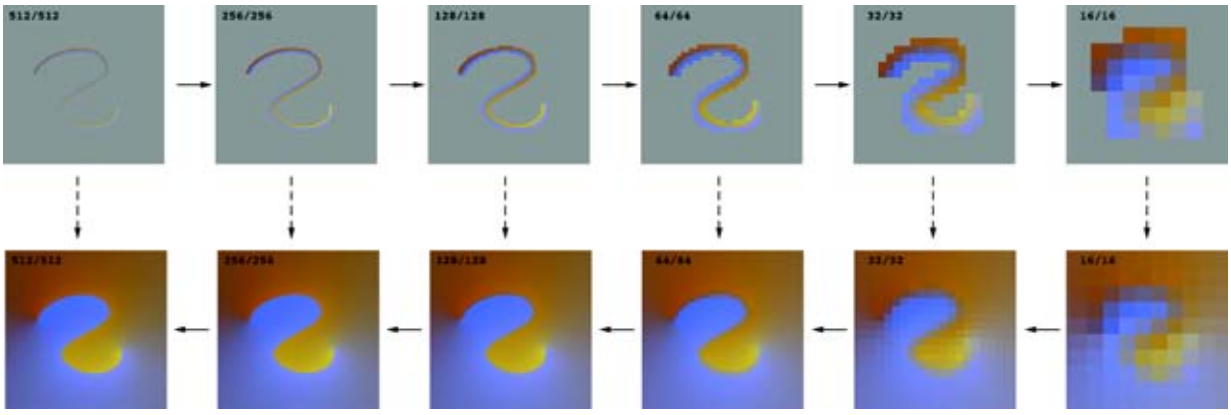


Figure 3.6: *The multigrid algorithm. Color and gradient constraints are repeatedly downsampled (top row). An initial solution is computed at the lowest level, by iteratively diffusing the color constraints. The solution is then refined at finer scales, by using the coarse-scale solutions and the finer-scale color constraints (bottom row).*

The algorithm works in a V-like manner; the color source image C and the gradients $\mathbf{w}_x$ and $\mathbf{w}_y$ are progressively downsampled, or *restricted* (Figure 3.6 top). The solution is computed first at the lowest resolution, and then upsampled and refined (Figure 3.6 bottom). Jacobi relaxations are used to solve for each level of the multigrid: for a given iteration k and a resolution level l, the color value $I(i,j)_l^k$ is updated as:

$$I_l^k(i,j) = \frac{I_l^{k-1}(i+1,j) + I_l^{k-1}(i-1,j) + I_l^{k-1}(i,j+1) + I_l^{k-1}(i,j-1) - \operatorname{div}\mathbf{w}_l(i,j)}{4}$$

The color constraints are re-imposed after each iteration:

$$I_l^k(i,j) = C_l(i,j) \text{ if pixel } (i,j) \text{ stores a color value}$$

To construct the image pyramid necessary for the multigrid solver, we downsample the gradient using a 3×3 kernel:

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

This restriction kernel, also used by McCann and Pollard [MP08], captures all gradient directions from the finer scale and preserves the gradient magnitude. The color constraints are downsampled with an average filter: a pixel at coarse scale receives the average of the constraints of the four corresponding pixels in the finer scale.

We limite the number of relaxation iterations to achieve realtime performances. Typically, for a 512×512 image we use $5l$ Jacobi iterations per multigrid level, with l the level number from fine to coarse. This number of iterations can then be increased when high quality is required. All our images have been rendered using an Nvidia GeForce 8800, providing realtime performance on a 512×512 grid with a reasonable number of curves (several thousands).

2.2.3 Reblurring

The last step of our rendering pipeline takes as input the color image containing sharp edges, produced by the color diffusion, and reblurs it according to blur values stored along each curve. However, because the blur values are only defined along curves, we lack blur values for off-curve pixels. A simple solution, proposed by Elder [Eld99], diffuses the blur values over the image similarly to the color diffusion described previously. We adopt the same strategy and use our multigrid implementation to create a blur map B from the blur values. The only difference to the color diffusion process is that blur values are located exactly on the curve so we do not require any gradient constraints. This leads to the following equation:

$$\begin{aligned} \Delta B &= 0 \\ B(i,j) &= \sigma(i,j) \text{ if pixel } (i,j) \text{ is on a curve} \end{aligned}$$

Giving the resulting blur map B, we apply a spatially varying blur on the color image (Figure 3.5(3)), where the size of the blur kernel at each pixel is defined by the required amount of blur for this pixel. Despite a spatially varying blur routine implemented on the GPU [BFSC04], this step is still computationally expensive for large blur kernels (around one second per frame in our implementation), so we bypass it during curve drawing and manipulations and reactivate it once the drawing interaction is complete.

2.2.4 Panning and zooming

Solving a Poisson equation leads to a global solution, which means that any color value can influence any pixel of the final image. Even though the local constraints introduced by the color sources reduce such global impact, this raises an issue when zooming into a sub-part of an image, because curves outside the current viewport should still influence the viewport's content. To address this problem without requiring a full Poisson solution at a higher resolution, we first compute a low-resolution diffusion on the unzoomed image domain, and use the obtained solution to define Dirichlet boundary conditions around the zooming window. This gives us a sufficiently good approximation to compute a full-resolution diffusion only within the viewport.

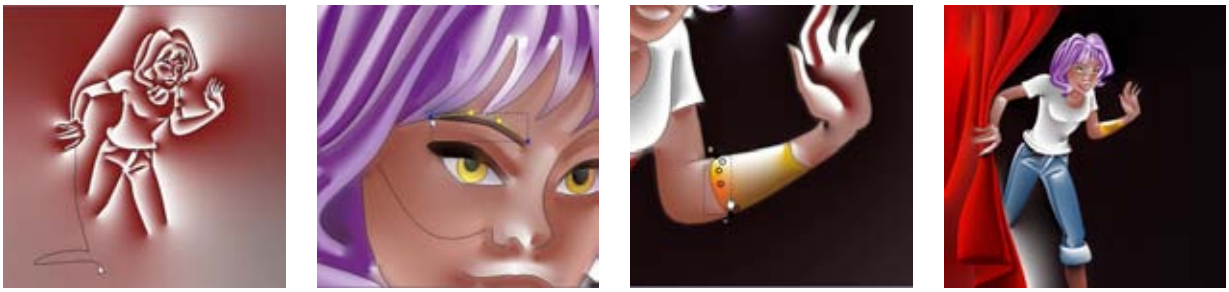
3 Creating diffusion curves

The process of creating images varies across artists. One may start from scratch and give free rein to his imagination while another may prefer to use an existing image as a reference. We provide the user with both options to create diffusion curves. For *manual* creation, the artist can create an image with our tool by sketching the lines of the drawing and then filling in the color. When using an image as a template, we propose two methods. *Assisted:* The artist can trace manually over parts of an image and we recover the colors of the underlying content. *Automatic:* the artist can automatically convert an image into our representation and possibly post-edit it.

3.1 Manual creation

To facilitate content creation for the artist, we offer several standard tools: editing of curve geometry, curve splitting, copy/paste, zooming, color picking, etc. We also developed specific tools: copy/paste of color and blur attributes from one curve to another, editing of attributes control points (add, delete, and modify), etc. The tutorial provided on our project page [2] gives a more complete description of our current prototype interface.

To illustrate how an artist can use our diffusion curves, we show in Figure 3.7 the different stages of an image being drawn with our tool. The artist employs the same intuitive process as in traditional drawing: a sketch followed by color filling.



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

Figure 3.7: *Example steps for manual creation: (a) sketching the curves, (b) adjusting the curve's position, (c) setting colors and blur along the diffusion curve and (d) the final result. The image was created by an artist at first contact with the tool and it took 4 hours to create.©Laurence Boissieux.*

3.2 Tracing an image

In many situations an artist will not create an artwork entirely from scratch, but instead use existing images for guidance. For this, we extract colors of an underlying bitmap along a drawn curve. This process is illustrated in Figure 3.8.

The challenge here is to correctly extract and vectorize colors on each side of a curve. We also need to consider that color outliers might occur due to noise in the underlying bitmap or because the curve positioning was suboptimal. We first uniformly sample the colors along the curve at a distance d (same as the one used for rendering) in the direction of the curve's normal. We then identify color outliers by measuring a standard deviation in a neighborhood of the current sample along the curve. We work in CIE L*a*b* color space (considered perceptually uniform for just-noticeable-differences), and tag a color as an outlier if it deviates too much from the mean in either the L*, a* or b* channel. We then convert back colors to RGB at the end of the vectorization process for compatibility with our rendering system.

[2] http://artis.imag.fr/Publications/2008/OBWBTS08/

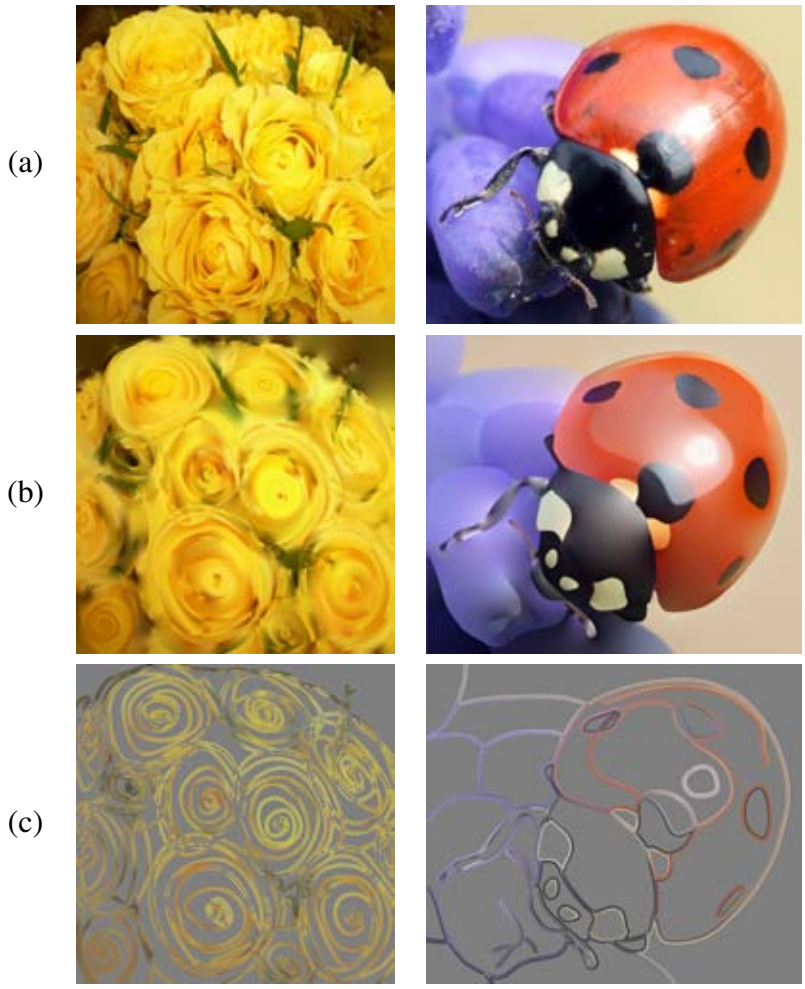


Figure 3.8: *Tracing with diffusion curves: (a) Original bitmaps; (b)* left*: Result of a stylistic tracing using color sampling (artist drawing time: less than a minute) ©Philippe Chaubaroux;* right*: Result of a tracing using active contours and color sampling (artist drawing time: 90 minutes). (c) The corresponding diffusion curves (color sources have been thickened for illustration purpose).*

To obtain a linear color interpolation similar to that used for rendering, we fit a polyline to the color points using the Douglas-Peucker algorithm [DP73]. The iterative procedure starts with a line connecting the first and last point and repeatedly subdivides the line into smaller and smaller segments until the maximum distance (still in CIE L*a*b*) between the actual values and the current polyline is smaller than the error tolerance ε. The end points of the final polyline yield the color control points that we attach to the curve. A creative example that uses color sampling is illustrated in Figure 3.8(b)-left image, where an artist has drawn very stylistic shapes, while using the color sampling feature to reproduce the global tone of the original image, similarly to an in-painting process [BSCB00].

When tracing over a template, one would normally want to position the curves over color discontinuities in the underlying image. Since it is not always easy to draw curves precisely at edge locations in a given image, we provide some help by offering a tool based on *Active Contours* [KWT87]. An active contour is attracted to the largest gradient values of the input bitmap and allows the artist to iteratively snap the curve to the closest edge. The contour can also be easily corrected when it falls into local minima, or when a less optimal but more stylistic curve is desired. Figure 3.8(b)-right shows the image of a lady bug created using geometric snapping and color extraction. While the artist opted for a much more stylized and smoothed look compared to the original, the image still conveys diffuse and glossy effects, defocus blur, and translucency.

3.3 Automatic extraction from a bitmap

Finally, an artist might want to add stylization and expression to an already existing image. We propose a method for automatically extracting and vectorizing diffusion curves data from a bitmap, based on the multi-scale edge extraction algorithm described in chapter 2.

Data Extraction: We first apply the scale space analysis that produces an edge map containing the edge locations and the blur values for each edge pixel. The next step extracts the colors on either side of the edges explicitly. For this, we connect pixel-chains from the edge map and proceed to sample colors in the original image on each side of the edge in the direction of the edge normal. In practice, the gradient normal to the edge is difficult to estimate for blurry edges, so we use the direction given by the normal of a polyline fitted to each edge. For an estimated blur σ, we pick the colors at a distance $3 \cdot \sigma$ from the edge location, which covers 99% of the edge's contrast, assuming a Gaussian-shaped blur kernel [Eld99]. While the $3 \cdot \sigma$ distance ensures a good color extraction for the general case, it poses numerical problems for structures thinner than 3 pixels ($\sigma < 1$); in this particular case, color cannot be measured accurately.

Conversion to diffusion curves: For vectorization of positions, we take inspiration from the approach used in the open source Potrace©software [Sel03]. The method first approximates a pixel chain with a polyline that has a minimal number of segments and the least approximation error, and then transforms the polyline into a smooth poly curve made from end-to-end connected Bézier curves. The conversion from polylines to curves is performed with classical least square Bézier fitting based on a maximum user-specified fitting error and degree of smoothness. For attribute vectorization, we use the same method as in Section 3.2.

Several parameters determine the complexity and quality of our vectorized image representation. For the edge geometry, the Canny threshold determines how many of the image edges are to be considered for vectorization; a despeckling parameter sets the minimum length of a pixel chain to be considered for vectorization; and finally, two more parameters set the smoothness of the curve fitting and the fitting error. For the blur and color values, two parameters are considered: the size of the neighborhood for eliminating outliers, and the maximum error accepted when fitting the polyline. For most images, we use a Canny high threshold of 0.82 and low threshold of 0.328, we discard pixel chains with less than 5 pixels, we use a smoothness param-

eter of 1 (Potrace default) and we set the fitting error to 0, so the curve closely approximates the original edges. For attributes, we consider a neighborhood of 9 samples, and the maximum error accepted is 2 blur scales for the blur and 30 CIE L*a*b* units for colors. Figure 3.9 and 3.11 show the result of our automatic conversion from photographs.

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

Figure 3.9: *Example of our reconstruction: (a) original image; (b) result after conversion into our representation; (c) automatically extracted diffusion curves; (d) RGB difference between original and reconstructed image (amplified by 4); note that the most visible error occurs along edges, most probably because, through vectorization, we change their localization.*

4 Results

Diffusion curves, as vector-based primitives, benefit from the advantages of traditional vector graphics: zooming-in preserves sharp transitions (Figure 3.11 (e)) and keyframing is easily performed via linear interpolation of geometry and attributes (Figure 3.10). Our representation is equally well suited for global and local image stylization. Curve shapes and attributes can be easily modified to obtain effects such as that presented in Figure3.11(d). For diffusion curves extracted from an image, we use the lifetime measure computed by the scale space analysis to adjust preservation of detail (Figure3.11(c)).

Figure 3.10: *Keyframing with diffusion curves: Three keyframes of an animation.* ©*Laurence Boissieux.*

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

Figure 3.11: *Stylization effects: (a) original bitmap; (b) Automatic reconstruction; (c) Reconstruction simplified by removing edges with low lifetime; (d) Global shape stylization applied to (c); (e) Enlargement of (b).*

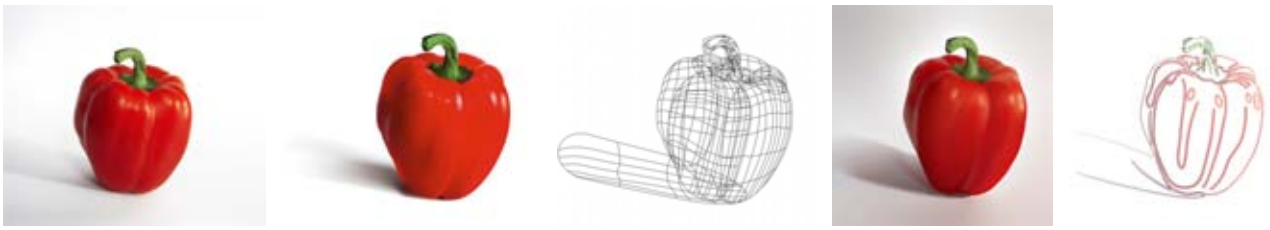
To validate our approach and to collect valuable practical feedback, we had various artists use our prototype. Most of our results were generated in these sessions. All artists were well versed in digital content creation tools, but had no technical background. They were given a brief tutorial (see our project webpage), amounting to approximately 10 minutes of instructions. The artists were able to create many varied and intricate examples from the very first session and found the manipulation of diffusion curves intuitive after a short accommodation phase. Manual image creation took anywhere from several minutes (Figure 3.8(b)) to a few hours (Figure 3.7). However, the artists agreed that a more powerful user interface would greatly speed up the creation process.

5 Discussion & Future work

In the previous sections, we presented our new vector-based primitive, and explained the various options at an artist's disposal to create smooth-shaded images thanks to this intuitive representation. We now compare our approach with the most commonly used vector tool for creating images with equally complex gradients: Gradient Meshes. Next, we identify the remaining challenges that we would like to address in future work.

5.1 Comparison with Gradient Meshes

Representational efficiency: In terms of sparsity of encoding, both gradient meshes and diffusion curves are very efficient image representations. A direct comparison between both representations is difficult, as much depends on the chosen image content (for example, gradient meshes require heavy subdivision to depict sharp edges and it can be difficult to conform the mesh topology to complex geometric shapes). Furthermore, Price and Barret [PB06] presented a more compact sub-division gradient mesh, yet all available tools employ a regular mesh. While the diffusion curves representation appears more compact at first glance (see Figure 3.12), it should be noted that each geometric curve can hold an arbitrary amount of color and blur control



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

Figure 3.12: *Gradient Mesh comparison: (a) Original photograph; (b,c) Manually created gradient mesh (© Brooke Nuñez Fetissoff http://lifeinvector.com/), with 340 vertices (and as many color control points); (d,e) Our drawing created by manually tracing over the image; there are 38 diffusion curves, with 365 geometric, 176 left-color, and 156 right-color control points.*

points (see Table 3.13). So, while the sparsity of encoding of both representations can be considered comparable, we would argue the flexibility of diffusion curves to be a significant benefit, as it allows us any degree of control on a curve, without a topologically-imposed upper or lower bound on the number of control points. On the other hand, the mesh-based structure of gradient meshes allows simple rasterization and bilinear interpolation, which better suits the rendering pipelines of existing vector graphic software than our fast Poisson solver.

	Curves	P	Cl	Cr	Σ
Roses (fig. 3.8 left)	20	851	581	579	40
Lady bug (fig. 3.8 right)	71	521	293	291	144
Curtain (fig. 3.7)	131	884	318	304	264
Dolphin (fig. 3.9)	1521	6858	3254	3271	3433

Figure 3.13: *Number of curves, geometric control points (P), left and right color control points (Cl, respectively Cr) and blur control points (Σ) for the images of this paper.*

Usability: We believe that diffusion curves are a more natural drawing tool than gradient meshes. As mentioned previously, artists commonly use strokes to delineate boundaries in an image. Diffusion curves also allow an artist to evolve an artwork gradually and naturally. Gradient meshes, on the other hand, require careful planning and a good understanding of the final composition of the intended art piece. Most gradient mesh images are a complex combination of several individual — rectangular or radial — gradient meshes, often overlapping. All these decisions have to be made before the relevant image content can be created and visualized.

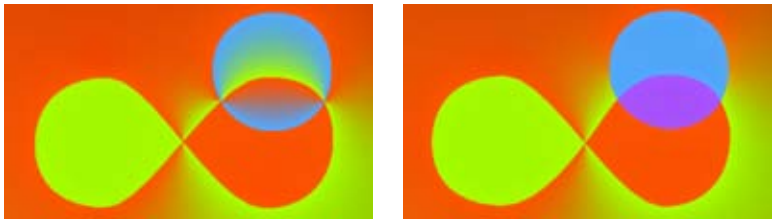
Topology: In some situations, the topology constraints of gradient meshes can be rather useful, for example when moving a gradient mesh to a different part of an image, or when warping the entire mesh. Such manipulations are also possible in our representation, but not as straightforward. For moving part of an image, the relevant edges have to be selected and moved as a unit. More importantly, without support for layering and transparency (see Section 5.2) it is difficult to ascertain how the colors of outer edges should interact with their new surroundings. A mesh warp could be implemented as a space warp around a group of edges.

5.2 Future challenges

Our current implementation of diffusion curves prevents their use in existing vector graphics software because of the non-trivial Poisson integration. An interesting area for future work appears to be the implementation of a Poisson solver that fits better in existing rendering pipelines. One solution would be to use triangular finite elements to generate a mesh that can then be rendered at any resolution with regular rasterization.

Currently, our representation is single layered, but we are aware that multiple, independent layers offer more flexibility to the artist. To fully take advantage of a layered system, we need to address the interaction with multiple layers and the additional computational demands. Blending of layers would also require a notion of transparency. Our current representation is more related to planar-maps [ASP07] that model vector graphics in a single layer.

Another potential improvement would be the way diffusion curves deal with intersections. Currently, diffusion curves present a specific (although predictable and meaningful) behavior: the colors attached to the two intersecting curves essentially compete with each other, which creates a smooth color gradient after diffusion (Figure 3.14(a)). If this default behavior is undesirable, the user can correct it by either adding color controls on each side of the intersection, or by splitting the curves in several parts with different colors (Figure 3.14(b)). Automating such behavior would represent a powerful tool for easing user interactions.



(a) (b)

Figure 3.14: *The default behavior of diffusion curves at intersections (a) can be corrected by curve splitting and color editing (b).*

Another limitation, common to all vector graphics, occurs in images or image regions that contain many small color or luminance variations, such as textures. In practice, most of the visual information of highly textured regions is captured by the automatic conversion, but imprecision occur when the texture is composed of many small structures (small compared to the distance d defined in Section 2.2.1). Moreover, the large amount of curves required to represent textures makes a vector representation inefficient and difficult to manipulate. Incorporating a diffusion curves version of texture synthesis tools is an interesting area of future research.

6 Conclusions

We have introduced diffusion curves as a new image representation, offering most of the benefits usually found in vector approaches, such as resolution independence, exact editability, and compactness; while at the same time allowing to depict highly complex image content, generally only realizable with raster graphics. By encoding image discontinuities, diffusion curve images are comparable both in quality and coding efficiency with gradient meshes, but are simpler to create (according to several artists who have used both tools), and can be captured from bitmaps fully automatically.

Part III

Stylization Textures for Videos and 3D Scenes

We introduce two methods that allow the application of stylized textures over 3D scenes and videos in a temporally coherent fashion. These two approaches address the specific problem of temporal coherence for color region stylization, leaving contour stylization for future work. In particular, we focus on the animation of the visual characteristics of watercolor paintings. Before describing our methods, we briefly define what is a stylization texture, along with an explanation of the inherent contradiction of temporal coherence.

chapter 4

Problem Statement

This chapter introduces the notions of stylization textures and temporal coherence, that are fundamental aspects of the methods described in Chapter 5 and 6.

1 Stylization Textures

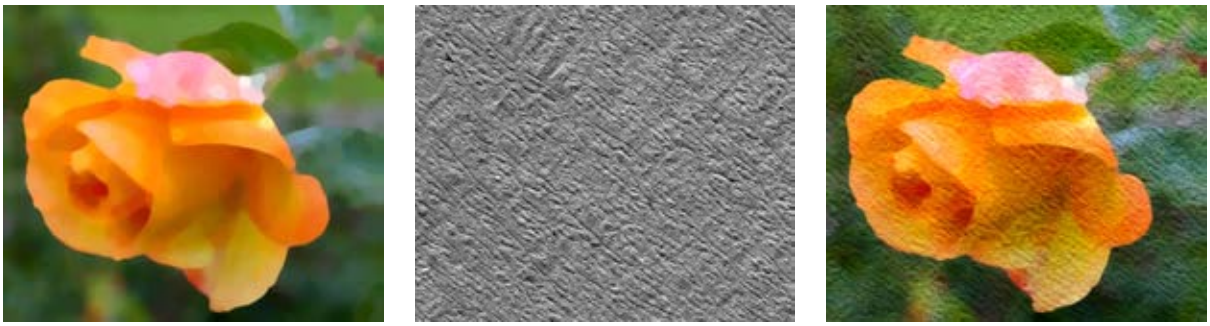
The stylized appearance of a painting or drawing is greatly due to the characteristic texture of the medium. Brush strokes, charcoal marks, paper grain, pigments are features of the medium that can be seen as textural effects over the original visual information to depict. In the case of watercolor, pigments are mixed with water but do not dissolve totally. The result is that, after drying, even on a totally smooth paper, the pigment density is not constant. On the one hand there are low-frequency density variations due to a non homogeneous repartition of water on the canvas; on the other hand there are high-frequency variations due to non-homogeneous repartition of pigments in water. In addition, the grain of the canvas itself introduces density variations since the pigments are deposited in priority in cavities of the paper. Figure 4.1 illustrates this behavior, that makes watercolor especially beautiful and evocative.

We reproduce similar pigmentation effects using gray level paper and pigment textures, where the texture value can be seen as a pigment density d that darken or lighten the image color C. We use an empirical blending equation [BKTS06] to produce this color variation: $C' = C\big(1-(1-C)(d-1)\big)$. This texture based approach is both fast and flexible, as the user is free to use procedural or scanned textures to reproduce a large variety of granulation patterns (Figure 4.2). We also show in the next chapter that stylization textures can be used to mimic other styles such as hatching or collage.

2 Temporal Coherence

In this dissertation we focus on the particular issue of stylizing animated content with style marks (pigments, strokes, etc), which raises the difficult question of **temporal coherence**. Traditional

Figure 4.1: *Real watercolor painting by Max Cabanes©exhibit strong granulation effects.*



(a) Original image (b) Gray level paper texture (c) Final image

Figure 4.2: *We apply a gray level texture over a color image to mimic variations in pigment density.*

media like painting or drawing are usually created on 2D canvas, while animations aim at representing stylized objects moving in 3D environments. Animating the 2D style marks along 3D trajectories implies the concurrent fulfillment of three constraints. First, the style marks should have a constant size and density in the image in order to preserve the 2D appearance of the medium. Second, the style marks should follow the motion of the 3D objects they depict to avoid the sliding of the style features over the 3D scene ("shower door effect") [Mei96]. Finally, a sufficient temporal continuity between adjacent frames is required to avoid popping and flickering. Temporal coherence for mark-based styles refers to the join satisfaction of these three contradictory constraints, and is a major topic in the research field of Non Photorealistic Rendering. We review in the two following chapters the existing methods proposed in the field.

We present two methods that animate stylization textures in a temporally coherent fashion. The first method addresses the stylization of 3D scenes and is especially well suited for real time applications like video games (Chap. 5). The second method targets the stylization of videos, where 3D information is not available. We rely on an optical flow algorithm to estimate the apparent motion and use it as a support for the animated texture (Chap. 6). Both methods share the common idea of addressing temporal coherence *locally in time*. In practice, the texture is constrained to follow the objects motion during enough frames so that the observer can "track" the texture elements and perceive their motion. If applied during too many frames, the animated texture would accumulate distortions and deviate from its original 2D appearance. The solution proposed in the two following chapters is to limit the temporal support of the animated texture in order to replace the distorted texture by an undistorted one. Because these different versions of the texture follow the same motion trajectories, the illusion of movement is preserved.

chapter 5

Dynamic Solid Textures for Real-Time Coherent Stylization

The research presented in this chapter was done in collaboration with Pierre Bénard and Joëlle Thollot during Pierre Bénard's master. Pierre Bénard did most of the implementation in this project. The result of this work has been published at I3D 2009 [BBT09]. Part of this research has been transferred by David Lanier[a] into a Mental Ray plug-in called Graphanim[b]*, in collaboration with Studio Broceliande[c].*

[a] http://www.davidlanier3d.com/
[b] http://www.studio-broceliande.fr/graphanim/index.htm
[c] http://www.studio-broceliande.fr/

In this chapter we describe *dynamic textures*, a method that facilitates the integration of temporally coherent stylization in real-time rendering pipelines. Central to our technique is an object space *infinite zoom* mechanism that guarantees a quasi-constant size and density of the texture elements in screen space for any distance from the camera. This simple mechanism preserves most of the 2D appearance of the medium supported by the texture while maintaining a strong temporal coherence during animation. The infinite zoom illusion can be applied to both 2D and 3D textures. However, we present our approach for 3D textures (the dynamic solid textures) which alleviate the need for complex computation or manual definition of 2D parameterizations over the 3D surfaces. As a result, our method is easy to integrate in 3D rendering pipelines where the distance of an object to the camera is readily available (Figure 5.1). These solid textures can be produced either procedurally or by synthesis from a 2D exemplar, which offers a great deal of flexibility to the final user. Although we focus in this manuscript on the specific style of watercolor, we show that our texture-based approach can be applied to other styles such as hatching.

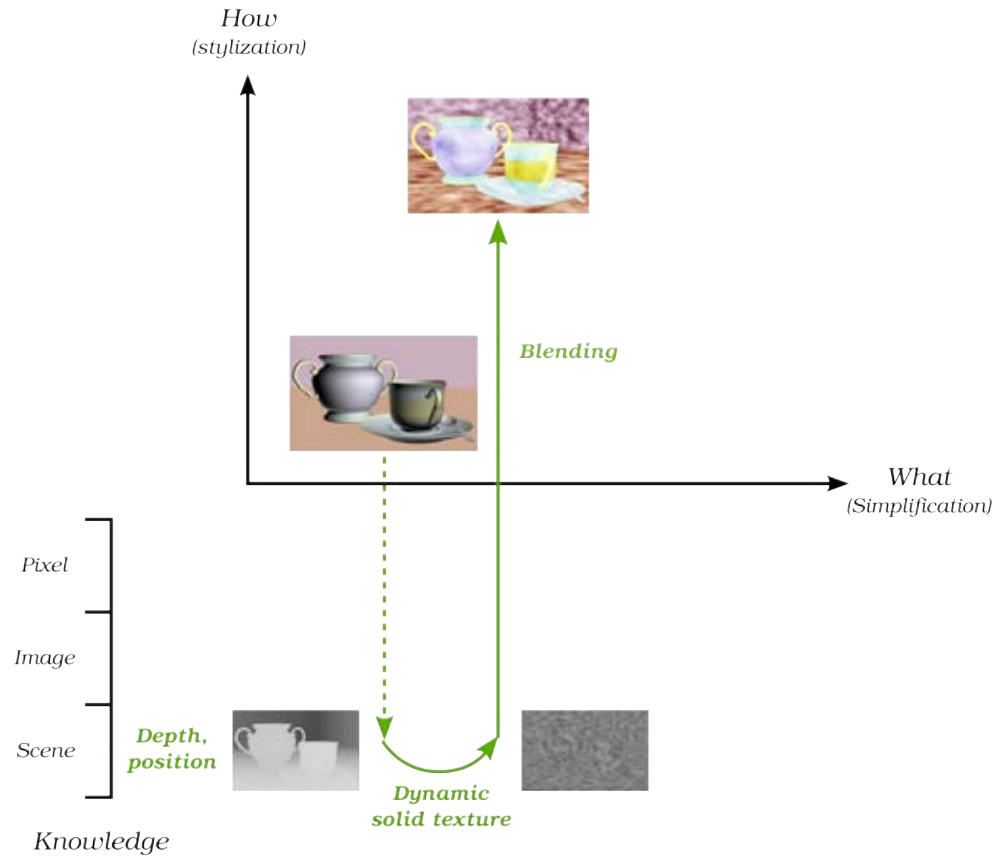


Figure 5.1: *Starting from a 3D scene, our method adapts the scale of solid textures according to the depth of the objects to produce texture elements with a uniform size in screen space.*

1 Related Work

In the following we review the methods proposed in the non photorealistic rendering literature to solve the temporal coherence problem of animated 3D scenes.

In order to resolve the contradiction between a 2D style and a 3D motion, Meier [Mei96] proposes in her seminal work on painterly rendering to decorrelate the appearance of style marks from their motion. In her approach, style marks (paint strokes in her case) are drawn with constant size billboards which preserve their 2D appearance. Each mark is then attached to a 3D anchor point on the 3D object that it depicts. Although this approach has been extended to numerous styles (painterly [Dan99, VBTS07], stippling [PFS03], watercolor [BKTS06]), the data structure required to manage the anchor points and the expensive rendering of each individual style element makes this family of methods not well-suited for real-time rendering engines.

The individual management of style marks can be avoided by grouping the style marks in textures. The *Dynamic Canvas* method [CTP+03] applies a 2D paper texture over the screen in order to stylize a 3D environment during a real-time walkthrough. This approach reproduces 3D

transformations of the camera with 2D transformations of the texture. Translations in depth of the camera are mimicked with an infinite zoom mechanism that preserves a quasi-constant size of the texture in screen space. Although this approach provides a convincing trade-off between the 3D motion of the camera and the 2D appearance of the paper, it is limited to navigation of static scenes with a restricted set of camera motions. Moreover, the image space mechanisms proposed by Cunzi et al. models the scene as a single plane which leads to sliding artifacts for strong parallax. Coconu et al. [CDH06] and Breslav et al. [BSM+07] adopt an approach similar to Dynamic Canvas by applying a stylization texture in screen space on each object of the scene. In these methods, the projected 3D transformations of the objects are approximated by their closest 2D transformations. Because the textures are only transformed in screen space, these methods well-preserve the 2D appearance of style marks. On the other hand, the approximation of a 3D transformation by a 2D transformation can lead to sliding effects for extreme 3D motions. A novel approach proposed recently by Han et al. [HRRG08] introduces an infinite zoom mechanism. This 2D multi-scale texture synthesis algorithm generates texture elements on the fly during the zoom. The zoom illusion produced by this method is more accurate than the one of Cunzi et al. but its computing cost limits its integration in real-time rendering pipelines. In this paper we extend the Dynamic Canvas infinite zoom mechanism to object space textures, which allows the real-time stylization of dynamic objects without any sliding effects.

The above approaches tend to sacrifice accuracy in 3D motion to preserve the 2D appearance of the style. However, the converse can also lead to reasonable solutions. This is the approach adopted by *art maps* [KLK+00] and *tonal art maps* [PHWF01] that map a stylization texture directly on the 3D objects of the scene. Using object space textures, the style marks are perfectly attached to the object but are severely distorted by the perspective projection. The art maps solution relies on the *mipmapping* mechanism to adapt the scale of the texture according to the distance to the camera. This approach corrects the texture compression induced by depth and can be extended to the correction of perspective deformation using the more complex *ripmaps* mechanism [KLK+00]. As noted by Praun et al. [PHWF01], higher temporal coherence can be obtained for binary styles by including the binary marks of one tonal art map level into the next level. Freudenberg et al. [FMS01] prove the performance of these methods by integrating an art map based non photorealistic rendering in the Fly3D game engine.

The infinite zoom mechanism described in this paper extends the art maps approaches in several ways. First, while the mipmap mechanism addresses the texture compression due to minification (zooming out), it produces simple linear blending for texture magnification (zooming in). Our mechanism in opposition produces quasi-constant size texture elements for any distance to the camera. Then, by combining several scales of the texture at a time, our approach is less specialized to binary styles than tonal art maps. Finally, mapping art maps on 3D objects requires the definition of a 2D parameterization of the 3D surfaces. The automatic definition of parameterization that avoids texture distortions or visible seams is still an active research topic. In the case of tonal art maps, Praun et al. [PHWF01] propose to automatically compute the parameterization using *lapped textures* but this approach requires additional data structures. Because our approach is based on solid textures, it does not require this definition of additional surface parameterization.

2 Dynamic Solid Textures

We present in this section the object-space infinite zoom mechanism central to the dynamic solid textures. We will describe in section 3 the application of our approach to the real-time coherent stylization of 3D animations, with various styles including watercolor.

2.1 Object Space Infinite Zoom Mechanism

The contradictory goals of the infinite zoom illusion are to maintain the size of the texture elements as constant as possible in screen space, while preserving the enlarging or shrinking of the texture elements required for a convincing feeling of zooming in or out. More generally, the infinite zoom on a signal can be seen as the infinite growth of its frequency. For an 1d signal (i.e. a sound) it corresponds to an endlessly increasing pitch, as demonstrated by Shepard [She64]. For 2D images, Cunzi et al. [CTP+03] and Han et al. [HRRG08] consider this process as the infinite generation of new visible details.

Drawing inspiration from the procedural noise function of Perlin [Per85] and the infinite zoom mechanism of Dynamic Canvas [CTP+03], our method relies on the *fractalization* of a texture. In Dynamic Canvas, a self-similar image is obtained by linearly blending n octaves (doubled frequency) of a texture. When the observer moves forward or backward, the frequencies of the octaves are continuously shifted to produce an illusion of zoom.

We define a *dynamic solid texture* as the weighted sum of n octaves Ω_i of the original solid texture. Each 3D object is then carved in such a solid texture, which naturally ensures a convincing feeling of zooming in and out: the texture will appear twice bigger when the object is twice closer to the camera. But care must be taken to keep the texture elements at a quasi-constant size in screen space. For this, we introduce the notion of *zoom cycle*, that occurs every time the appearant size of the texture doubles. In that case, each octave is replaced by the following one and a new high frequency octave is created, as illustrated in Figure 5.2. Note that the fractalization process introduces new frequencies in the texture, along with a loss of contrast, as discussed in section 4. While reducing the number of octaves limits these artifacts, it also makes the apparition and disparition of texture elements more visible. Empirically, we observed that $n = 4$ octaves is enough to deceive human perception.

2.2 Proposed Algorithm

In practice, an object is embedded in its dynamic solid texture by deriving texture coordinates from vertex coordinates, in the local mesh frame. This object space texturing overcomes the Dynamic Canvas limitation of approximating an entire scene with a plane. For the sake of simplicity, we use only one cube of solid texture that we sample at different rates to retrieve all n octaves.

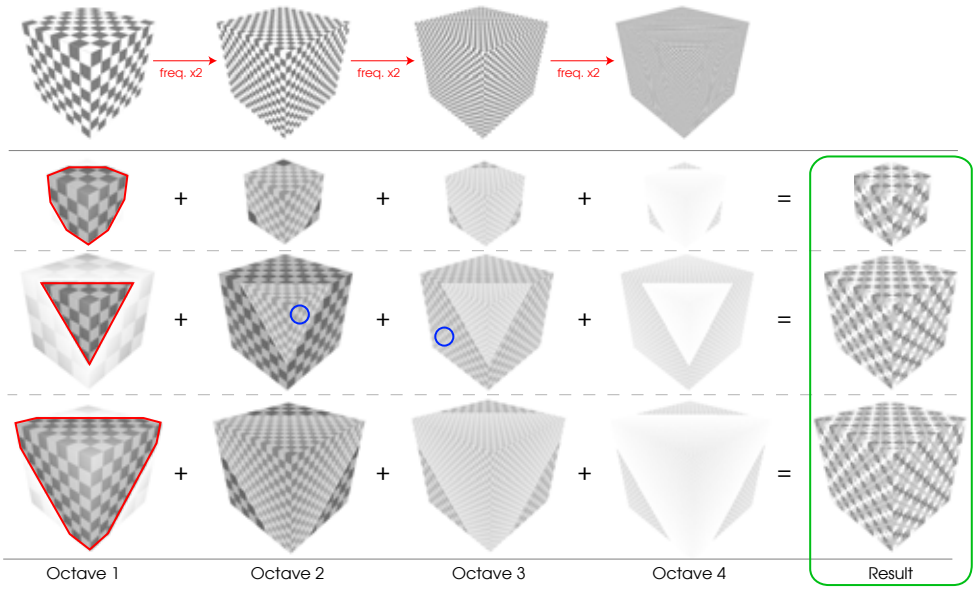


Figure 5.2: *Tridimensional infinite zoom mechanism (using a solid checkerboard texture for illustration purpose). Observe the globally invariant frequency of the solid texture produced by our algorithm (last column). The red line delineates the boundary between two consecutive zoom cycles. Note the frequency similarity between two adjacent octaves for two consecutive zoom cycles (blue circles).*

For a vertex $p(x,y,z)$ at distance z_{cam} from the camera, the 3D texture coordinates (u,v,w) for the octave i is given by:

$$(u,v,w)_i = 2^{i-1}(x,y,z)/2^{\lfloor \log_2(z_{cam}) \rfloor}$$

In this equation, the 2^{i-1} term scales the sampling rate, so that the texture retrieved for one octave is twice smaller than the texture for the previous octave. The $\lfloor \log_2(z_{cam}) \rfloor$ term accounts for the refreshing of the octaves at each zoom cycle: the distance z_{cam} can be decomposed in $log_2(z_{cam})$ zoom cycles, making $\lfloor \log_2(z_{cam}) \rfloor$ the indicator of how many times each octave has been doubled during the zoom.

During a zoom cycle, we modulate each octave with a weight $\alpha_{i=1...n}(s)$, that is dependent on s, the interpolation factor between the beginning and the end of the cycle:

$$s = \log_2(z_{cam}) - \lfloor \log_2(z_{cam}) \rfloor \ \in [0,1]$$

We must impose three constraints on the weights in order to ensure a smooth transition during the frequency shift (Figure 5.3). First, to avoid the sharp appearance/disappearance of texture elements, the first octave should appear at the beginning of the cycle while the last should disappear at its end:

$$\alpha_1(0) = 0 \ \text{ and } \ \alpha_n(1) = 0$$

Second, the weight of the intermediate octaves at the end of the cycle should be equal to the weight of the following octaves at the beginning of the next cycle:

$$\alpha_i(1) = \alpha_{i+1}(0) \quad \forall i \in \{1,\ldots,n-1\}$$

Finally, the weights should sum to 1 to preserve a constant intensity. In our implementation, we use a linear blending – fast to compute and coherent with the linearity of the zoom –, with the following weights:

$$\begin{array}{lcllcl} \alpha_1(s) & = & s/2 & \alpha_2(s) & = & 1/2-s/6 \\ \alpha_3(s) & = & 1/3-s/6 & \alpha_4(s) & = & 1/6-s/6 \end{array}$$

The full process is illustrated in Figure 5.2. The frequency shift, corresponding to the distance after which the zoom cycle restarts, is highlighted by the red line. Note the frequency similarity between the octaves i and $i+1$ for two consecutive zoom cycles. This correspondence ensures, after the blending, the texture continuity (last column).

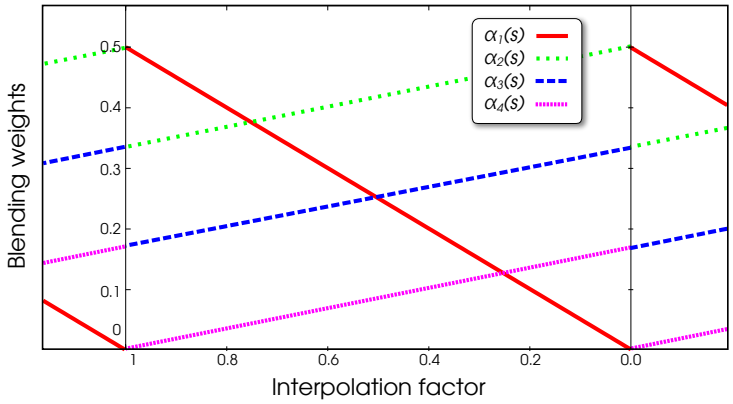


Figure 5.3: *Evolution of the blending weights during a zoom cycle. Note the continuity at the beginning and the end of the zoom cycle, which ensures an infinite number of cycles.*

2.3 Implementation details

We have implemented this infinite zoom algorithm in GLSL[1] and integrated it in the rendering engine OGRE[2]. The *vertex shader* assigns the 3D texture coordinates, which can be the position of the vertices in the local object coordinate system or, for deformable objects, this position in an undeformed pose of the object. The user can also specify an additional scaling factor to define the global size of the texture with regards to the depicted object. We compute the distance between a vertex and the camera as its z coordinate in the camera frame. Then the *fragment shader* blends linearly the four octaves of the solid texture for each pixel, according to the formula detailed in the previous section.

[1] OpenGL Shading Language : http://www.opengl.org/documentation/glsl/

[2] Shader sources are available on the project webpage: http://artis.imag.fr/Publications/2009/BBT09/

We produced the solid textures following two different approaches. We generated some of them procedurally (Perlin noise [Per85, Ola05], for example) using shaders. The more natural and complex textures have been synthesised from 2D exemplars using the algorithm proposed by Kopf et al. [KFCO+07].

2.4 Results

We compare in Figure 5.4 our dynamic solid textures with traditional texture mapping. With a standard (meaning fixed scale) mapping, the size of texture elements varies with depth due to perspective projection. On the contrary, with our infinite zoom approach, texture elements keep a globally constant size in image space, independent of the zoom factor.

(a) Top view (b) Phong shading (c) Traditional 3D texture mapping (d) Dynamic solid texture

Figure 5.4: *Comparison of our dynamic solid textures (d) with standard texture mapping (c). Regardless of the distance from the camera (top and bottom row), our method provides quasi-constant size of texture elements.*

Compared with Dynamic Canvas in Figure 5.5, our approach suffers from perspective deformations when the surface is almost tangential to the viewing direction, and from discontinuities at occlusion boundaries. However, these artifacts are limited by the infinite zoom mechanism that effectively reduces the apparent depth of the scene. On the other hand, as highlighted by the red dot on Figure 5.5, sliding effects occur with the Dynamic Canvas approach. In our case, the texture elements follow perfectly the 3D motion of the scene. Note that the approaches of Coconu et al. [CDH06] and Breslav et al. [BSM+07] would suffer from the same sliding problem.

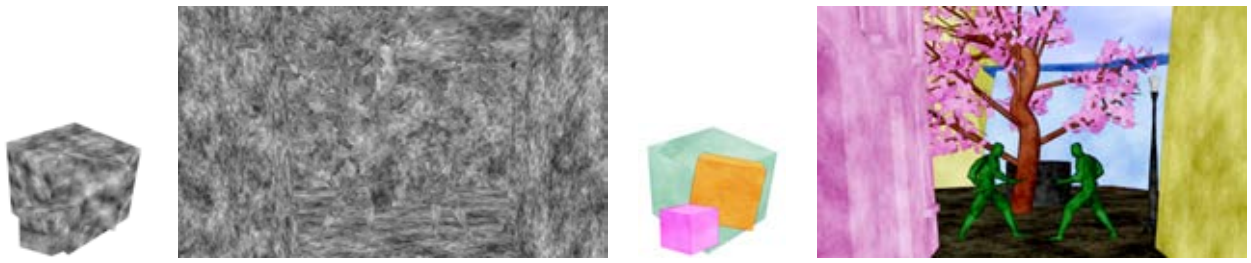
The main advantages of our method are its simplicity of integration in existing rendering engines and its real-time performance. Our implementation in OGRE induces a small additional cost on the order of 10% in comparison with a traditional Gouraud shading (computed in a shader). For the complex scene of Figure 5.6 (135k tris) rendered at a resolution of 1280×1024 pixels, the framerate decrease from 70 fps to 65 fps with a 2.4GHz Core 2 Duo 6600, 4Go memory and a Geforce 8800 GT. This makes dynamic solid textures perfectly suited for real time applications such as video games.

(a) Top view (b) Dynamic Canvas (c) Dynamic solid textures

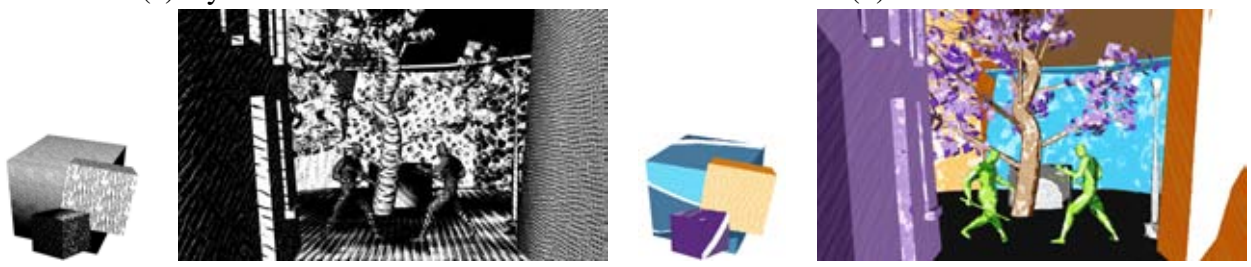
Figure 5.5: *Comparison with Dynamic Canvas: note the sliding of the texture with the Dynamic Canvas method, highlighted by the red dot.*

3 Application to coherent stylization

The technique of dynamic solid textures introduced in this chapter can be used in a variety of rendering methods simply by replacing standard textures. We illustrate this principle with three real-time stylization algorithms (see Figure 5.6 and 5.8). The first describe a watercolor rendering, where we use the dynamic textures to mimic the pigmentation grain in a temporally coherent fashion. We then demonstrate that our approach can be used for binary styles such as hatching, and for an original style that benefits from the variety of textures that our method can combine in one image (collage style).



(a) Dynamic solid texture (b) Watercolor



(a) Binary style (b) Collage

Figure 5.6: *Complex scene (135k tris) rendered with various styles.*

3.1 Watercolor

Watercolor granulation effects can be reproduced by overlaying a graylevel texture over the 3D rendering, so that the lightening and darkening of the original color mimics the variations of pigment density over a canvas. Figure 5.6(b) illustrates the richness of the medium obtained with this approach, using either procedural or synthesized dynamic solid textures. The resulting temporal coherence is best appreciated in a video[3]. Studio Broceliande[4] uses this approach to produce animations that mimic the style of traditional French and Belgium comic books (Figure 5.7).

(a) Blacksad (French graphic novel)

(b) Graphanim rendering based on our algorithm

Figure 5.7: *Our watercolor rendering (b) is used by Studio Broceliande to reproduce the look of French and Belgium graphic novels (a).*

3.2 Binary style

We obtain black-and-white shading styles (pen-and-ink, stippling, charcoal, etc) using a rendering pipeline very similar to the one of Durand et al. [DOM+01]. In this method, strokes are simulated by truncating a *threshold structure* – a grayscale texture seen as a heightfield – at different height with respect to the target tone. We apply a similar threshold on our dynamic solid textures. Figure 5.6(c) illustrates the diversity of binary styles we can combine in one image by simply using a different dynamic solid texture for each object. During the animation, binary strokes appear and disappear progressively thanks to the infinite zoom mechanism.

Note that contrary to existing methods for pen-and-ink stylization [HZ00, PHWF01], our solid texture based approach does not orient the binary strokes along the principal directions of the surface. This limitation of the solid textures is discussed in section 4.

[3] Videos are available on the project webpage: http://artis.imag.fr/Publications/2009/BBT09/

[4] http://www.studio-broceliande.fr/

3.3 Collage

We finally propose a new stylization process that we call *collage*. This new style takes advantage of the diversity of the texture gallery that one can synthesize. Traditional collage consists in creating an image as a composition of several small strips of paper with various colors and textures. We mimic this style by assigning different dynamic solid textures to each tone of the image (obtained by a discretized shading model similar to toon shading [LMHB00]). In order to reinforce the paper aspect of the collage, a white border and a wobbling effect is added between the tone strips (Figure 5.6(d)). The accompanying video illustrates the accurate temporal coherence of the paper strips, which would be difficult to obtain without our dynamic solid textures.

(a) Watercolor (b) Binary style (c) Collage

Figure 5.8: *Simple scene rendered with various styles.*

4 Discussion and Future Work

Infinite zoom mechanism The main limitation of our method, shared with Dynamic Canvas [CTP+03] and to some extent with mipmaps-based approaches [KLK+00,PHWF01,FMS01], is the linear blending of multiple octaves that creates new frequencies and induces a global contrast loss compared with the original texture. Self-similar textures, such as Perlin fractal noise, paper or watercolor textures, are not severely affected by this blending. However, more structured textures can be visually altered, as individual features tend to overlap (the checkerboard texture being an extreme case). We plan to address this limitation in future work. One possible solution would be to replace the linear blending by a salience aware blending in the spirit of the work of Grundland et al. [GVWD06]. This blending should better preserve the relevant features of the texture. The required salience data could be easily encoded in solid *feature maps* [WY04].

Texture mapping We chose to develop the infinite zoom mechanism for solid textures because it avoids the need for an adequate 2D parameterization. However, the flip side of this choice is that textures are decorrelated from the 3D surfaces. This can be seen as a limitation for styles that benefit from the orientation of texture elements along the surface. An example of such styles is pen-and-ink, for which it has been shown that orienting strokes along the principal

directions of the surface emphasizes the shape of the objects [HZ00, PHWF01]. Nevertheless, the infinite zoom mechanism described in this paper is independent of the texture dimension and can also be applied on 2D textures if the required parameterization is available.

A limitation shared by all texture-based approaches concerns the texturing of the highly deformable objects. In this case, defining the texture coordinates as the vertices positions of the undeformed object creates additional dilatation/shrinking of the simulated medium, decreasing the perception of a two-dimensional look.

5 Conclusions

We introduced in this chapter dynamic solid textures, that maintain a quasi-constant size of textures in screen space independently of depth. In the context of non-photorealistic rendering, dynamic solid textures preserve the 2D appearance of style marks while ensuring temporal coherence. In addition, the simplicity and efficiency of this approach facilitates its use in real-time applications such as video games.

In the future we plan to extend the range of applications of our method. As an example, any type of feature line that stays fixed on the object surface (*ridges* [OBS04], *demarcating curves* [KST08]) could be stylized according to a dynamic texture.

Due to its inherent 3D nature, our infinite zoom mechanism cannot be applied on animations that do not contain depth information, such as standard videos. In the next chapter we describe a different method specifically tailored for such inputs, where the apparent 2D motion of a video (the *optical flow*) is used to animate 2D textures.

chapter 6

Bidirectional Texture Advection for Video Watercolorization

The work presented in this chapter and in Chapter 1 was done during an internship at Adobe Advanced Technology Labs in Seattle in 2006. The result of this research has been published at SIGGRAPH 2007 [BNTS07], in collaboration with Fabrice Neyret, Joëlle Thollot and David Salesin. We have presented our work on watercolor rendering as invited speakers at Imagina 2008 and FITA 2008[a].

[a] Forum International des Technologies de l'Animation (Angoulème, France)

While the method proposed in chapter 5 targets the stylization of 3D scenes, we introduce in this chapter a novel method to apply watercolor effects on videos, where depth information is not available. We identify that only the projected motion in screen space is required for convincing temporal coherence, making the full 3D information of the scene unnecessary. From this observation, the primary contribution of our method is to maintain coherence for textures by employing texture advection along lines of optical flow (Figure 6.1). We furthermore extend previous approaches by incorporating advection in both *forward* and *reverse* directions through the video, which allows for minimal texture distortion, particularly in areas of disocclusion that are otherwise highly problematic.

1 Related Work

We review in this section the different approaches that have been proposed to stylized video sequences. In general, the difficulty with extending non-photorealistic rendering techniques from static to moving images is that, without careful consideration to temporal coherence, the resulting animations exhibit one of three problems: the illustration effects remain fixed in place across the image, an unwanted artifact that has become known as the *shower door* effect; or the illustration effects are fixed to the 3D objects and follow their motion, which contradicts the 2D nature of the

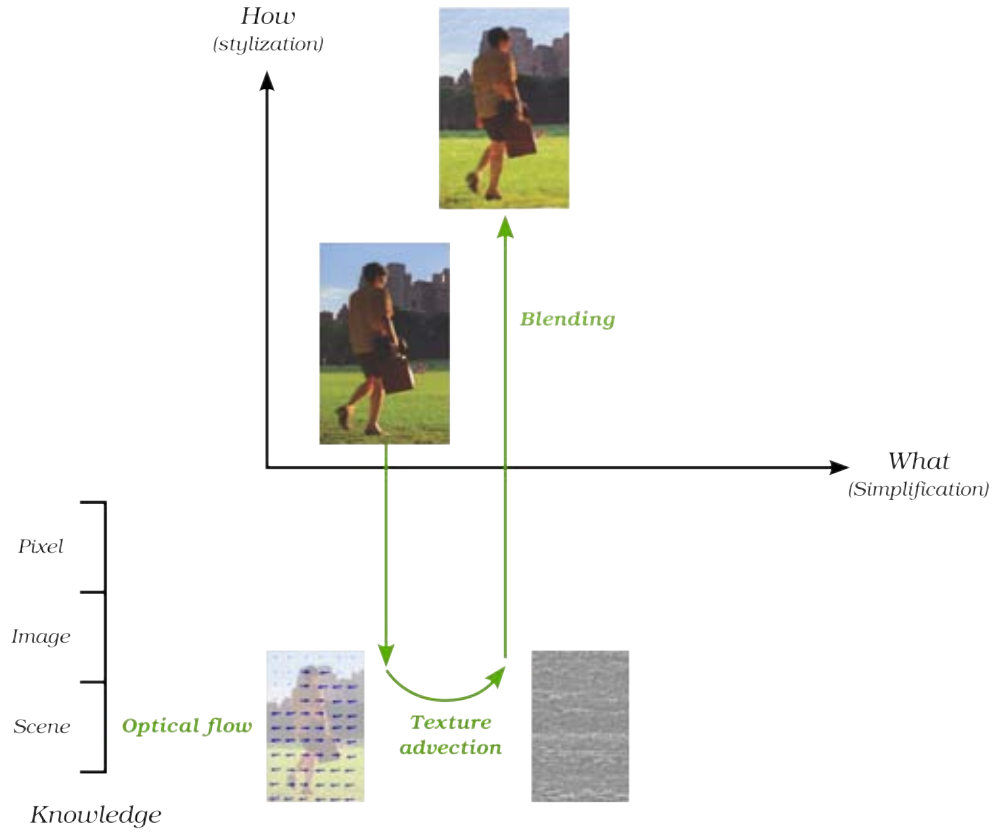


Figure 6.1: *Our method advects a pigmentation texture along an optical flow field in order to ensure temporal coherence of granulation effects in a watercolor rendering.*

medium; or the illustration effects exhibit no temporal coherence whatsoever, randomly changing in position and appearance from frame to frame, which can be even more visually distracting.

Several techniques have been developed to combat these problems in various NPR styles, although most of this work concerns the production of computer-animated 3D scenes, in which reliable information about the coherence of objects from frame to frame is readily available. For watercolorization, these approaches rely on particles distributed over 3D surfaces [LD06, BKTS06] or require texture mapping in 3D [LM01]. Most of the work on video stylization has been in the realm of primitive-based rendering, where discrete paint strokes [Lit97, HP00, HE04, CRH05] or other primitives [WJFC02] are applied to create the stylization. Applying such method to watercolor raises the difficult question of finding a set of 2D primitives that, once combined, produces a continuous texture (typically pigments or paper).

As demonstrated in the previous chapter, texture-based approaches are well suited to create animations where a global continuous texture evolves across the animation frames in a temporally coherent fashion. Taking inspiration from the *Dynamic Canvas* [CTP+03] and our *Dynamic Solid Texture* approach (Chap. 5), our video stylization performs a dynamic adaptation of the stylization textures, while at the same time tracking dynamic object motions in a changing video

scene. Our work also shares some similarity to Fang's *RotoTexture* [Fan06], in that both attempt to provide textures that track dynamic scenes; however, Fang's work is concerned with maintaining the appearance of 3D textures rather than 2D. Other work in scientific visualization [MB95, JEH01], fluid simulation [Sta99, Ney03], and artistic image warping [Sim92] shares the goal of evolving texture along a 2D vector field. Our work builds on the advection approach that these schemes introduced.

2 Texture advection

The "granulation" of watercolor pigments provides a large share of the richness of continuous tone techniques. However, in order to create effective watercolor animations, the pigmentation texture P must satisfy three competing constraints: first, it must maintain its appearance in terms of a more or less homogeneous distribution and frequency spectrum. Second, it must follow the motion in the scene to avoid the "shower door" effect. Finally, it must not change abruptly to avoid popping.

To resolve this conflict we build on previous work on advected textures [MB95, Ney03], classically used to depict flow in scientific visualizations. The general idea of such methods is to initialize the texture mapping on the first frame of an animation, and then evolve the mapping with the motion flow. In these methods the texture mapping is reinitialized whenever the statistical spatial properties of the current texture become too dissimilar to the original one.

We employ this same basic idea to our situation of applying texture to video, substituting optical flow for fluid flow. One significant complication, which arises quite frequently for videos but not for continuous fluid flows simulations, is the occurence of *disocclusion boundaries*: places where new pixels of the background are revealed as a foreground object moves across a scene. Optical flow fields at disocclusions are essentially undefined: there is no pixel in the prior frame corresponding to these disoccluded boundary regions. Classical advected textures are designed for handling only continuous space-time distortions. In the absence of continuity, they tend to fail quite badly, as we show in Figure 6.3.

In order to handle disocclusion boundaries effectively, we introduce the notion of *bidirectional advection*: simultaneously advecting two texture layers in opposite directions in time — from the first frame of the video to the last, and from the last frame to the first. We use a combination of these two texture layers weighted at each pixel by the local quality of the texture from each direction. In the rest of this section we describe our algorithm and its propeties.

2.1 Advection computation

In the following, we will use $\mathbf{x} = (x, y)$ for screen coordinates, and $\mathbf{u} = (u, v)$ for texture coordinates. An advected texture relies on a field of texture coordinates $\mathbf{u}(\mathbf{x}, t)$, which is displaced following a vector field $\mathbf{v}(\mathbf{x}, t)$: for any frame t, the vector $\mathbf{u}$ defines the location within the texture image P_0 to be displayed at position $\mathbf{x}$, i.e. the *mapping*. For simplicity we assume that

$\mathbf{x}$ and $\mathbf{u}$ coordinates are normalized on the interval $[0,1]$ and that $\mathbf{u}(\mathbf{x},0) = \mathbf{x}$. Our vector field is obtained from an optical flow extracted from the video (we rely on a classical gradient-based method available in Adobe After Effects). The vector $\mathbf{v}$ indicates for each frame t where the pixel at position $\mathbf{x}$ came from within frame $t-1$: $\mathbf{v}(\mathbf{x},t) = \mathbf{x}_{t-1} - \mathbf{x}_t$.

The purpose of advection is to "attach" the texture P_0 to the moving pixels of the video, which is theoretically done by displacing the texture coordinates according to the vector field[1]: $\mathbf{u}(\mathbf{x},t) = \mathbf{u}(\mathbf{x}+\mathbf{v}(\mathbf{x},t),t-1)$. However, this backward mapping is problematic wherever the optical flow is poor or ill-defined, as at disocclusion boundaries. We will discuss how these problematic cases are handled.

2.2 Controlling the distortion

With this basic approach, the distortion of the advected texture increases with time. To conceal this distortion, Max and Becker's [MB95] and Neyret's [Ney03] approaches blend two or three phase-shifted texture layers, respectively (See Figure 6.2(a)). In both schemes, the distortion is reset periodically (i.e., $\mathbf{u}(\mathbf{x},\tau) = \mathbf{u}(\mathbf{x},0)$), allowing the original texture to be used again. The regeneration period τ is chosen via a user defined delay [MB95] or a dynamic estimation of the distortion [Ney03]. The fact that the regeneration occurs periodically guaranties that our system can handle videos of arbitrary length.

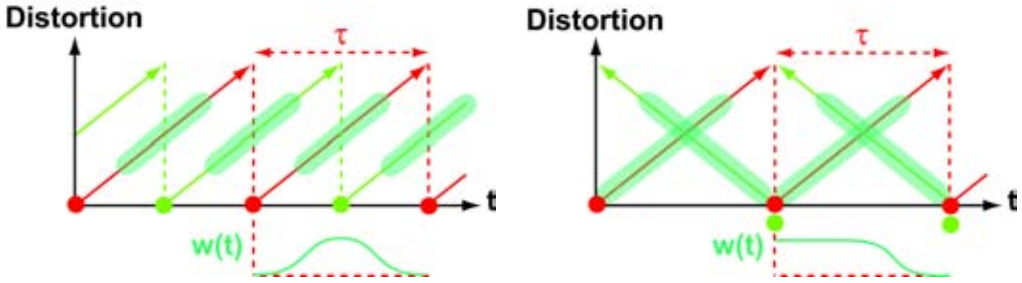
Like Max and Becker, we rely on two texture layers, but we combine one "forward" mapping $\mathbf{u}_f$, advected from the beginning to the end of the video sequence, with one "reverse" mapping $\mathbf{u}_r$, advected from the end to the beginning (Figure 6.2(b)). The final advected pigment texture P' is a combination of these two fields, calculated on a per-pixel basis by taking into account the local distortions ω_f and ω_r of the forward and reverse mappings $\mathbf{u}_f$ and $\mathbf{u}_r$, respectively, within a given frame:

$$P'(\mathbf{x},t) \;=\; \omega_f(\mathbf{x},t)P_0\big(\mathbf{u}_f(\mathbf{x},t)\big) \;+\; \omega_r(\mathbf{x},t)P_0\big(\mathbf{u}_r(\mathbf{x},t)\big)$$

We will show precisely how these distortion measures ω_f and ω_r are computed later on. For now, to understand the intuition behind this approach, it suffices to note that texture distortion gradually increases in the direction of advection. Since $\mathbf{u}_f$ is advected forward, its distortion increases with time. However, since $\mathbf{u}_r$ is advected backwards through the video sequence, its distortion *decreases* with time. The combination of the two textures can therefore be used to create a less distorted texture. Moreover, a disocclusion in the forward sequence becomes an occlusion in the reverse, which is no longer a source of distortion; thus, a well-defined texture can always be used. (A similar observation was noted by Chuang et al. [CAC+02] in their work on video matting.)

The technical challenges of this approach are to quantify the visual distortions and to choose a clever way of combining the two advected fields. Three competing goals have to be taken into account: (1) minimizing the distortion; (2) avoiding temporal discontinuities; and (3) limiting

[1] In order to manage boundary conditions properly, we assume that the texture $P_0(u)$ is periodic, and that $\mathbf{u}(\mathbf{x},t) \approx \mathbf{u}(\mathbf{x},t-1) + \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \cdot \mathbf{v}(\mathbf{x},t)$ whenever $\mathbf{x}+\mathbf{v}(\mathbf{x},t)$ seeks outside $[0,1]^2$ in the spirit of [MB95].



(a) Max and Becker scheme (b) Our scheme

Figure 6.2: *Approaches to controlling the distortion of the advected texture. (a) Max and Becker scheme [MB95] uses two phase-shifted, overlapping texture fields at any given time. (b) Our scheme also uses two texture fields at a time but advected in opposite time directions. In the two diagrams, the green solid areas represents, roughly, the relative contribution of each advected layer to the final texture's appearance (weight $w(t)$). Note that in prior work (a), in order to maintain temporal coherence, the advected layers do not begin contributing significantly until they are already somewhat distorted. In our method (b), by contrast, textures contribute maximally where they are least distorted. Blending textures advected in opposite time directions also allows for less distortion everywhere, since distortion is decreasing in one texture just as it is increasing in the other. Finally, bidirectional advection handles disocclusion boundaries much better since it corrupts only one of the layers.*

visual artifacts such as contrast variation. In the rest of this section, we detail our method: how we quantify the distortion (Section 2.3), and how we adjust the weights of the two advected fields (Section 2.4).

2.3 Distortion computation

We need a way to estimate the visual quality of a distorted texture at each pixel. Various methods exist to compute the distortion of a non-rigid shape. The general principle is to compute a deformation tensor and to choose an appropriate norm to compute the distortion.

All deformation tensors are based on the deformation gradient tensor F, corresponding to the Jacobian matrix of the shape coordinates (in our case the texture coordinates): $F_{ij}(\mathbf{x},t) = \partial \mathbf{u}_i(\mathbf{x},t)/\partial x_j$. As in previous work, we do not wish to consider translations and rotations to be distortions because they do not alter the visual properties of the texture. Instead, it is typical to rely on a strain tensor, which cancels antisymmetric components. However, unlike Neyret [Ney03], we wish to deal with large displacements, so we cannot use the infinitesimal strain tensor, which is the classical approximation. We therefore choose the Cauchy-Green tensor: $G = F^T F$ (one can verify that multiplying F by its transpose cancels the rotations). The eigenvectors of G give the principal directions of deformations, and the tensor's eigenvalues λ_i give the squares of the principal deformation values: an eigenvalue $\lambda_i > 1$ corresponds to a stretch, whereas an eigenvalue $\lambda_i < 1$ corresponds to a compression.

We want to derive an estimation of the visual quality of the *distortion* ξ as a scalar in $[0,1]$ with 0 representing no distortion, and 1 representing distortion that is intolerable. We assume that compression and stretching are equally bad from a visual standpoint. We therefore define the *visual deformation* in the two principal directions σ_1 and σ_2 as: $\sigma_i(\mathbf{x},t) = \max\left(\sqrt{\lambda_i(\mathbf{x},t)}, 1/\sqrt{\lambda_i(\mathbf{x},t)}\right)$.

We define the visual distortion ξ as the quadratic mean of both deformations, normalized to $[0,1]$:

$$\xi(\mathbf{x},t) \quad = \quad \frac{\xi'(\mathbf{x},t) - \xi_{min}}{\xi_{max} - \xi_{min}}$$

where $\xi'(\mathbf{x},t) = \sqrt{\sigma_1^2(\mathbf{x},t) + \sigma_2^2(\mathbf{x},t)}$ is an unnormalized scalar measure of the visual distortion; $\xi_{min} = \sqrt{2}$ is the minimum value of ξ' (representing no distortion); and ξ_{max} is a maximum bound over which the distortion is considered too high, measured experimentally. We use $\xi_{max} = 5$.

2.4 Adjusting weights

Given the distortions ξ_f and ξ_r of each advected mapping $\mathbf{u}_f$ and $\mathbf{u}_r$, we must now find the appropriate set of weights ω_f and ω_r at each pixel in order to minimize the final distortion. Our weights must satisfy a number of properties: They should lie on the interval $[0,1]$; sum to 1 at every pixel; be inversely related to the texture distortion; gradually decrease to zero at regeneration events in order to maintain overall temporal continuity; and vary smoothly, both in space and in time. We choose the following definition for the weights, which satisfy all of these properties:

$$\omega_f(\mathbf{x},t) = \frac{\omega'_f(\mathbf{x},t)}{\omega'_f(\mathbf{x},t) + \omega'_r(\mathbf{x},t)} \qquad \omega_r(\mathbf{x},t) = \frac{\omega'_r(\mathbf{x},t)}{\omega'_f(\mathbf{x},t) + \omega'_r(\mathbf{x},t)}$$

with

$$\omega'_f(\mathbf{x},t) = g_f(\mathbf{x},t)\, h_f(t) \qquad \omega'_r(\mathbf{x},t) = g_r(\mathbf{x},t)\, h_r(t)$$

Where g_f and g_r are measures of the distortions of the forward and reverse textures relative to the other:

$$g_f(\mathbf{x},t) = \frac{1 - (\xi_f - \xi_r)}{2} \qquad g_r(\mathbf{x},t) = \frac{1 - (\xi_r - \xi_f)}{2}$$

and h_f and h_r are temporally decaying weighting functions:

$$h_f(t) = \cos^2\left(\frac{\pi}{2}\,\frac{t \bmod \tau}{\tau}\right) \qquad h_r(t) = \sin^2\left(\frac{\pi}{2}\,\frac{t \bmod \tau}{\tau}\right)$$

where τ is the delay between two regenerations.

Figure 6.3 shows the resulting advected texture for a test scene. If the distortion rate is regular the blending behaves like the one of Max and Becker [MB95]; however, when the distortion rate is high, for example at disocclusion boundaries, our blending results are much better.

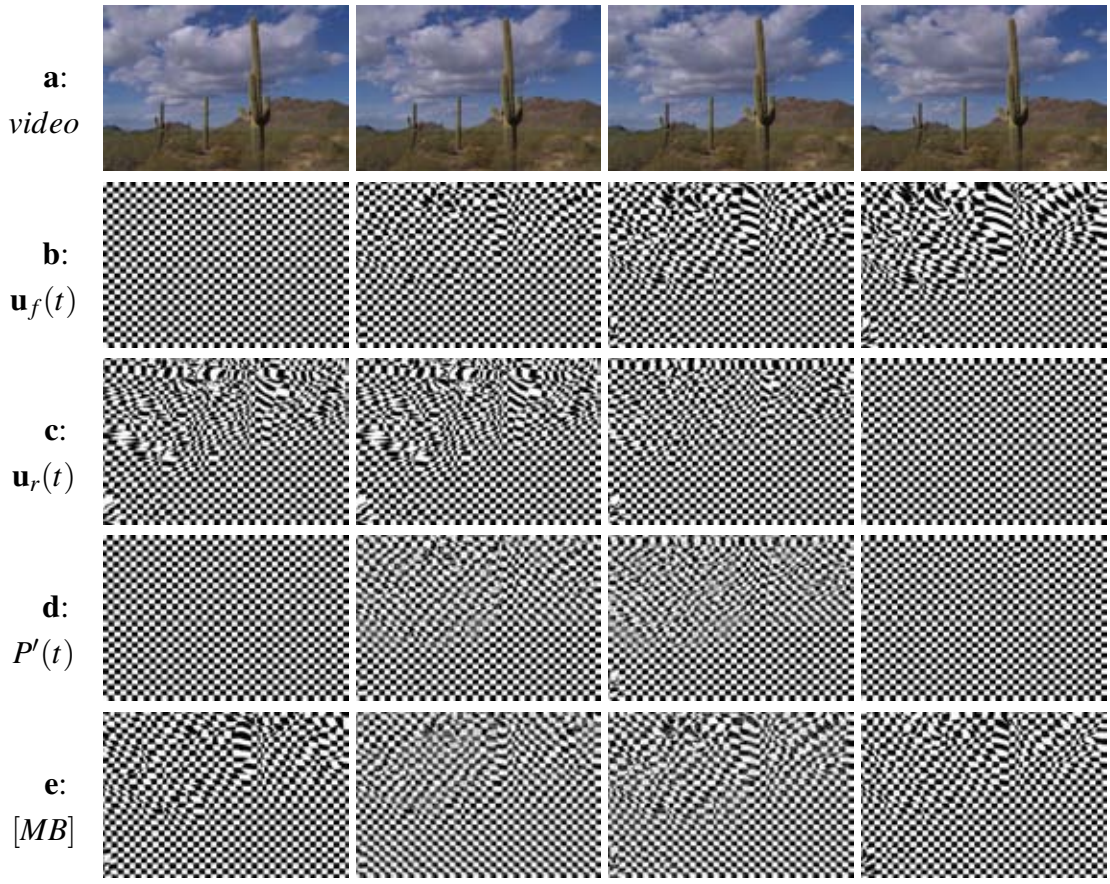


Figure 6.3: *Analysis of our bidirectional advected texture. (a) Original video sequence. (b) A checkerboard texture (for illustrative purpose), advected forward using optical flow. (c) The same texture, advected in the reverse time direction using reverse optical flow. (d) The combined, bidirectional advected texture. (e) Advected texture using the previous approach of Max and Becker. Note how the bidirectional advected texture in (d) shows much less distortion in all frames than the previous approach in (e).*

2.5 Limiting contrast oscillation and tuning τ

The blending of the two advected layers produces a cycle between frames showing one single texture ($\omega_f = 1$ or $\omega_r = 1$) and frames with two blended textures (for all other values of ω_f and ω_r). Contrast is reduced, especially as ω_f and ω_r approach the same value of $1/2$, because high frequencies are dimmed. To reduce the perception of the resulting oscillation of contrast, we include a second pair of advected layers with a regeneration cycle halfway out of phase. The average of these two advected textures gives a nearly constant amount of contrast. As two out-of-phase identical textures moving in the same direction may produce spatial correlation (i.e., ghosting), we use a visually similar but different texture image for this pair. Thus, our complete model relies on four layers.

In previous methods, the regeneration period τ is chosen as a tradeoff between texture distortion and loss of motion perception (if the texture is regenerated too often there is no longer advection). As pointed out by Neyret [Ney03], the ideal value of τ is generally not the same in each region of the image, depending on the local velocity. Thus, Neyret proposes an adaptive regeneration scheme, which consists in advecting several texture layers with increasing regeneration periods, and locally interpolating between layers to minimize the distortion. In this way, fast regeneration occurs only in regions of high distortion. Our method already has some spatial adaptation. Still, this techniques is also applicable in our case since velocity varies in the optical flow. Being able to choose between various periods also helps to minimize the distortion due to disocclusion. In practice, we found that two pairs of layers were sufficient in most cases. One fast layer ($\tau = 15$ frames) allows for good correction of disocclusions, while a slower layer ($\tau = 30$ or 60 frames) provides good motion perception in slow motion areas. The fact that the regeneration occurs periodically guaranties that our system can handle videos of arbitrary lenght.

3 Results and discussion

Figure 6.4 illustrates the of result of our method, composited over an abstracted video obtained with the method described in Chapter 1. Our results are best viewed in the accompanying videos [2] that demonstrates the effectiveness of our advection scheme when dealing with complex motions and occlusion boundaries (Figure 6.5).

We have implemented our method as an Adobe After Effects plug-in. Our performance statistics are dependent on the whole system, and our algorithms have not been optimized. Currently, advection takes 5 seconds per frame.Computation could be sped up considerably using graphics hardware, as the current bottlenecks are mainly texture access and bilinear interpolation.

The quality of the texture advection is limited by the quality of the optical flow. Errors in the optical flow lead to "swimming" artifacts in the pigmented textures. Such errors are especially visible in relatively unstructured areas of the scene, particularly near occlusion boundaries. Other kind of motion representations, such as motion layers [WA94] could correct these artifacts in some situations, but we believe that the vector field representation can handle motions that other representations cannot (e.g., zoom, large parallax on the same object). We have also applied our advection scheme on a computer generated sequence which shows how the advection can accurately depict complex motions and occlusions giving a correct motion flow. When dealing with such 3D scenes, our texture advection method corrects the remaining perspective distortions that are not handled by the *Dynamic Solid Textures* introduced in chapter 5. The downside is that bidirectional texture advection requires the knowledge of the entire animation, preventing the use of this approach for real-time applications. Comparing the results of texture advection and dynamic solid textures also reveals that once animated and stylized, both methods produce very similar results. In particular, the motion of the texture elements – perceived and interpreted as tridimensional by the observer – tends to conceal their 2D characteristics during animation.

[2] Videos are available on the project webpage: http://artis.imag.fr/Publications/2007/BNTS07/

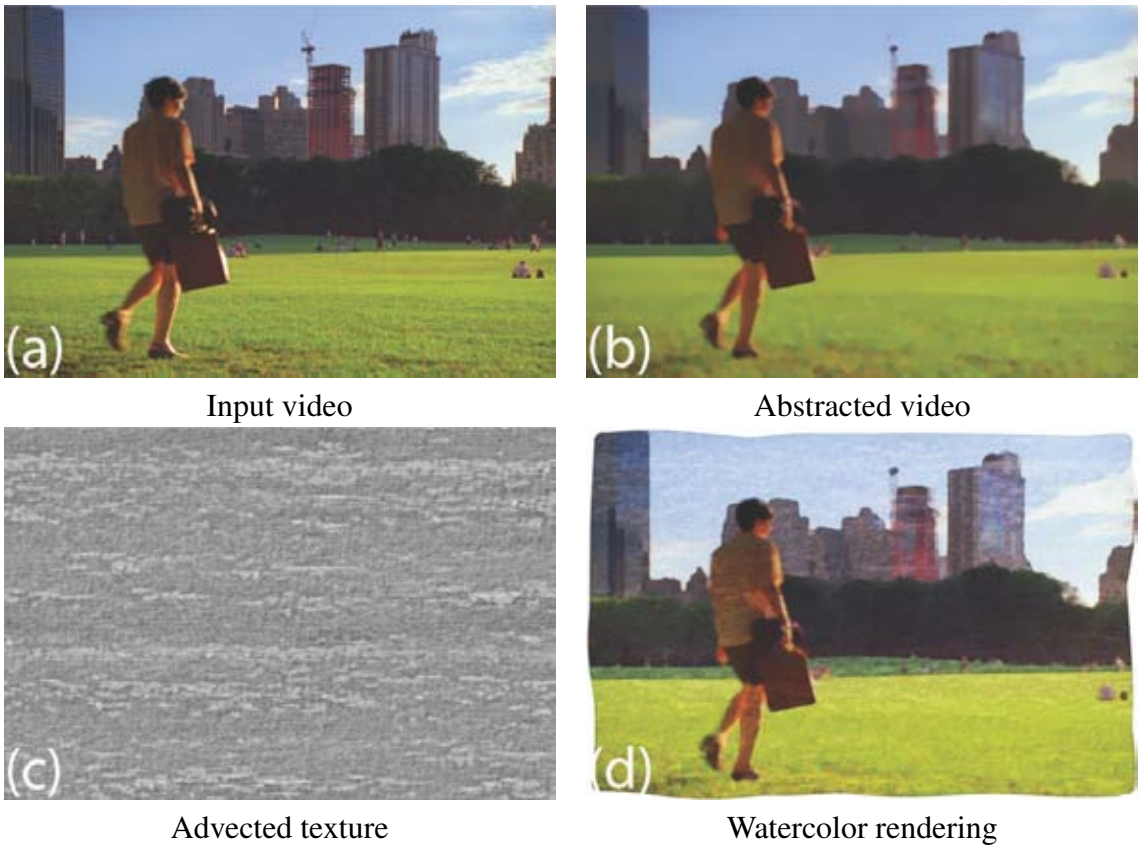


Figure 6.4: *Watercolor compositing. The final watercolor rendering (d) is obtained from an input video (a) as the compositing of the abstracted video (b) and the advected pigmentation texture (c).*

Figure 6.5: *Final results. Note the absence of distortions, even in the presence of motion boundaries (left) or complex motions (right).*

4 Conclusions

We have presented an approach to video watercolorization that provides good temporal coherence given good optical flow. By extending texture advection to handle the types of distortions commonly found in video sequences, we are able to produce computer-generated watercolor animations that are convincing for most scenes.

Although the proposed method has been developed keeping the specific characteristics of watercolor in mind, our texture advection scheme could be applied to other texture-based styles, similarly to the binary and collage styles demonstrated in chapter 5. In our experiments so far, we have found that the more "structured" the appearance of the medium's effects, the more objectionable any distortions due to incorrect optical flow appears. The non-rigid transformations induced by an optical flow also produce a "fluid" animation that is effective for wet medium like watercolor, but less convincing for dry medium such as charcoal. These artifacts might be ameliorated by decreasing the frame rate, as in, for example, much of Bill Plympton's animation[3].

[3] http://www.plymptoons.com/

Part IV

Manipulating Reflectance and Lighting in Photographs

In this part we explore the extraction of scene level information to facilitate further photograph editing. We focus on the decomposition of reflectance and illumination from a single photograph. This is an ill-posed problem that remains challenging for automatic methods, which motivates our user-assisted approach that allows fine decomposition from a limited number of user strokes. Thanks to this decomposition, the illumination and reflectance components of a photograph can be editied separately, which we illustrate with examples of re-texturing and relighting.

chapter 7

User Assisted Intrinsic Images

This research was initiated while I was a visiting student at MIT CSAIL Cambridge in 2008. The results of this collaboration with Sylvain Paris and Frédo Durand has been published at SIGGRAPH ASIA 2009 [BPD09].

(a) Input (b) User scribbles (c) Reflectance (d) Illumination (e) Re-texturing

Figure 7.1: *Starting from a single photograph (a), the user indicates a few fully lit pixels ((b), white scribbles), pixels sharing a similar reflectance ((b), blue scribbles), and pixels sharing similar illumination ((b), red scribbles). From these indications, our system estimates the reflectance (c) and illumination (d) components of the image. This decomposition facilitates advanced image editing such as re-texturing (e).*

Lighting and materials in the scene are critical pieces of information in an image, but are not always easily controllable by the photographer during capture. The ability to manipulate these components once the photograph has been taken is the goal of many computational photography applications, including relighting, material alteration and re-texturing. Unfortunately, in a photograph, illumination and reflectance are conflated through complex interaction and the separation of those components, called *intrinsic images* [BT78], has long been an open challenge. A pixel

can be seen as the per-color-channel product of an *illumination* component, also called *shading*, and a *reflectance* component, also called *albedo*. Given a single image, the problem is severely ill-posed: a dark-yellow pixel can come from, e.g. a white material illuminated by a dark-yellow light, or from a dark-yellow material illuminated by a bright white light.

In this chapter, we introduce a new image decomposition technique that relies on sparse constraints provided by the user to disambiguate illumination from reflectance. Figure 7.1(b) illustrates these indications that can correspond to pixels of similar reflectance, similar illumination, or known illumination.

Central to our technique is a new propagation method that estimates illumination from local constraints based on a few assumptions on image formation and reflectance distributions. In particular, we reduce the number of unknowns by assuming that local reflectance variations lie in 2D subspaces of the RGB color space. Although this simplification requires color images and cannot handle cases such as a black-and-white checkerboard texture, we show that it can handle a broad class of images with complex lighting. In order to enable fast and accurate solution, we also introduce a novel down-sampling strategy that better preserves the local color distributions of an image and enables rapid multigrid computation. Although the main contribution of our method resides in the extraction of the reflectance and illumination components of an image, we illustrate the benefit of this decomposition with image manipulations including reflectance editing and re-lighting. Figure 7.2 visualizes such realistic to realistic image manipulations, where the illumination and reflectance information can be seen as two layers of the scene level.

1 Related Work

The decoupling of reflectance from illumination was introduced by Barrow and Tenenbaum [BT78] as *intrinsic images*. The *reflectance* describes how an object reflects light and is also often called *albedo*. The *illumination* corresponds to the amount of light incident at a point (essentially irradiance). Although it is often refered to as *shading*, it includes effects such as shadows and indirect lighting.

Physically-based inverse rendering, such as Yu and Malik [YM98], seeks to invert the image formation model in order to recover the lighting conditions of a scene, but requires known geometry.

Using several images of the same scene under different illuminations, Weiss [Wei01] proposes a method to estimate a reflectance image along with the illumination map of each input image. This approach has then been extended by Liu et al. [LWQ+08] to non-registered image collections in order to colorize black-and-white photographs. The use of varying illumination has also been applied to shadow removal in a flash/no-flash setup by Agrawal et al. [ARC06].

Due to its inherent ill-posedness, decomposition of intrinsic images from a single image cannot be solved without prior knowledge on reflectance and illumination. Based on the Retinex theory [LM71], Horn [Hor86] assumes that reflectance is piecewise constant while illumination is smooth. This heuristic allows for the recovery of a reflectance image by thresholding the small

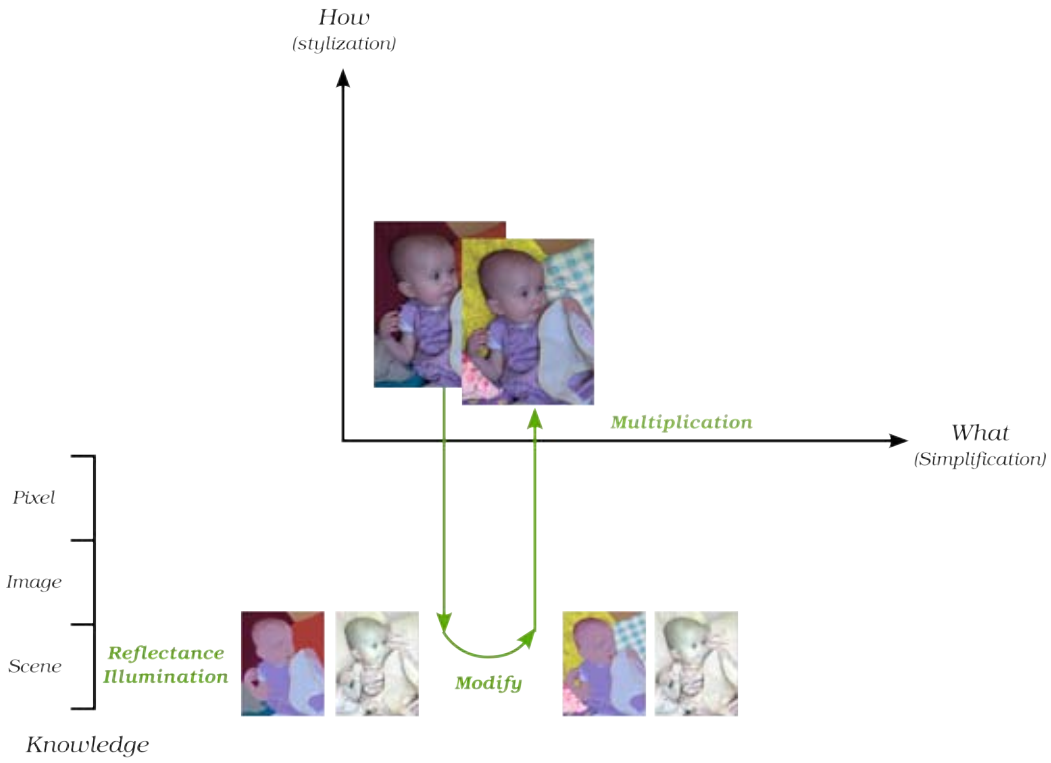


Figure 7.2: *Our method extracts the reflectance and illumination components of an image, which corresponds to two layers of the scene. This high level knowledge of the depicted content can be used to modify scene characteristics such as textures.*

image gradients, assumed to correspond to illumination. Sinha and Adelson [SA93] discriminate illumination from reflectance edges based on their junctions in a world of painted polyhedra: T junctions are interpreted as reflectance variations, while arrow and Y junctions correspond to illumination. Tappen et al. [TFA05] rely on a classifier trained on image derivatives to classify reflectance and illumination gradients. Despite these heuristics and classifiers, many configurations of reflectance and illumination encountered in natural images remain hard to disambiguate. Shen et al. [STL08] propose to enrich these local approaches with non-local texture constraints. Starting from a Retinex algorithm, their texture constraints ensure that pixels that share similar texture will have the same reflectance.

A large body of work has been proposed for the specific problem of shadow removal, either automatically [FHD02,FDL04] or based on user interaction [MTC07,WTBS07,SL08]. The common idea of these methods is to identify shadow pixels, either via boundary detection or region segmentation. Once shadows are detected, they can be removed by color correction or gradient domain filtering. These methods focus on cast shadow removal with clear boundaries, while we also target the removal of smooth shading where the boundary between lit and shaded regions cannot be delimited. Note that although the approach of Finlayson et al. [FHD02, FDL04] relies on the estimation of an illumination-free image, this image is grayscale and does not represent

the true reflectance.

Intrinsic images decomposition is also related to other image decompositions. Matting algorithms [CCSS01, LLW08] aim to separate the foreground and background layers of an image along with their opacity based on user indications. User-assisted approaches have been proposed to separate reflections from a single image [LW07]. Automatic decompositions have been introduced to restore pictures corrupted by haze [Fat08], and to perform white balance of photographs taken under mixed lighting [HMP+08]. Although all these methods do not directly target the extraction of illumination from a single image, our energy formulation is inspired by the *matting Laplacian* used in the work of Levin et al. [LW07] and Hsu et al. [HMP+08]. We rely on a similar assumption that, in natural images, material colors lie on subspaces of the RGB space [OW04].

2 Overview

Our decoupling of illumination and reflectance is based on user-provided constraints and a new propagation model. The user can use sparse strokes to mark parts that share the same reflectance or where illumination does not vary, in particular at material boundaries. The user also needs to provide at least one fixed illumination value to solve for global scale ambiguity.

Based on the assumption that reflectance values are low-rank in local windows, we derive a closed-form equation that depends only on illumination. Adding the user constraints, we can solve a linear least-square system that provides the illumination. Reflectance is simply inferred by a division.

Our least-square solution works best with local windows of medium sizes, which raises computational challenges. We have found that standard multigrid approaches yield unsatisfactory results in our case, because traditional downsampling operators do not respect local color distribution. This is why we introduce a new downsampling scheme for techniques like ours that rely on local color distributions. We show that it enables dramatic speedup and achieves high accuracy.

3 Reflectance-Illumination Decomposition

We first detail our assumptions about the observed image, in particular the fact that reflectance colors are locally planar. We show how it leads to a quadratic energy where illumination is the only unknown, and where user constraints can easily be included.

Image Formation Model As commonly done, we assume that the interaction between light and objects can be described using RGB channels alone. We focus on Lambertian objects and a single light color, although we show later that these hypotheses can often be alleviated in practice, in particular to handle colored light.

With this model, the observed color at a pixel is:

$$\mathbf{I} = s\,\mathbf{L} * \mathbf{R} \tag{7.1}$$

where s is the *illumination*, a non-negative scalar modeling the incident light attenuation due to factors such as light travel, occlusion and foreshortening, $\mathbf{L}$ is the RGB color of the light, $*$ denotes per-channel multiplication, and $\mathbf{R}$ is the material RGB *reflectance* that describes how objects reflect light. For clarity, we assume that the light is white, (or equivalently that the input image is white balanced). This means $\mathbf{L} = (1,1,1)^\mathsf{T}$ and Equation 7.1 becomes:

$$\mathbf{I} = s\mathbf{R} \tag{7.2}$$

3.1 Low-Rank Structure of Local Reflectance

Our objective is to retrieve the illumination s and the RGB components of $\mathbf{R}$ at each pixel. The difficulty is that Equation 7.2 provides only three equations, one per RGB channel. In addition, there is a scale ambiguity between illumination and reflectance, that is, if $\mathbf{R}_0$ and s_0 are a valid decomposition, then $k\mathbf{R}_0$ and $\frac{1}{k}s_0$ are also valid for any scalar factor $k > 0$.

We overcome the ill-posedness of the problem by a local low rank assumption on the reflectance colors. We are inspired by a variety of recent studies that show structure and sparsity in the distribution of colors in natural scenes. In particular, Omer and Werman [OW04] show that the set of reflectance colors is sparse, which led to practical matting [LLW08] and white-balance [HMP+08] techniques.

We build on this strategy and assume that, locally, the reflectance colors are low rank. Specifically, we assume that they lie in a 2D plane that does not go through the origin (black) in RGB space (Figure 7.3). They need not span a full plane and can, for example, consist of a single color or a color line. We acknowledge that this restriction prevents us from treating cases where only luminance variations occur, for example a black-and-white checkerboard, but it enables a practical algorithm that achieves satisfying results on a broad range of natural scenes as demonstrated by our results. In particular, this configuration encompasses cases such as colored textures, constant color objects (1 point in RGB space), edges between two objects (2 points), and t-junctions (3 points).

From these hypotheses, locally, there exists a 3D vector $\mathbf{a}$ such that the reflectance values satisfy:

$$\mathbf{a} \cdot \mathbf{R} = 1 \tag{7.3}$$

Using Equations 7.2 and 7.3, we get

$$\mathbf{a} \cdot \mathbf{I} = s. \tag{7.4}$$

We have turned the original equation where the unknowns are multiplied together by an equation that is linear in the unknowns $\mathbf{a}$ and illumination. Below, we further show that we can eliminate $\mathbf{a}$ and directly solve for s. Note also that there is no scale ambiguity anymore since $\mathbf{R}_0$ and $k\mathbf{R}_0$ cannot both be part of the same reflectance plane unless $k = 1$ (Eq. 7.3).

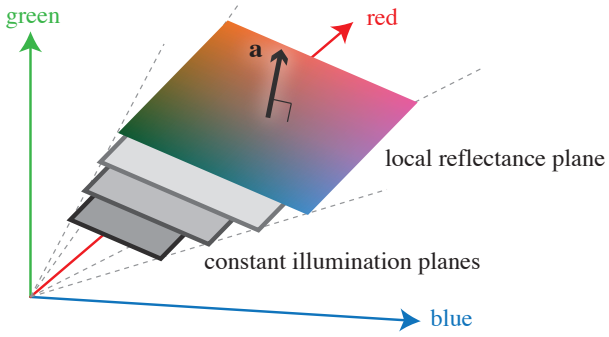


Figure 7.3: *Planar reflectance assumption. We assume that reflectance variations lie locally in a plane in color space. Pixels in planes parallel to the reflectance have constant illumination (Eq. 7.3).*

3.2 Reduction to Illumination Alone

We now seek to eliminate $\mathbf{a}$ and obtain an equation on s alone. We will then solve this equation in a least-squares fashion to account for user constraints and model deviations. We follow an approach inspired by the work of Levin et al. [LLW08] in the context of image matting. In a nutshell, we apply our low-rank assumption to all local windows (of e.g. 5×5 pixels) and assume that $\mathbf{a}$ is a constant over a window. We seek to minimize $(s - \mathbf{a} \cdot \mathbf{I})^2$ at each pixel of a window. Note that a pixel participates in many windows and that the unknowns are shared between those local energy terms. This overlap between windows is what enables information propagation. We can eliminate $\mathbf{a}$ from the equations because of the redundant information from the pixels in a window.

Energy Based on Local Windows For a pixel i, we formulate the following energy over its neighboring pixels in a local window $\mathcal{W}_i$:

$$\tilde{e}(s, \mathbf{a}_i) = \sum_{j \in \mathcal{W}_i} (s_j - \mathbf{a}_i \cdot \mathbf{I}_j)^2 \tag{7.5}$$

We add a regularizer to this energy so that the minimum is always well defined. Indeed, if there is a vector $\mathbf{b}$ such that $\mathbf{b} \cdot \mathbf{I}_j = 0$ for all $j \in \mathcal{W}_i$, then we have $\tilde{e}(s, \mathbf{a}_i) = \tilde{e}(s, \mathbf{a}_i + k\mathbf{b})$ for any real k. Such ambiguity occurs in common cases such as objects with constant reflectance $\mathbf{R}_0$ for which any $\mathbf{b}$ orthogonal to $\mathbf{R}_0$ yields $\mathbf{b} \cdot \mathbf{I}_j = \mathbf{b} \cdot s_j \mathbf{R}_0 = 0$. We address this with a regularizing term:

$$e(s, \mathbf{a}_i) = \sum_{j \in \mathcal{W}_i} (s_j - \mathbf{a}_i \cdot \mathbf{I}_j)^2 + \varepsilon \mathbf{a}_i^2 \tag{7.6}$$

where we choose ε small so that it has an influence only in ambiguous cases ($\varepsilon = 10^{-6}$ in our implementation). Summing over the image, we obtain the energy:

$$E(s, \mathbf{a}) = \sum_i e(s, \mathbf{a}_i) = \sum_i \left[\sum_{j \in \mathcal{W}_i} (s_j - \mathbf{a}_i \cdot \mathbf{I}_j)^2 + \varepsilon \mathbf{a}_i^2 \right] \tag{7.7}$$

However, both s and $\mathbf{a}$ are unknown in the above equation. We follow a similar strategy to Levin et al. [LLW08] and show that $\mathbf{a}$ can be expressed as a function of s, although in our case, the model is linear, not affine as with matting.

Illumination as Only Variable To eliminate $\mathbf{a}$, we observe that, over a window, we have a single unknown $\mathbf{a}$, three input channel values per pixel, and only one s unknown per pixel. By fixing the illumination to $\bar{s}$ in Equation 7.6, we express $\mathbf{a}$ as a linear function of $\bar{s}$. We rewrite Equation 7.6 in matrix form with two vectors S_i and A_i, a matrix M_i, n the number of pixels in $\mathcal{W}_i' = \{j_1 \dots j_n\}$, and I^{r}, I^{g}, I^{b} and a^{r}, a^{g}, a^{b} being the RGB components of $\mathbf{I}$ and $\mathbf{a}$ respectively:

$$e(s,\mathbf{a}_i) = \left[\underbrace{\begin{pmatrix} s_{j_1} \\ \vdots \\ s_{j_n} \\ 0 \\ 0 \\ 0 \end{pmatrix}}_{S_i:\ (n+3)\times 1} - \underbrace{\begin{pmatrix} I^{\mathrm{r}}_{j_1} & I^{\mathrm{g}}_{j_1} & I^{\mathrm{b}}_{j_1} \\ \vdots & \vdots & \vdots \\ I^{\mathrm{r}}_{j_n} & I^{\mathrm{g}}_{j_n} & I^{\mathrm{b}}_{j_n} \\ \sqrt{\varepsilon} & & \\ & \sqrt{\varepsilon} & \\ & & \sqrt{\varepsilon} \end{pmatrix}}_{M_i:\ (n+3)\times 3} \underbrace{\begin{pmatrix} a^{\mathrm{r}}_i \\ a^{\mathrm{g}}_i \\ a^{\mathrm{b}}_i \end{pmatrix}}_{A_i:\ 3\times 1} \right]^2 \tag{7.8}$$

For a fixed illumination, we have $e(\bar{s},\mathbf{a}_i) = (\bar{S}_i - M_i\,A_i)^2$ which is a classical linear least-square functional with A as the unknown. The minimizer is given by: $A_i^{\min} = (M_i^{\mathsf{T}} M_i)^{-1} M_i^{\mathsf{T}} \bar{S}_i$. With this result, we rewrite the local energy as a function of the illumination only:

$$f(S_i) = \left(S_i - M_i (M_i^{\mathsf{T}} M_i)^{-1} M_i^{\mathsf{T}} S_i \right)^2 \tag{7.9}$$

Using the matrix $N_i = Id - M_i (M_i^{\mathsf{T}} M_i)^{-1} M_i^{\mathsf{T}}$ with Id the identity matrix, we obtain the global energy that is a function of the local illumination vectors $\{S_i\}$ only:

$$\sum_i (N_i S_i)^2 = \sum_i S_i^{\mathsf{T}} N_i^{\mathsf{T}} N_i S_i \tag{7.10}$$

This defines a functional in which each illumination value s appears in several local vectors S_i. To obtain a formula where each variable appears only once, we regroup all the s values into a large vector S that has as many elements as pixels. Then, we rearrange the terms of the $N^{\mathsf{T}}N$ matrices into a large matrix L. Each time the s_i and s_j variables interact in Equation 7.10, this contributes to the $(i,j)^{\text{th}}$ element of L with the element of the corresponding $N^{\mathsf{T}}N$ matrix:

$$L(i,j) \;=\; \sum_{k \mid (i,j)\in\mathcal{W}_k'} N_k^{\mathsf{T}} N_k(i_k, j_k) \tag{7.11}$$

where i_k and j_k are the local indices of i and j in $\mathcal{W}_k'$. With L, we obtain the least-square energy that represents our image-formation model based on local reflectance planes:

$$F(S) = S^{\mathsf{T}} L\, S \tag{7.12}$$

Discussion By assuming a constant $\mathbf{a}$ vector in each window, we seek an illumination function s that can be locally expressed as a linear combination of the RGB channels. Even though each window has its own $\mathbf{a}$ vector, the choice of these vectors is constrained because the windows overlap. For instance, if we consider a pixel i_0 that belongs to two windows with vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ respectively, the illumination s_{i_0} at i_0 should minimize both $(s_{i_0} - \mathbf{a}_1 \cdot \mathbf{I}_{i_0})^2$ and $(s_{i_0} - \mathbf{a}_2 \cdot \mathbf{I}_{i_0})^2$, which couples the $\mathbf{a}_1$ and $\mathbf{a}_2$ vectors and ensures information propagation across windows (Figure 7.4).

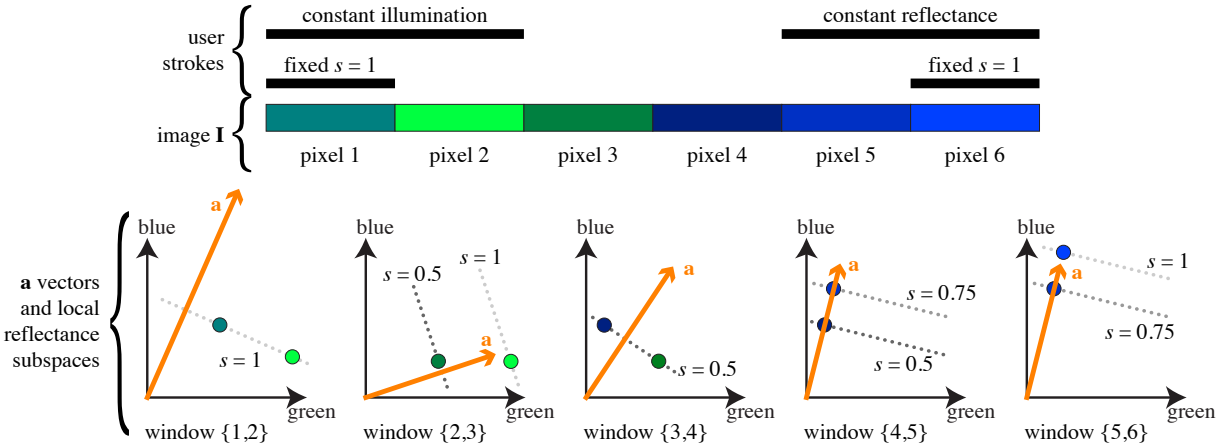


Figure 7.4: *We illustrate how our optimization scheme behaves on a simple example of a 1D image with green-blue colors. The orange* $\mathbf{a}$ *vectors define the local reflectance subspaces (lines in this example, planes on normal 2D images). The iso-illumination lines are shown with gray dotted lines. The* $\mathbf{a}$ *vectors are not explicitly computed by our optimization scheme. They are displayed for visualization purposes. Each window contains 2 pixels. Each pixel belongs to two windows, which constrains the reflectance subspaces. In addition, the user has specified the strokes shown above the pixels. The two end pixels are fixed to* $s = 1$*, which constrained the isoline* $s = 1$ *to go through them. The second pixel is constrained to have the same illumination as the first one. This fully defines the isoline* $s = 1$ *in the first window. In the second window, the isoline* $s = 1$ *is constrained by the second pixel. Since there is no constraint on the third pixel, the reflectance subspace is defined by the optimization process in order to minimize the least-squares cost. The third window with the pixels 3 and 4 is also fully determined by the optimization. The fourth window is partially constrained by the fifth pixel that has a constrained reflectance which defines its illumination relatively to the sixth pixel. The fifth and last window is fully constrained by the fixed illumination and constant reflectance strokes.*

3.2.1 User Strokes

We propose 3 types of brushes so that the user can specify local cues about the reflectance and illumination (Figure 7.5(a)). In this section we detail how these constraints are integrated in our energy formulation, while a typical interactive session is described in section 5.

(a) Input scribbles (b) **a** vectors (c) Reflectance (d) Illumination

Figure 7.5: *(a) The user can specify pixels sharing a constant reflectance (blue), a constant illumination (red), or a fixed illumination (white). Each type of scribble is represented with a single color for illustration purpose, while in our implementation strokes of similar color share the same constraint. (b) Visualization of the* **a** *vectors that vary across the image to fit the user constraints. (c,d) Resulting reflectance and illumination.*

The first tool is a *constant reflectance brush*. If two pixels share the same reflectance $\mathbf{R}_0$, then $\mathbf{I}_1 = s_1\mathbf{R}_0$ and $\mathbf{I}_2 = s_2\mathbf{R}_0$ that leads to $s_1\mathbf{I}_2 = s_2\mathbf{I}_1$. We define a least-square energy function at a pixel i covered by a reflectance stroke $\mathcal{B}_i^{\mathrm{R}}$:

$$z(\mathcal{B}_i^{\mathrm{R}}) \sum_{j \in \mathcal{B}_i^{\mathrm{R}}} \left(s_i\mathbf{I}_j - s_j\mathbf{I}_i\right)^2 \tag{7.13}$$

where $z(\cdot)$ is a normalization factor that ensures that strokes have an influence independent of their size, $z(\mathcal{B}) = 1/|\mathcal{B}|$ with $|\mathcal{B}|$ the number of pixels in $\mathcal{B}$. For convenience, if there is no stroke on i, we define $\mathcal{B} = \varnothing$ and $z(\mathcal{B}) = 0$. We sum over the image to obtain:

$$U^{\mathrm{R}}(s) = \sum_i z(\mathcal{B}_i^{\mathrm{R}}) \sum_{j \in \mathcal{B}_i^{\mathrm{R}}} \left(s_i\mathbf{I}_j - s_j\mathbf{I}_i\right)^2 \tag{7.14}$$

We also provide a *constant illumination brush* to the user to indicate regions with constant s values. Between a pair of pixels, this means $s_1 = s_2$, which translates into the energy $(s_1 - s_2)^2$. At the image level, this gives:

$$U^{\mathrm{S}}(s) = \sum_i z(\mathcal{B}_i^{\mathrm{S}}) \sum_{j \in \mathcal{B}_i^{\mathrm{S}}} \left(s_i - s_j\right)^2 \tag{7.15}$$

where $\mathcal{B}_i^{\mathrm{S}}$ is a constant-illumination stroke covering i and z is the same normalization factor as in the previous case.

Finally, we define a *fixed illumination brush* so that the user can specify absolute illumination values that are used as hard constraints in our optimization. In practice, we use this brush only to indicate fully lit pixels. These areas can be easily recognized by users. We do not use this brush for intermediate illumination values which would be harder to estimate for users. Formally, the brush defines a set $\mathcal{C}$ of pixels which illumination values are fixed, that is for all $i \in \mathcal{C}$, $s_i = \bar{t}_i$ with $\bar{t}_i$ the user-specified value at pixel i. For instance, $\bar{t} = 1$ for fully lit regions.

3.2.2 Constrained Least-square System

We combine the functional modeling the reflectance subspaces with the user input to obtain the following optimization:

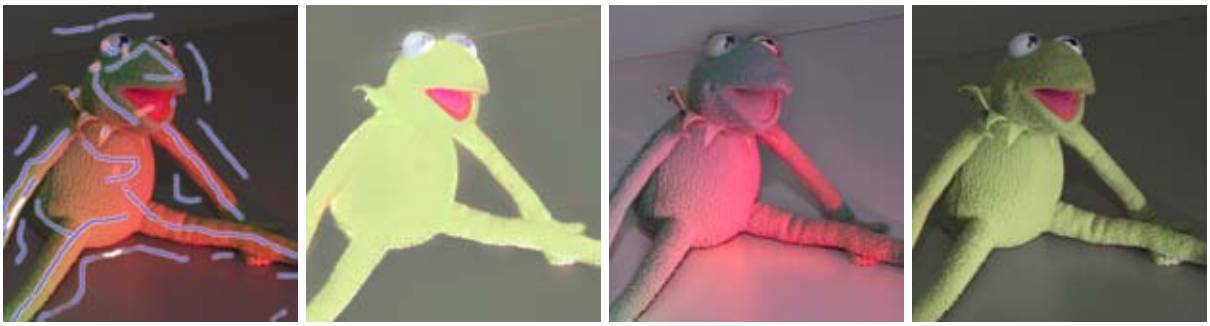
$$\begin{aligned} \arg\min_s \; & F(s) + w\left[U^{\mathrm{R}}(s) + U^{\mathrm{S}}(s)\right] \\ & \text{such that } \forall i \in \mathcal{C}, s_i = \bar{t}_i \end{aligned} \tag{7.16}$$

where w controls the importance of the strokes.

In practice, we use $w = 1$ to give equal importance to our image model and to the user cues, which yields consistently satisfying results. Equation 7.16 defines a constrained least-square optimization since each term is a quadratic function of s. A minimizer can be obtained using classical linear solvers. For small neighborhoods $\mathcal{W}_i'$, the system is sparse. As an example, for 3×3 windows the L matrix (Eq. 7.12) has only 25 non-zero entries per pixel and the overhead of the brushes is negligible. But for large neighborhoods, L is less sparse and the computation becomes expensive. Section 4 describes a downsampling-upsampling scheme adapted to our problem.

3.3 Colored Illumination

Although our method is derived under the assumption that illumination is monochromatic, that is, s is a scalar, we found that it is able to cope well with colored illumination coming for instance from interreflections. In this case, the illumination is a RGB vector $\mathbf{s} = (s^{\mathrm{r}}, s^{\mathrm{g}}, s^{\mathrm{b}})^{\mathsf{T}}$ and our image formation model becomes: $\mathbf{I} = \mathbf{s} * \mathbf{R}$ with $*$ the per-channel multiplication. We use the previously described optimization method to compute each illumination component separately. The only difference is how we interpret the user strokes. For the channel $c \in \{\mathrm{r}, \mathrm{g}, \mathrm{b}\}$, the constant reflectance energy becomes: $\sum_i z(\mathcal{B}_i^{\mathrm{R}}) \sum_{j \in \mathcal{B}_i^{\mathrm{R}}} \left(\mathbf{s}_i^c I_j^c - s_j^c I_i^c\right)^2$ and the constant illumination energy: $\sum_i z(\mathcal{B}_i^{\mathrm{S}}) \sum_{j \in \mathcal{B}_i^{\mathrm{S}}} \left(s_i^c - s_j^c\right)^2$. Using these new definitions, we minimize Equation 7.16 for each RGB channel.



(a) User scribbles (b) Reflectance (c) Illumination (d) White balanced

Figure 7.6: *Our illumination estimation is robust to colored illumination (c). Mapping the estimated colored illumination to gray level and multiplying back with reflectance results in a white balanced image (d).*

4 Distribution-Preserving Downsampling

The energy formulation described in the previous section relies on color distributions over local windows. However, to capture the color distribution of a textured reflector, the size of the window must be at least as big as the pattern. For example, in Figure 7.7 the blue lines are constrained as reflectance variations by a user scribble, but are interpreted as illumination variations away from the scribble because a 3×3 window does not cover two lines at a time (first row). 7×7 windows are required in order to capture the blue line texture away from the scribble (second row).

Unfortunately, large windows imply a less sparse least-square system and higher memory and computation costs. This suggests the use of multiresolution approaches such as multigrid, but we have found that standard methods perform poorly in our case. This is because the downsampling they rely on does not respect color distributions. For example, nearest neighbor or even bicubic interpolation can quickly discard colors that do not appear in enough pixels (Figure 7.9, first and second row). We introduce a new downsampling scheme that does not focus on SSD error but seeks to faithfully preserve the set of colors in a region.

Our new distribution-preserving downsampling is based on two key ideas. First, rather than define the computation of low-resolution pixels one at a time, we consider blocks of low-resolution pixels, because we need multiple pixels to store a distribution of colors. That is, we downsample high-resolution blocks of $w \times w$ pixels into blocks of $v \times v$ pixels (typically from 4×4 to 2×2). Second, we select the best representatives of the color distribution of a block using clustering.

We integrate this downsampling scheme in a multigrid solver and further show that coarse-level results can be upsampled accurately by upsampling the $\mathbf{a}$ vectors rather than the illumination itself. This requires explicitly extracting the per-window $\mathbf{a}$ values, but this is easy given s.

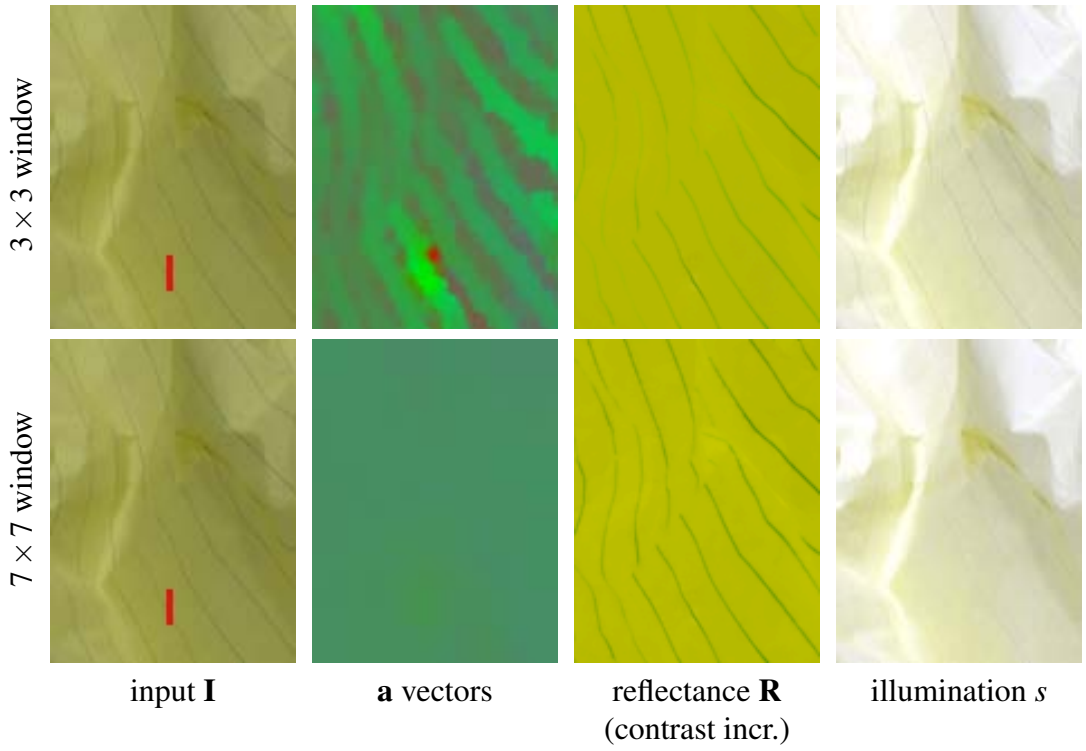


Figure 7.7: *A* 3×3 *window does not capture the color variations away from the scribble (red pixels). A* 7×7 *window ensures a better propagation of the* **a** *vectors modeling the reflectance planes.*

Downsampling Our scheme downsamples large $w \times w$ windows into small $v \times v$ windows such that $w = pv$ with w, v, and p integer numbers. In this configuration, each pixel in the low-resolution image corresponds to $p \times p$ high-resolution pixels. First, we divide the input image into $w \times w$ blocks and extract v^2 representative colors in each block with the K-means algorithm. Then, we rearrange these v^2 pixels into a $v \times v$ block. To preserve the image structure, we test all the possible layouts and select the one that minimizes L^2 distance with the original image. The comparison between this downsampling scheme and standard nearest neighbor and linear downsampling is illustrated in Figure 7.8. We use the low-resolution image $\mathbf{I}_{\downarrow}$ to compute illumination and reflectance as described in Section 3.

Upsampling To obtain full-resolution illumination and reflectance, one could apply a standard upsampling scheme but generic techniques ignore our image formation model. We leverage the planar reflectance structure to derive a high-quality and computationally efficient upsampling method. First, we use Equation 7.8 to compute the $\mathbf{a}_{\downarrow}$ vectors at low resolution. Then, we upsample these vectors using an existing technique to obtain $\mathbf{a}_{\uparrow}$. In practice, we use bilinear upsampling. Finally, we compute the high-resolution illumination $s_{\uparrow}$ using the reflectance planes defined by $\mathbf{a}_{\uparrow}$, that is: $s_{\uparrow} = \mathbf{a}_{\uparrow} \cdot \mathbf{I}$. Levin et al. [LLW08] describe a similar approach to speed up their matting algorithm. The high-resolution reflectance $\mathbf{R}_{\uparrow}$ is then computed as $\mathbf{R}_{\uparrow} = \mathbf{I}/s_{\uparrow}$.

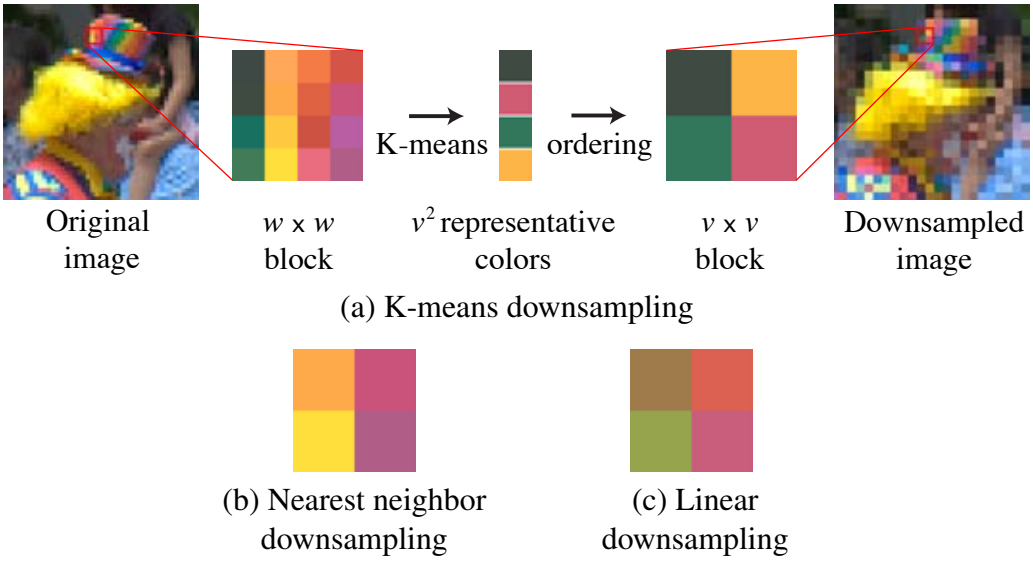


Figure 7.8: *For a downsampling factor p, we use K-means clustering to extract v^2 representative colors of each $w \times w$ image block. These representative colors are then ordered to form the downsampled image (a). In comparison, nearest neighbor (b) looses the green and gray color information, while linear downsampling (c) produces a loss of contrast.*

Figure 7.9 (third row) illustrates the benefits of this strategy for textured surfaces: 3×3 windows are used on a downsampled input to compute a decomposition that closely match the solution obtained with 7×7 windows at fine resolution. In comparison, nearest neighbor (first row) and linear downsampling (second row) incorrectly include the blue lines in the illumination.

Multigrid Solver To further improve the performances of the linear solver, we apply the above multi-resolution approach recursively, in the spirit of multigrid algorithms [BHM00]. The idea behind multigrid methods is to efficiently solve for the low frequency components in a low resolution version of the domain, and refine the high frequency components at higher resolutions. In practice, we pre-compute a pyramid of the image at multiple scales using our distribution-preserving scheme that reduces 4×4 windows into 2×2 blocks. We compute the corresponding L matrix at each level (Eq. 7.12). We also cache the results of the K-means downsampling. When the user adds scribbles, we use this information to propagate the constraints in the pyramid at virtually no cost. At run time, we follow a coarse-to-fine strategy. The solution at a given level is upsampled to the next level and refined with a few Gauss-Seidel iterations. We iterate this scheme until we reach the finest resolution.

Combining our distribution-preserving downsampling and smart upsampling, even the coarsest level yields a good approximation of the final solution. We use this property to provide early visual feedback to the user by directly upsampling the solution of the first pyramid level using the method described in the previous paragraph. The rest of the pyramid is processed in the background. With our unoptimized C++ implementation on dual-core 3 GHz PC, a 800×600 image

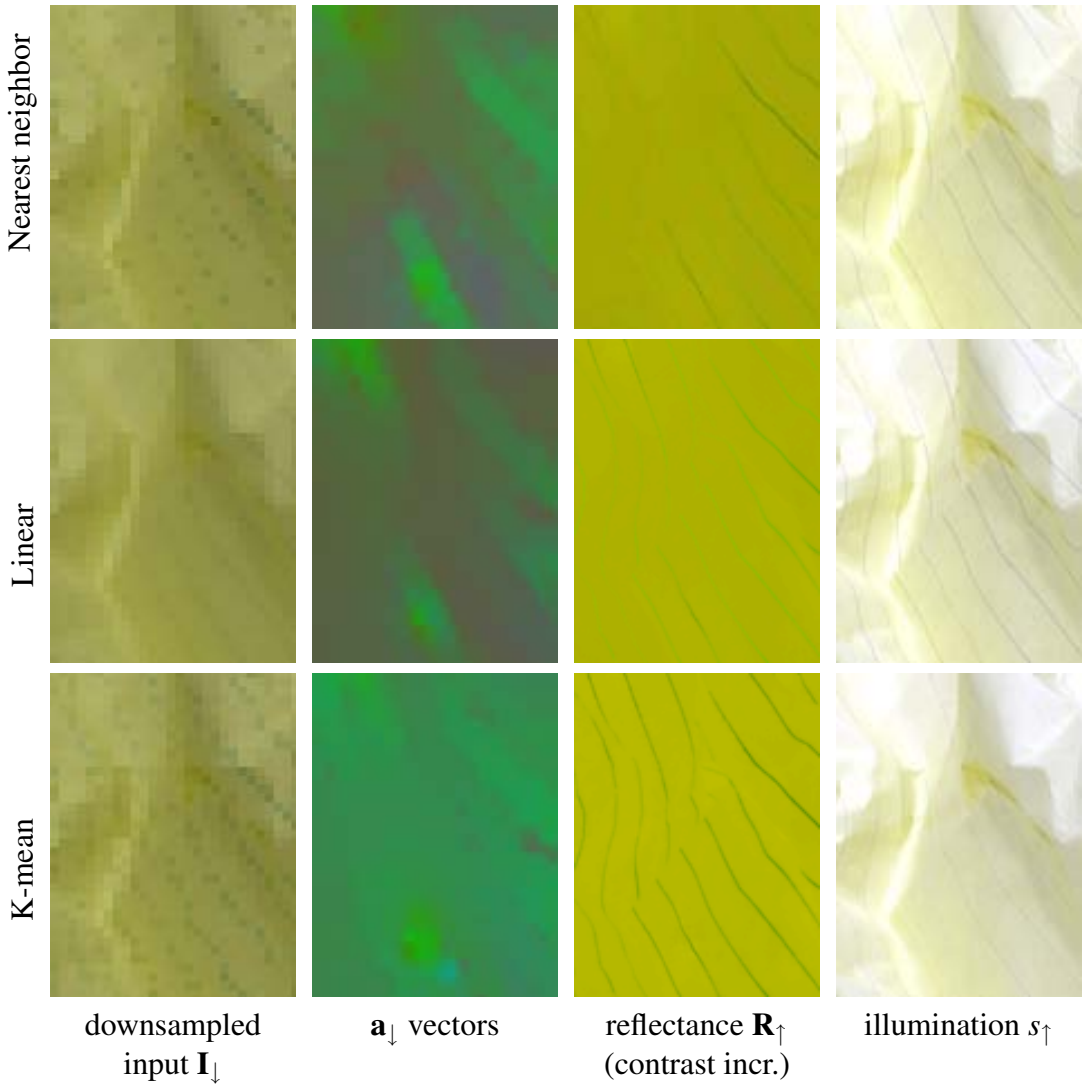


Figure 7.9: *Multi-resolution strategy. We approximate large windows at fine resolution as small windows in a coarse version of the image. Nearest neighbor downsampling and linear downsampling fail to preserve the color variations at coarse resolution. Our K-means scheme preserves the blue and yellow color variations, leading to a better approximation of the reflectance plane.*

requires 20 seconds to precompute the L matrices (this happens only once), visual feedback is available within a second and fully converged result after about 15 seconds. Table 7.2 details the computation time for the images in this paper.

Since we use a standard least-square approach, one could use an optimized solver such as the GPU-based method of Buatois et al. [BCL07] or McCann and Pollard [MP08] to further speed up the computation.

5 Results and Applications

Inputs The energy derivation described in section 3 assumes linear image values. When dealing with JPEG inputs (Figure 7.16), we approximate linear images by inverting the gamma, although this inversion can be sensitive to quantization and saturation. For these reasons, most of the results in this paper have been generated from RAW images, that provide linear values. Moreover, the 16 bit depth of RAW images allows a greater accuracy in dark areas of the image when computing reflectance as $\mathbf{R} = \mathbf{I}/s$. This is true for any intrinsic image method.

User Interactions In a typical session, a user starts by specifying a few fixed-illumination strokes in order to fix the global scale ambiguity between bright objects in shadow and dark objects in light. The resulting initial solution can then be iteratively refined by adding constant-reflectance and constant-illumination strokes. In theory, only one fixed-illumination stroke is enough to numerically solve the gobal scale ambiguity. However, if the scene contains disconnected components, for example a subject in front of a distant background, each component needs a fixed stroke since the illumination information cannot be propagated. For example in Figure 7.5, one fixed-illumination scribble is required for the pink flower, and one for the green leaves because these two regions form different connected components that do not share a similar reflectance or a similar illumination.

The fixed-illumination brush is often applied over the brightest points of objects, which are easy to identify for a human oberver. An inaccurate fixed illumination value only introduces a global scaling factor over connected color regions, which still produces a plausible result. The constant-reflectance and constant-illumination strokes are most effective when applied to regions with complementary variations, e.g. using the constant-illumination brush across reflectance edges often significantly improves the result. Similarly, the constant-reflectance brush is effective when applied over inter-reflections and other colored lighting variations. Our video illustrates these intermediate steps [1].

Figure 7.10 illustrates how the per-pixel error evolves with the number of strokes, computed from ground truth data with two sets of strokes scribbled by different users. Because no method exists to obtain ground truth decompositions from color photographs, we created a synthetic scene inspired by Figure 7.1 and computed a global illumination solution (Figure 7.13). Typically, the fixed illumination strokes drastically reduce the error by fixing global ambiguities, while the constant reflectance and illumination strokes correct small, but visually important, local errors. The two different sets of strokes quickly converge to decompositions that differ only by a global scaling factor.

Our approach is robust to small variations in scribble placement and value, which makes it easy to use. To assess this, we have computed several results where we have randomly moved the scribbles up to 15 pixels and randomly changed the fixed illumination values up to 5%. All the results look equally plausible with objects appearing slightly brighter or darker, and remain

[1] Video available on the project webpage: http://artis.imag.fr/Publications/2009/BPD09/

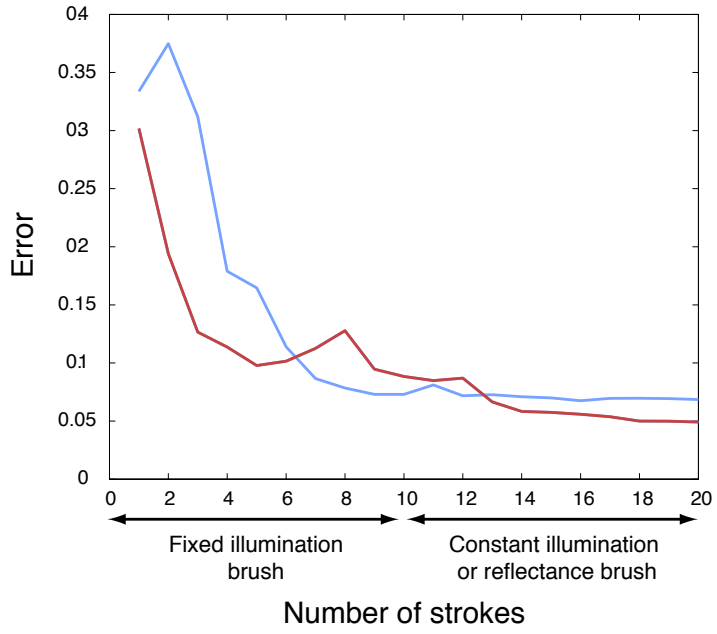


Figure 7.10: *Per-pixel error computed from a ground truth synthetic image, for two different sets of scribbles. Fixed illumination scribbles quickly produce a good estimate (*10 *first strokes), while constant illumination or reflectance scribbles are used to refine the result. The remaining error is due to the global scaling factor that has been over or under estimated.*

usable in graphics applications (see results in supplemental materials). Table 7.1 reports the average per-pixel error for various scribble alterations on the synthetic image of Figure 7.13. While the amount of user scribbles required is comparable to other approaches [LLW04, LLW08], an important difference is that, in our approach, most of the scribbles do not require the users to specify numerical values. Instead, user only indicates similarity between reflectance or illumination, which is more intuitive to draw. Table 7.2 details the number of strokes for the examples in this paper.

Scribbles	Per-pixel error (%)
Fixed-illumination values set to ground truth values	2.28
Fixed-illumination values set to 1	5.98
Fixed-illumination values set to ground truth values, position randomly altered up to 15 pixels	7.12
Fixed-illumination values set to ground truth values randomly altered up to 5%	5.19

Table 7.1: *Average per-pixel error computed from a ground truth synthetic image with randomly altered scribbles.*

	Res.	User strokes	Matrix (s)	Solving (s)
Baby (fig. 7.1)	533×800	58	20.64	11.93
Flower (fig. 7.5)	750×500	15	18.21	7.87
Clown (fig. 7.11)	486×800	33	18.83	9.53
St Basile (fig. 7.16)	800×600	81	23.29	14.56
Paper (fig. 7.16)	750×500	36	18.17	9.2

Table 7.2: *Resolution, number of scribbles and computation time for matrix pre-computation and solving for the results of this paper. Note that we use the coarse level of the pyramid to display an approximate solution after 1 second (cf. text for detail).*

Intrinsic Decompositions Figure 7.11 illustrates the intrinsic image decomposition that our method produces on a colorful photograph. In comparison, a luminance computation will produce different values for light and dark colors.

(a) Input (b) Naive luminance (c) Illumination (d) Reflectance

Figure 7.11: *While a naive luminance computation produces lower values for dark colors (b), our approach provides a convincing estimation of illumination (c) and reflectance(d).*

We compare our approach with Tappen et al. [TFA05] algorithm in Figure 7.12 and Figure 7.16. This method combines a Retinex classifier based on chromaticity variations and a bayesian classifier trained on gray-level images. The result of these classifiers is a binary map that labels the image gradients as reflectance or illumination. The final reflectance and illumi-

nation images are reconstructed using a Poisson integration on each color channel. The main limitation of this approach is that a binary labelling cannot handle areas where both reflectance and illumination variations occur, such as in highly textured areas, along occlusion boundaries or under mixed lighting conditions. On the other hand, this approach is fully automatic.

(a) User scribbles

(b) Our reflectance and illumination

(c) Tappen's reflectance and illumination from a single image

Figure 7.12: *Comparison with the automatic approach of Tappen et al. [TFA05].*

In Figure 7.13 we show a similar comparison on ground truth data from a synthetic image. The information specified by the user together with our propagation algorithm allow us to extract fine reflectance and illumination, but our planar reflectance assumption prevents us from considering the black pixels of the eyes as reflectance. Because Tappen's method is automatic, there is a remaining scale ambiguity over the reflectance and illumination after Poisson reconstruction.

We fix the brightest point of the illumination to a value of 1 in Figure 7.13(c) but the estimated illumination is still darker than the ground truth. The reflectance in Figure 7.13(c) is not uniform because of the occlusion edges that are classified as reflectance but also contain illumination variations.

(a) Ground truth

(b) Our method from a single image and user scribbles

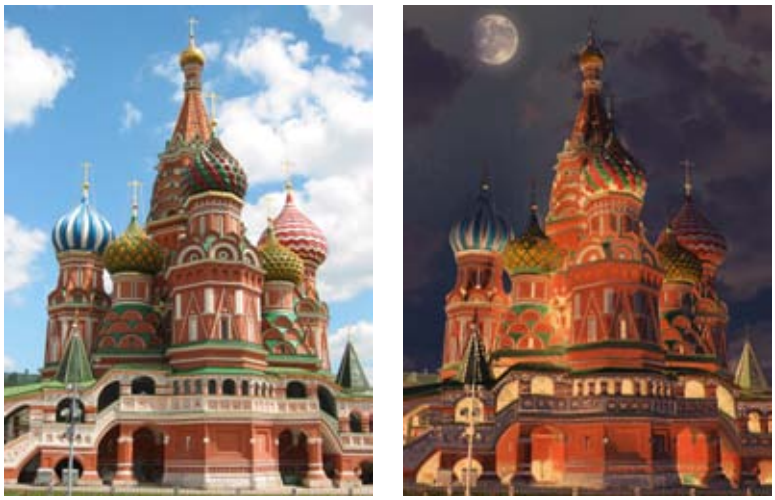
(c) Tappen's method from a single image

Figure 7.13: *Comparison with ground truth data from a synthetic image. Compared to the automatic method of Tappen et al. [TFA05] (c), our user assisted approach produces finner results, but interprets the black pixels of the eyes as shadow (b).*

Finally, we compare in Figure 7.18 our method with the automatic approach of Shen et al. [STL08]. While their texture constraints greatly improve the standard Retinex algorithm, posterization artifacts are visible in the resulting reflectance due to the clustering that imposes that pixels of the same texture receive the same reflectance value. Increasing the number of clusters reduces the posterization but also reduces the benefit of the texture constraints. As any automatic approach, the result cannot be corrected in case of failure, such as in the St Basile image where different regions of the sky receive very different reflectance. Moreover, their method cannot handle colored illumination such as inter-reflections.

Reflectance Editing One of the simplest manipulations offered by intrinsic images is editing one of the image components (reflectance or illumination) independently of the other. Figure 7.1(e) and 7.15 give examples of reflectance editing inspired by the re-texturing approach of Fang and Hart [FH04]. We use a similar normal-from-shading algorithm to estimate the normals of the objects. Textures are then mapped in the reflectance image and displaced according to the normal map. We finally multiply the edited reflectance image by the illumination image to obtain a convincing integration of the new textures in the scene. While Fang and Hart obtained similar results using the luminance channel of the image, our illumination represents a more accurate input for their algorithm and other image based material editing methods [KRFB05] when dealing with textured objects (Figure 7.15).

Relighting Once reflectance and illumination have been separated, the scene can be relighted with a new illumination map. Figure 7.14 illustrates this concept with a simple yet effective manipulation. Starting from a daylight picture, we invert the illumination image to create a night-time picture. Although this simple operation is not physically accurate, we found that it produces convincing results on architectural scenes because the areas that do not face the sun in the daylight image are the ones that are usually lit by night (interiors and surfaces oriented to the ground). We polish the result by adding a fake moon and mapping the gray level illumination to an orange to purple color ramp.



(a) Original image (b) Night-time relighting

Figure 7.14: *From a daylight picture (a), we invert the illumination and add a fake moon to create a night-time picture (b). Note however that the process reveals saturation and blocky artifacts from the JPEG compression. Original image by Captain Chaos, flickr.com.*

Discussion Although we found that our method works well in most cases, we acknowledge that similarly to many inverse problems, the results can be sensitive to the input quality. JPEG

(a) Input (b) Our method

(c) Naive luminance

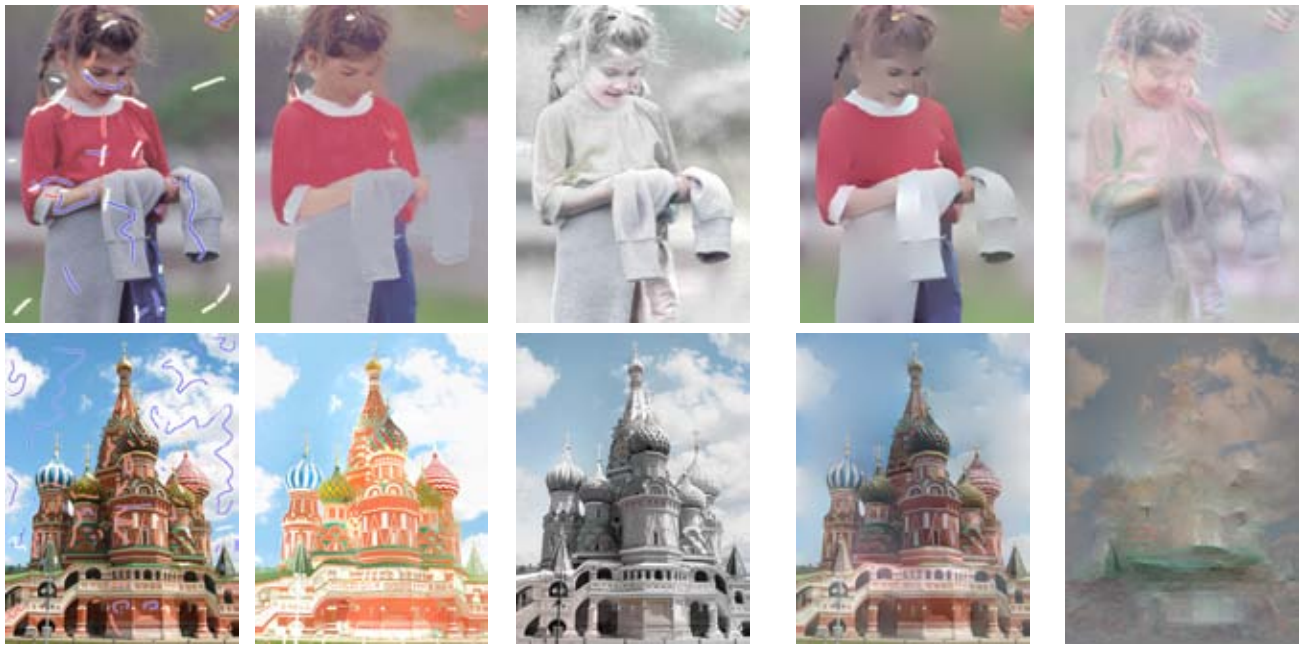
(d) Tappen's method

Figure 7.15: *Our approach (b) produces illumination maps that are more accurate than luminance (c) for re-texturing textured objects. On this highly textured image, the automatic classifier of Tappen et al. [TFA05] cannot decompose reflectance and illumination properly (d), which impacts the result of the manipulation.*

artifacts and noise can become visible if one applies an extreme transformation to the illumination and reflectance components, especially since JPEG compresses color information aggressively. For instance, inverting the illumination as in Figure 7.14 can reveal speckles on strongly compressed images. Over-exposure and color saturation are other sources of difficulty since information is lost. For instance, the clown in Figure 7.11 contains out-of-gamut colors that have been clipped by the camera. As a consequence, some residual texture can be discerned in the illumination component. However, existing methods share all of these difficulties that are inherent to the intrinsic image decomposition. In Figure 7.17, we compare our result with Weiss' technique [Wei01]. Even with 40 images taken with a tripod and controlled lighting, illumination variations remain visible in the reflectance component, thereby having the opposite bias of our method.

6 Conclusions

We have described in this chapter a method to extract intrinsic images based on user-provided scribbles. It relies on a low-rank assumption about the local reflectance distribution in photographs, which allows us to compute the illumination as the minimizer of a linear least-square functional. We have also presented a new downsampling scheme based on local clustering that focuses on preserving color distribution. Our method can handle complex natural images and can be used for applications such as relighting and re-texturing.



(a) User scribbles (b) Our reflectance and illumination from a single image and user scribbles (c) Tappen's reflectance and illumination from a single image

Figure 7.16: *Comparison with the automatic approach of Tappen et al. [TFA05]. St Basile Cathedral image by Captain Chaos, flickr.com.*

(a) User scribbles (b) Our reflectance and illumination from a single image and user scribbles (c) Weiss' reflectance and illumination from 40 images

Figure 7.17: *Comparison with the multiple image approach of Weiss [Wei01].*

Figure 7.18: *Comparison with the automatic approach of Shen et al. [STL08], from a single image. St Basile Cathedral image by Captain Chaos, flickr.com.*

Part V

Conclusions

The ability to create and modify images in a variety of appearances (realistic, stylized, simplified) greatly benefit the visual communication. In this dissertation we have explored various ways of navigating from one form of image to another (Figure 7.19). We have proposed methods to transform a complex image into a simplified representation (Chapter 1 and 2), and to enrich a simple line drawing with complex color gradients (Chapter 3). We have also presented two methods to produce stylized animations from videos and 3D scenes (Chapter 5 and 6), and a user assisted approach to facilitate the manipulation of lighting and material in photographs while preserving their realistic aspect (Chapter 7).

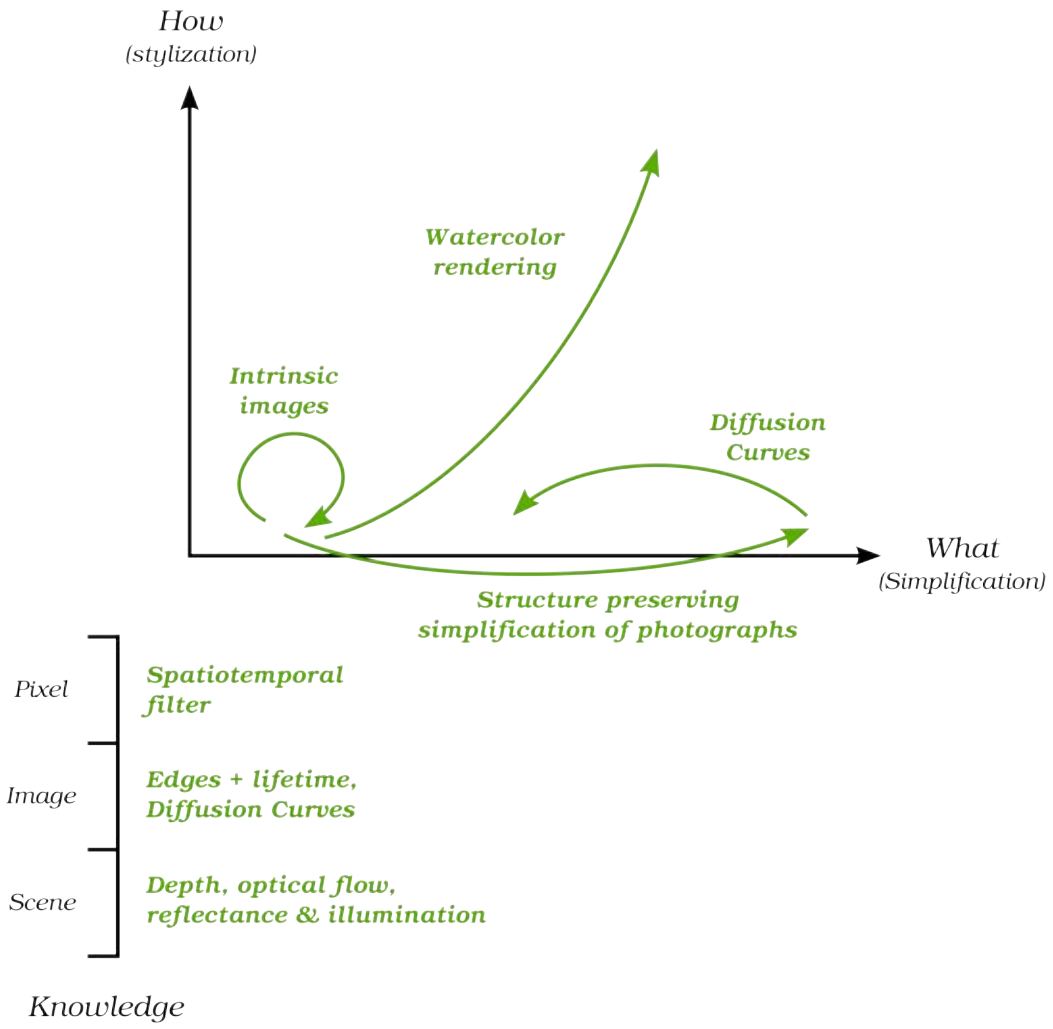


Figure 7.19: *Visualization of the image manipulations proposed in this manuscript to navigate in the space of visual representations.*

All these image manipulations rely on different levels of knowledge concerning the depicted scene. Some of the proposed methods rely on existing algorithms to obtain this knowledge (for example the optical flow estimation), but we also introduced novel methods for extracting information about the scene. A multi-scale edge structure of an image is introduced in Chapter 2

and the image decomposition into illumination and reflectance is described in Chapter 7. To produce an image from manipulated visual content, we have proposed several dedicated rendering algorithms. Our fast Poisson solver renders color images from diffusion curves in real time (Chapter 3); and our bidirectional texture advection (Chapter 5) and infinite zoom mechanism (Chapter 6) offer convincing temporal coherence for two different kind of input sequences (videos and 3D animations).

Our work on image editing, simplification and stylization opens a number of potential research directions. We review a few of these directions.

1 Simplification and Abstraction

The two simplification methods presented in this dissertation process visual data globally, without consideration of the local content of the image. In the future, we would like to explore how more knowledge about the depicted scene can drive content aware simplification. In traditional illustration, different means of simplification are used for different kinds of visual information. For example, cast shadows are often more simplified than objects, because humans are very tolerant to shadow inconsistency [DCFR07]. Repetitive textures can be suggested through indications that depict the texture pattern only locally, and rely on the human visual system to interpolate this textural clues over the simplified areas (Figure 7.20). This is a very efficient way to transmit complex visual information without introducing too mush visual clutter.

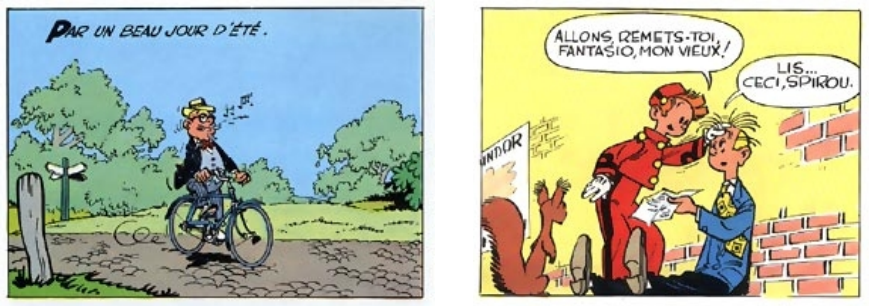


Figure 7.20: *Texture indications can efficiently depict repetitive textures (bricks, cobblestone) without introducing visual clutter.* ©*Dupuis.*

However, to the best of our knowledge, no existing method is able to produce accurate and controllable shadow simplification or texture indication from a single photograph. The main challenge of this family of applications is to decompose the image into different "layers" of visual content that could be simplified independently. In our terminology, this corresponds to an exploration of the knowledge of the depicted content at its scene level. The method presented in Chapter 7 could be used to simplify reflectance and illumination separately, although related methods on cast shadow removal [SL08] may be more suited to shadow simplification. Concerning texture indications, computer vision algorithms could segment textured areas and extract the

representative patterns. The challenge will then be to identify and instantiate the simplification rules used by illustrators. For instance, texture indications tend to be drawn along edges of texture regions. Existing work relies on the user to specify these locations [WS94], but an automatic approach may be derived based on how a particular layout of indications minimizes visual clutter while effectively depicting a textured area.

Another direction of research we would like to pursue is the abstraction of an image by means of *factorization* (or *generalization*). Factorization produces abstract representation by retaining the common features shared by multiple instances of an object and neglecting details specific to only one instance [Kra07]. Factorization is the basis of many abstract representations because of its capacity to convey general concepts. Icons and road signs are extreme examples of factorization where complex objects such as a human bodies are represented only by their common features (two arms, two legs and a head) (Figure 7.21). Various computer vision algorithms rely on the factorization of large image data bases to drive scene or object recognition [TFF08]. In the same spirit, one could leverage the variety of photographs available in image collections such as Flickr©to generate generic representations of landscapes, people, or famous places.

(a) The Beatles©, Abbey Road (b) Pedestrian warning

Figure 7.21: *While a photograph captures a specific instance of a subject (a), an abstracted depiction formulates general concept by retaining only the common features of multiple instances (b).*

Factorization is also commonly used in scientific illustrations such as botanical drawings where the artist studies several specimens of the same plant at several phases of their growth to create a single drawing that depicts the common features of a species (Figure 7.22). Such illustrations are often done by hand, and we believe that providing an automatic algorithm to create similar illustrations from multiple images or 3D objects would facilitate the process. The main difficulty in this case would be to identify the characteristic features of plants across specimens and arrange them in a seamless composition that remains understandable.

Figure 7.22: *Botanical illustrations factorize the common features of several specimens of a species. Image from www.soc-botanical-artists.org.*

2 Evaluation of Stylized Animation

We have presented in this manuscript two methods to apply a stylization texture over an animation. Both methods offer a convincing trade-off between the 2D appearance of the texture and the 3D motion of the depicted scene. However, in order to achieve high temporal coherence, several versions of the texture need to be blended together, which modifies the visual characteristics of the texture. We have observed that the perception of this degradation varies from a texture pattern to another. In our experiments, we have also noticed that reducing the frame rate can improve the perception of temporal coherence, similarly to traditional animations that typically contain 12 frames per seconds.

These observations raise the difficult question of measuring the quality of a stylization algorithm, both in terms of appearance and temporal coherence. How closely does a given stylization match a real painting? How should a painting evolve during an animation? To what extent are sliding or popping artifacts noticeable? How can we measure the importance of inaccurate motion, of pattern degradation? Answers to these questions could delimit the range of styles that are well suited to existing methods, or drive novel algorithms with increased temporal coherence. Existing user studies on non photorealistic rendering algorithms focus on the evaluation of how stylized rendering resemble hand drawn images [MIA+08, CGL+08] or improve depiction [SD04, CSD+09]. But to the best of our knowledge, in depth studies of temporal coherence of stylized rendering have yet to be performed. However, related effects have been studied in the area of video compression and denoising, and the corresponding experiments and quality measures could be valuable sources of inspiration for our community. While some questions may be

answered by perception experiments, some others are likely to remain open. For instance, we believe that there is no perfect answer on how a painting should evolve, and that only the artist can make such an aesthetic choice. In that case the role of non photorealistic rendering is to offer to the artist a sufficient variety of choices and intuitive control over them.

3 Drawing Complex Images

The Diffusion Curve primitive, presented in Chapter 3, facilitates the creation of complex color gradients in vector graphics. These color gradients can depict realistic effects such as smooth shadows or shallow depth of field. However, many other realistic effects remain hard to create from scratch using existing vector graphics tools.

We would like to follow this direction of research and propose new drawing tools for the creation of realistic images with a "sketch and colorize" metaphor. A large amount of the complexity of natural images comes from materials and textures. This has motivated Winnemöller et al. [WOBT09] to use diffusion curves to control texture coordinates and normals in vector images. Given this enriched diffusion curve, a user can "drape" a texture over a line drawing and design normal fields for dynamic lighting effects. We would like to identify other visual features that contribute to the realism of an image, and to propose intuitive tools to draw and control these effects. While diffusion curves appear to be a compact representation to represent and manipulate smooth color variations, other primitives may be preferable to represent different image features such as highlights or caustics. One source of inspiration for this work could be the *photorealism* art movement, where artists are able to produce paintings that are almost as realistic as real photographs (Figure 7.23). Studying their painting and their creation workflow could help to identify what makes these painting realistic, and how each effect could be numerically drawn and manipulated. Research on the perception of realism could also benefit to this work. Khan et al. [KRFB05] have shown that because humans are tolerant to certain physical inaccuracies in an image (reflections, refractions), manipulations such as material editing can be performed without altering the perception of a plausible photograph. Similar inaccurate but plausible effects would greatly enhance the realism of vector graphic images.

4 Lightfield Editing

Photograph editing is a vast research domain where many directions are yet to be explored. One particular aspect we would like to investigate is the editing of multiple images. Research in computational photography has led to the apparition of camera designs that can capture a scene under several neighboring views, i.e. lightfield cameras [NLB+05]. A lightfield captures rich information about the 3D scene without the explicit definition of 3D models. This additional information allows complex modifications in the image after the shot, such as change of viewpoint and change of focus, which represent a major benefit over traditional photographs.

Figure 7.23: *Artists of the photorealism movement create paintings that are hardly distinguishable from photographs.* Red Menu *by Ralph Goings, 1981.*

In order to make lightfields a viable alternative to conventional photographs, lightfields should be as easy to edit as regular pictures. However, editing lightfields locally remains an open challenge as a modification in one view needs to be coherently propagated to the other views. Multiview stereo algorithms could be used to drive edit propagation, but such algorithms are known to be prone to errors. In the context of user defined edits, we believe that user interactions could be leveraged to refine the results of automatic stereo algorithms, in the spirit of the approach proposed in chapter 7.

In addition, we are interested in investigating the new effects that can be produced in an image from lightfield data. The "bokeh", which is the appearance of out-of-focus points in a photograph, could for example be controlled after the capture. The shape of the bokeh is defined by the aperture of the camera lens, and variations of this shape can lead to various aesthetic effects in pictures with shallow depth of field. Figure 7.24 illustrates a custom bokeh that has been created by placing a heart shaped mask in the aperture of a camera. Similar custom bokeh could be produced and controlled from lightfield data with a synthetic aperture [IMG00], and could even be modified locally in the image, which is not feasible with a conventional camera.

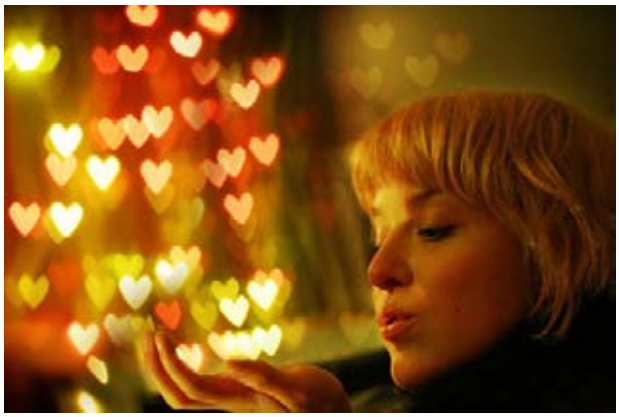
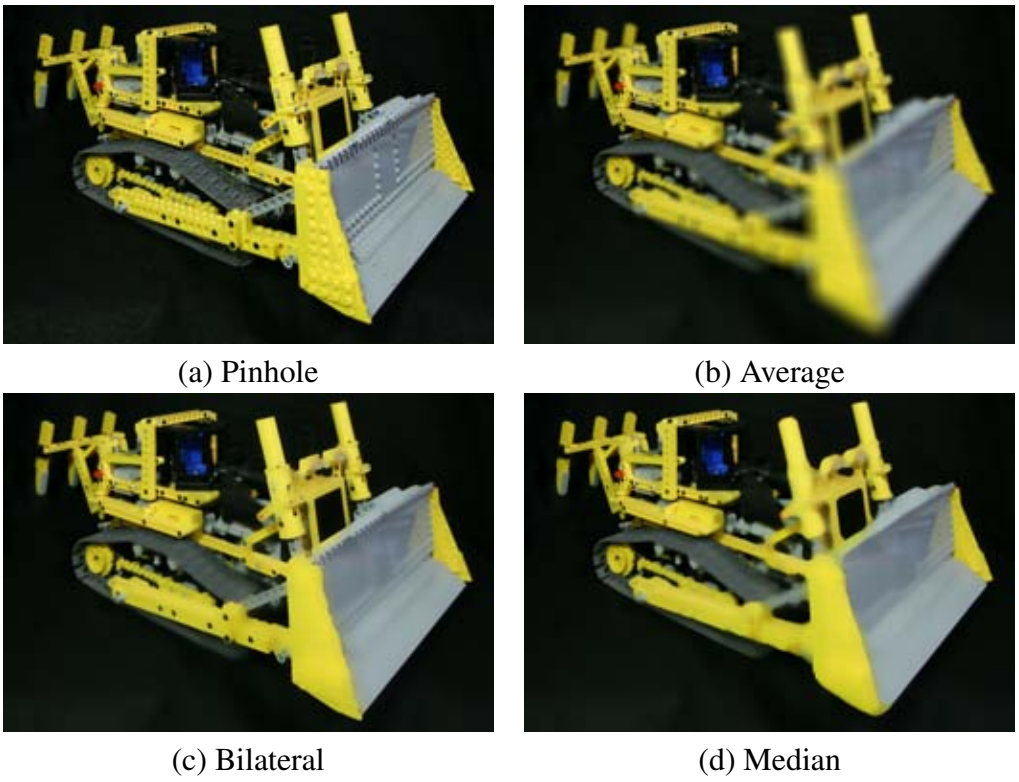


Figure 7.24: *Bokeh, i.e. the shape of out-of-focus points of the scene, can be customized to create various artistic effects. Here the heart shape of the bokeh is obtained by placing a mask in the aperture of the camera. ©isabel bloedwater@flickr.com.*

Finally, effects that could *not* be created from a conventional camera can be obtained from a lightfield. In Figure 7.25 we give our initial results on *non-linear apertures*, that produce stylized depth of field from lightfield data. A non linear aperture is similar in spirit to the conventional aperture of a camera, except that it replaces the blur outside the depth of field by a more complex detail removal filtering, such as a median or bilateral filter. The resulting non-realistic apertures produce stylized images where the amount of abstraction varies automatically with depth. These preliminary results have been presented as a technical talk at Siggraph 2009 [Bou09], and we plan to explore similar lightfield stylization in the future. The intrinsic 3D information of a lightfield could allow, for example, the extraction and rendering of view dependent scene features such as silhouettes or suggestive contours [DFRS03]. In that sense, the additional information captured by computational cameras facilitates the identification of knowledge about the depicted content.

To conclude, this thesis investigates a number of methods to modify the content of visual representations and to improve their expressive power. The variety of images that can be created range from realistic to stylized, from complex to simple. We believe that keeping this variety in mind will lead to novel forms of depiction. As an example, non linear apertures (Figure 7.25) combine Computational Photography and Non Photorealistic Rendering algorithms to create images that are photorealistic in the plane in focus, and stylized in defocus areas.



(a) Pinhole (b) Average

(c) Bilateral (d) Median

Figure 7.25: *Compared to a traditional aperture that blurs out of focus points (b), non-linear apertures produce depth dependent non photorealistic simplifications. For example a bilateral filter smooths out details while preserving contours (c) and a median filter simplifies out of focus shapes (d).*

appendix A

Résumé en français

Dans notre société de communication, les images sont un support privilégié pour transmettre efficacement de l'information et des idées. Les panneaux routiers, les sites internet, la télévision, les affichages publicitaires, les emballages ou encore les bandes dessinées sont autant d'exemples d'images qui peuplent notre environnement pour communiquer des messages visuellement. La variété de ces exemples illustrent le fait que différents types d'images sont nécessaires pour exprimer différents messages. Les photographies sont les représentations visuelles les plus simples pour capturer et partager la **réalité** que l'on voit à un moment donné. Cependant, des changements dans la prise de vue, comme le point de vue, l'éclairage ou l'exposition peuvent avoir un impact important sur la perception de l'image finale (figure A.1(a) and (b)). Les images réalistes peuvent contenir trop d'informations, résultant en un encombrement visuel qui distrait l'observateur de l'information à transmettre. C'est pourquoi des représentations **simplifiées** sont utilisées pour attirer l'attention de l'observateur et permettre l'interprétation rapide et sans équivoque du message (figure A.1(c)). Les dessinateurs utilisent également des procédés de **stylisation** qui se démarquent de la réalité pour transmettre des informations subjectives et stimuler l'imagination de l'observateur (figure A.1(d)). Le but de cette thèse est de **permettre à un utilisateur de créer des images réalistes, simplifiées ou stylisées qui correspondent au message qu'il cherche à transmettre**. Nous décrivons dans les sections suivantes les trois types d'images que nous souhaitons manipuler, pour ensuite détailler nos approches et contributions.

(a) Photographie quelconque
©63vwdriver@flickr.com

(b) Couché de soleil
©Mr.Photo@flickr.com

(c) Illustration technique
©www.the-blueprints.com

(d) Dessin stylisé
©Kevin Kidney, Disney

Figure A.1: *Des images différentes pour des illustrations différentes. Comparé à une photographie quelconque d'une VW Beetle (a), l'éclairage d'un couché de soleil (b) met en valeur l'image de liberté associée à cette voiture bon marché. Un dessin technique (c) retient les caractéristiques intrinsèques de la voiture tout en supprimant le éléments visuels inutiles à la description de la forme. Un dessin stylisé suggère une information subjective, comme l'aspect "mignon" de cette petite voiture.*

1 Images réalistes

Une photographie capture l'information visuelle selon les choix du photographe [Dur00]. Le réglage de la profondeur de champ, de l'exposition, de l'éclairage ont un impact direct sur la façon dont l'image va être interpétée ensuite. Par exemple, une longue exposition peut être utilisée pour suggérer le mouvement au travers du flou de bougé, alors qu'une courte exposition va figer un instant imperceptible à l'oeil nu. La plupart de ces choix doivent être faits au moment de la prise de vue, ce qui contraint le photographe à transmettre l'information telle qu'il la concevait sur l'instant.

La capacité à modifier les configurations de prise de vue après coup représente un outil puissant pour adapter une photographie à des changements d'intention. De la même façon, le photographe a souvent peu de contrôle sur ce qui va apparaître dans le cadre de la photographie. Des objets, matériaux ou éclairages indésirables peuvent nuire à l'intention originelle. Par exemple, un coucher de soleil est préférable à une lumière de plein jour pour donner une atmosphère chaleureuse à un lieu, alors que la présence de brouillard produit une ambiance plus oppressante (figure A.2). Il faut le talent et la patience d'un artiste pour maîtriser de tels effets photographiques, et beaucoup de photographes amateurs sont déçus par leurs photographies qui ne correspondent pas à l'image qu'ils avaient en tête lors de la prise de vue.

La recherche en photographie numérique tente de lever ces contraintes et permettre la modification du contenu d'une image tout en préservant son réalisme. Le chapitre 7 de ce manuscrit

contribue à cette recherche en décrivant une méthode assistée par l'utilisateur pour extraire l'information d'illumination et de réflectance d'une photographie. Cette décomposition permet ensuite l'édition indépendante de l'éclairage ou de la couleur des objets dans une image.

(a) Taj Mahal
en plein jour
©ironmanixs@flickr.com

(b) Taj Mahal
au couché du soleil
©betta design@flickr.com

(c) Taj Mahal
dans le brouillard
©foxypar4@flickr.com

Figure A.2: *Des changements dans la scène photographiée, comme l'éclairage ou la présence de brouillard, modifient l'ambiance de l'image.*

2 Images simplifiées

Une photographie capture l'information visuelle du monde qui nous entoure dans toute sa complexité, mais l'abondance d'informations peut distraire l'observateur du véritable message à transmettre. Dans ce contexte, la simplification permet de mettre en avant l'information à communiquer en négligeant les détails inutiles. Une image simplifiée amplifie le message essentiel et concentre l'attention de l'observateur sur ce qui est représenté [McC94]. Les manuels d'assemblage ou les panneaux routiers utilisent par exemple des représentations symboliques pour que l'information à transmettre soit rapidement identifiable et non ambiguë. Les images simplifiées sont également universelles [McC94]. Alors qu'une photographie représente un sujet particulier, une image simplifiée peut représenter n'importe quel sujet similaire. La simplification est alors un processus de factorisation qui illustre des concepts généraux en extrayant les caractéristiques communes de plusieurs instances d'une classe [Kra07]. Enfin, varier le niveau de détail dans une image permet d'attirer l'attention d'un observateur sur des parties spécifiques de la scène représentée (figure A.3). Cet effet est bien connu des photographes qui utilisent une faible profondeur de champs pour flouter le décor et isoler le sujet d'une photographie.

Nous présentons dans ce document deux méthodes pour gommer les détails d'une photographie et d'une vidéo, sans connaissance a priori du contenu visuel (chapitre 1 et 2). L'utilisateur peut guider la simplification pour mettre en valeur des zones particulières de l'image, ou pour imiter l'utilisation d'un pinceau grossier dans un rendu stylisé.

Figure A.3: *"Le Papillon", aquarelle par Eric Alibert. Le papillon est représenté avec beaucoup plus de détails que le fond afin d'attirer l'attention.*

3 Images stylisées

Les images réalistes représentent une scène de façon objective, ce qui limite leur expressivité. Les images stylisées (peinture, dessin) utilisent des primitives graphiques comme les coups de pinceaux pour transmettre l'information visuelle de façon plus subjective. C'est cet éloignement de la réalité qui confère aux images stylisées leur pouvoir évocatif. En représentant l'information de manière non réaliste, la stylisation sollicite l'imagination de l'observateur qui peut ainsi interpréter l'image en fonction de son propre point de vue. L'illustration stylisée est par exemple fréquemment utilisée en bande dessinée pour immerger le lecteur dans un univers imaginaire. Les images stylisées sont aussi populaires dans l'illustration architecturale ou archéologique afin de suggérer le caractère hypothétique de l'information représentée: l'image est soit une proposition pour quelque chose qui n'existe pas encore, soit une hypothèse pour quelques chose qui n'existe plus (figure A.4).

Dans ce mémoire nous nous intéressons plus particulièrement à la création d'*animations* dans un style aquarelle, à partir de scènes 3D (chapitre 5) et de vidéo (chapitre 6). Cette recherche contribue au domaine du rendu non-photoréaliste ou rendu expressif qui a pour but de faciliter et accélérer la production d'images stylisées. Notre travail sur les animations stylisées a été

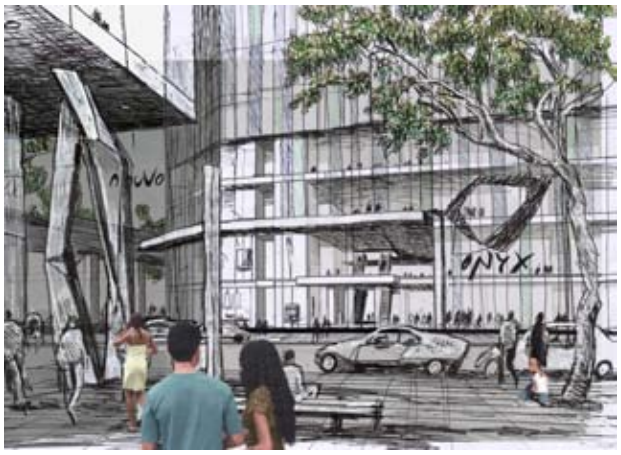


Figure A.4: *Dans cette illustration architecturale, un style crayonné est utilisé pour transmettre l'idée que le batiment n'est qu'une proposition, et des photographies de piétons lient cette proposition à la réalité. ©http://zlgdesign.files.wordpress.com/.*

initié par une collaboration avec le studio d'animation *Studio Brocéliande*[1], en partant du constat que beaucoup de bandes dessinées françaises et belges sont colorisées à l'aquarelle, ce qui leur confère un style bien particulier appécié du public. Le but de notre recherche est de reproduire l'aspect de ces bandes dessinées en animation pour permettre au Studio Brocéliande de produire des dessins animés qui s'adressent au même public.

4 Nos contributions

Dans son étude sur la bande dessinée, McCloud [McC94] positionne les trois principaux types d'images figuratives (réalistes, stylisées, simplifiées) dans un triangle délimité en bas par l'axe de la "représentation" qui correspond à *quoi* est illustré, et à gauche par l'axe de la "rétine" qui correspond à *comment* c'est illustré. Le coté droit du triangle est la limite du langage ou les images deviennent des symboles et du texte. Nous adoptons une représentation similaire pour organiser les différents types d'images que nous souhaitons créer et manipuler. La figure A.5 illustre notre espace de représentation qui est composé d'un axe allant du réalisme vers le stylisé, et d'un axe allant du complexe vers le simple. Notre but est de permettre à un utilisateur de naviguer dans cet espace pour créer les images qui correspondent à son intention.

Notre espace des représentations visuelles peut inclure tout type d'images figuratives. Nous pensons cependant que ce grand nombre d'images potentielles rend illusoire le développement d'une approche générique autorisant la navigation continue dans cet espace d'images. Notre approche consiste à **instancier des manipulations d'images spécifiques qui permettent la navigation d'un point de l'espace des représentations à un autre**. Les méthodes proposées dans

[1] http://www.studio-broceliande.fr/

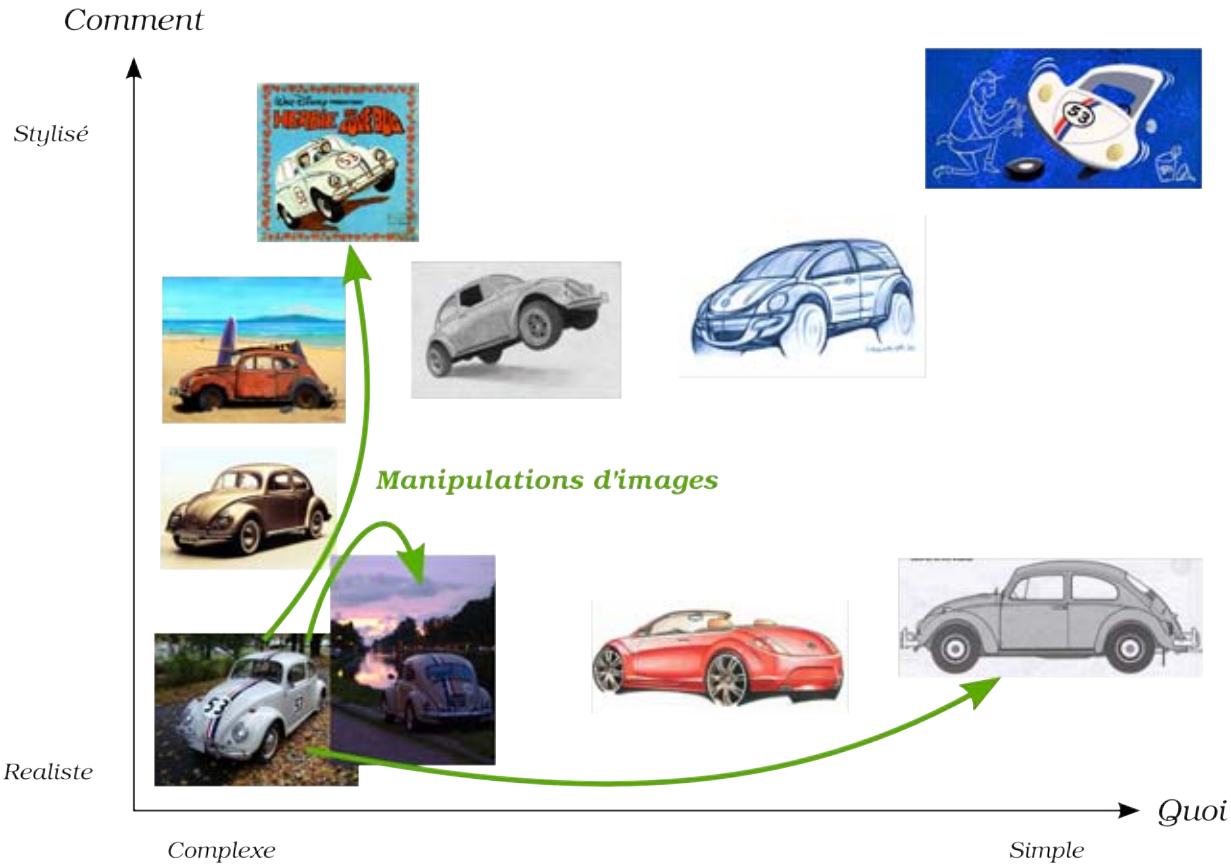


Figure A.5: *Nous organisons les images dans un espace 2D de stylisation et de simplification. Les manipulations d'images proposées dans ce mémoire permettent de naviguer d'un point de cet espace à un autre.*

ce manuscrit suivent une organisation commune en trois étapes. Dans la première étape, les caractéristiques de l'image nécessaires à la manipulation sont identifiées. Ces caractéristiques fournissent une connaissance sur le contenu de la scène représentée. Une connaissance partielle est dans beaucoup de cas suffisante. Par exemple, l'algorithme de stylisation de vidéo ne nécessite que la connaissance du mouvement apparent de la scène. Les caractéristiques de l'image peuvent être indiquées par l'utilisateur, fournies sous forme de modèles 3D, ou comme dans la plupart des méthodes proposées dans ce document, extraites de photographies ou vidéos en utilisant des algorithmes de vision par ordinateur. Une fois cette connaissance de la scène obtenue, l'image est manipulée au travers d'opérateurs qui peuvent créer, supprimer ou modifier le contenu visuel. Enfin, les images finales sont obtenues à partir de l'information manipulée grâce à des algorithmes de rendu dédiés. Ce mémoire est structuré autour des trois types d'images que nous souhaitons manipuler. La première partie traite des images simplifiées, la seconde des images stylisées, et enfin la troisième des images réalistes. Les manipulations d'images instanciées dans chaque partie contribuent à la création d'une variété de représentations visuelles, comme détaillé dans les paragraphes suivants.

Images simplifiées (chapitres 1, 2 et 3) La principale difficulté de la simplification d'image est d'identifier les structures importantes de l'image pour les discriminer des détails. Dans le chapitre 1 nous présentons une méthode pour simplifier les formes d'une vidéo sans introduire de clignotements dans l'animation. En considérant que les détails de formes correspondent aux petits éléments de l'image, nous montrons que **la simplification de formes dans une vidéo peut être effectuée par de simples filtres d'image bas niveau**, sans nécessiter l'extraction explicite des formes qui rend les méthodes existantes complexes à mettre en oeuvre. Ces filtres sont étendus au domaine temporel pour maintenir la stabilité et la continuité de l'animation.

En plus de cette approche géométrique de la simplification, le chapitre 2 propose une définition du détail fondée sur la perception visuelle. L'intuition principale de ce travail est de **représenter la structure de l'image au travers de ses contours**, et d'estimer l'importance de chaque contour au moyen d'un modèle de la perception visuelle. Cette approche est motivée par le fait que la majorité de l'information d'une image est contenue dans les contours, et que le système visuel humain est très sensible aux variations de contraste [Pal99]. Grâce à cet outil, l'utilisateur peut choisir quelles zones de l'image il souhaite mettre en avant (figure A.6).

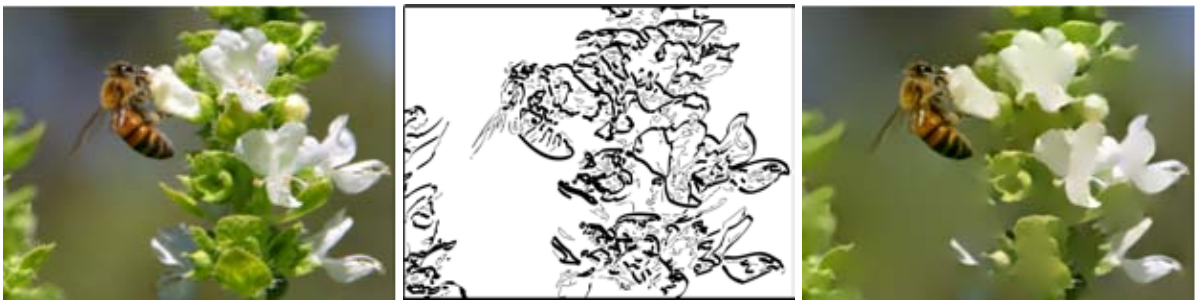


Figure A.6: *A partir d'une photographie (gauche), notre approche permet à l'utilisateur de manipuler la structure de l'image pour créer des représentations simplifiées. L'image du milieu est le résultat d'un rendu type dessin au trait où l'épaisseur des lignes est proportionnelle à leur importance. Dans l'image de droite, seuls les details de l'abeille sont préservés pour attirer l'attention. Image en provenance de www.pdphoto.org.*

Cette représentation d'une image par contours est étendue dans le chapitre 3 où nous décrivons une nouvelle primitive vectorielle, appelée *courbe de diffusion*. Une courbe de diffusion est une courbe géométrique dont chaque coté (gauche et droit) est colorisé. Une image couleur est obtenue en propageant les couleurs définies le long de ces courbes. Les courbes de diffusion facilitent l'enrichissement de dessins au trait avec des variations de couleurs complexes comme des effets d'ombrage ou de flou (figure A.7). Nous montrons que les courbes de diffusion permettent la représentation d'images réalistes dans un format vectoriel, ce qui reste un défi pour la plupart des primitives vectorielles existantes. Les courbes de diffusion sont aussi un outil de dessin intuitif qui permet aux artistes de travailler comme sur un support papier, en représentant les structures principales des images comme des dessins au trait avant de les coloriser. Enfin, notre approche permet la vectorisation automatique de photographies.

Figure A.7: *Des courbes de diffusion (gauche), et l'image couleur correspondante (droite). Des effets d'ombrages complexes indiquent les plis du tissu. ©Laurence Boissieux.*

Images stylisées (chapitre 4, 5 et 6) Cette thèse introduit deux méthodes pour styliser des vidéos et des animations 3D. Les deux méthodes traitent le problème de la *cohérence temporelle* qui reste l'un des défis majeurs du rendu non-photoréaliste. En animation traditionelle, les phénomènes aléatoires comme le grain du papier ou la taille des coups de pinceaux varient d'une image à l'autre et créent ce qu'on appelle des incohérences temporelles. Bien que les clignotements et vibrations produites par ces variations aléatoires peuvent être appéciés comme partie du style (voir par exemple les courts métrages de *Bill Plympton*), ces phénomènes peuvent rapidement gêner le spectateur. De plus, les artistes n'ont presque pas de contrôle sur ces effets de bord. Dans les animations générées par ordinateur, les marques de style comme les pigments ou les coups de pinceau ne sont pas placées aléatoirement sur chaque image mais animées d'une image à l'autre. Ceci permet un meilleur contrôle de la cohérence temporelle mais soulève une contradiction: les marques de style doivent suivre le mouvement 3D des objets mais préserver leur apparence 2D.

L'idée commune des deux approches présentées dans cette dissertation est que **résoudre cette contradiction localement dans le temps est suffisant pour obtenir une illusion convaincante de cohérence temporelle**. Avec un support temporel limité, la stylisation suit le mouvement 3D pendant suffisamment d'images pour produire une bonne perception du mouvement, mais est regénérée à intervalles réguliers pour éviter un éloignement de l'apparence 2D. Nous appliquons ce principe à deux algorithmes, chacun tirant avantage du type de données à styliser. Pour les scènes 3D, le chapitre 5 introduit un *méchanisme de zoom infini* simple et rapide qui produit des textures de stylisation avec une taille constante à l'écran (figure A.8). Cette technique masque la majorité des indices 3D causés par la projection perspective, et est particulièrement bien adaptée aux applications temps réel comme les jeux vidéo. Pour les vidéos, le chapitre 6 présente un

nouveau *schéma d'advection de texture* qui minimise les distorsions de textures quand une stylisation est animée le long du flux optique. La stylisation résultante préserve une forte apparence 2D pour n'importe quelle image de l'animation.

(a) ©Blacksad

(b) Rendu aquarelle
basé sur notre algorithme

Figure A.8: *Notre algorithme de stylisation de scènes 3D (b) est utilisé par le Studio Bro-celiande pour reproduire l'apparence des bandes dessinées françaises et belges (a).*

Images réalistes (chapitre 7) Dans le domaine de l'édition d'images réalistes, le chapitre 7 propose une nouvelle méthode pour estimer l'illumination et la réflectance d'une scène à partir d'une seule photographie. Estimer cette information, appelée *images intrinsèques*, est un problème sous-contraint et les méthodes automatiques sont mises en difficulté par les scènes naturelles complexes. Nous montrons que ce problème peut être résolu grâce à une combinaison d'**indications utilisateur** et d'hypothèses simples sur les **distributions de couleurs dans les images naturelles**. Plus particulièrement, l'utilisateur indique les régions de réflectance ou d'illumination constante, et ces marques sont propagées dans l'image par une minimisation d'énergie qui tend à imposer que les distributions de réflectance soient localement de dimension réduite. La connaissance tirée de cette décomposition permet des éditions d'image avancées comme le changement de l'éclairage ou des textures (figure A.9).

(a) Photographie (b) Indications utilisateur (c) Réflectance (d) Illumination (e) Changement de texture

Figure A.9: *Notre système propage des indications utilisateur (montrées en (b)) pour extraire à partir d'une seule image l'illumination et la réflectance de la sène (c-d). Dans (b), les indications blanches correspondent à des pixels pleinement éclairés, les indications bleues correspondent à des pixels partageant une même réflectance et les indications rouges marquent des pixels qui partagent une même illumination. Cette décomposition facilite des éditions d'image avancées comme le changement de textures (e).*

5 Conclusion

Une communication visuelle efficace se fait au travers d'images aux apparences variées (images réalistes, stylisées, simplifiées). Dans ce mémoire nous avons exploré différents moyens pour changer le style ou la complexité d'une image, comme visualisé dans la figure A.10. Nous avons proposé des méthodes pour transformer une image complexe en une représentation simplifiée (chapitre 1 et 2), et pour enrichir un dessin au trait avec des dégradés de couleur réalistes (chapitre 3). Nous avons aussi présenté deux méthodes pour produire des animations stylisées à partir de vidéos et de scènes 3D (chapitre 5 et 6), ainsi qu'une méthode assistée par l'utilisateur qui facilite la manipulation de l'éclairage et des couleurs des objets dans une photographie (chapitre 7).

De nombreuses autres manipulations d'images pourraient être proposées pour couvrir l'espace des représentations visuelles. Dans le future, nous aimerions explorer les zones "intermédiaires" de cet espace, et en particulier la convergence des algorithmes issus de la synthèse d'images réalistes et ceux issus de la synthèse d'images stylisées. La reproduction d'effets réalistes dans des styles artistiques comme la peinture ou le dessin (style *hyper-réaliste*, figure A.11) entrent par exemple dans cette direction de recherche.

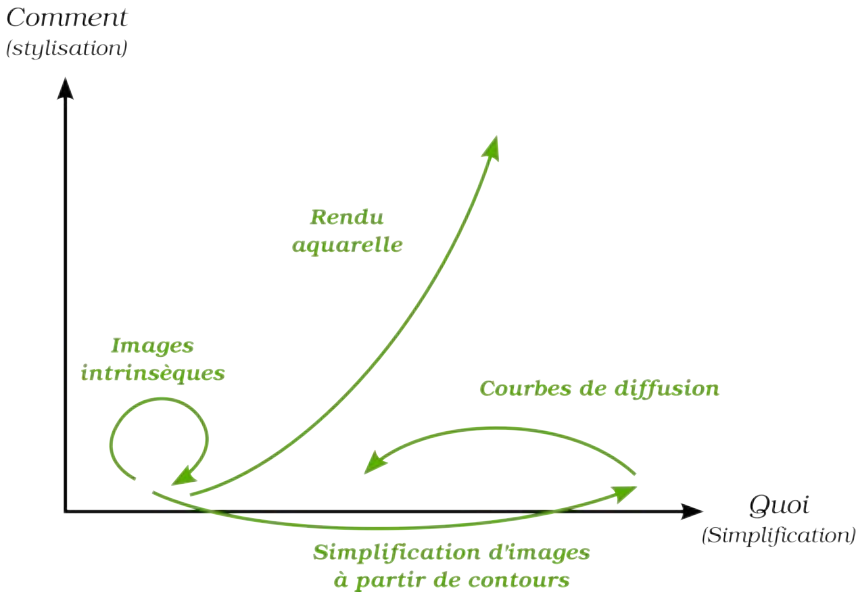


Figure A.10: *Visualisation des manipulations d'images proposées dans ce mémoire pour naviguer dans l'espace des représentations visuelles.*

Figure A.11: *Le style hyper-réaliste se situe à l'interface entre les images réalistes et les images stylisées.* Red Menu *by Ralph Goings, 1981.*

Bibliography

[ADA+04] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM TOG (Proc. of SIGGRAPH 2004)*, pages 294–302, 2004.

[AHJ+90] B. Alp, P. Haavisto, T. Jarske, K. Oistamo, and Y. A. Neuvo. Median-based algorithms for image sequence processing. In *SPIE Vol. 1360, Visual Communications and Image Processing*, pages 122–134, 1990.

[ARC06] Amit Agrawal, Ramesh Raskar, and Rama Chellappa. Edge suppression by gradient field transformation using cross-projection tensors. In *CVPR*, pages 2301–2308, 2006.

[AS01] Maneesh Agrawala and Chris Stolte. Rendering effective route maps: improving usability through generalization. *SIGGRAPH 2001*, pages 241–249, 2001.

[ASP07] Paul Asente, Mike Schuster, and Teri Pettit. Dynamic planar map illustration. *ACM TOG (Proc. of SIGGRAPH)*, 26(3):30, 2007.

[BBT09] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. Dynamic solid textures for real-time coherent stylization. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, pages 121–127, 2009.

[BCL07] Luc Buatois, Guillaume Caumon, and Bruno Lévy. Concurrent number cruncher: An efficient sparse linear solver on the gpu. In *High Performance Computation Conference*, 2007.

[Ber87] Fredrik Bergholm. Edge focusing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(6):726–741, 1987.

[BFSC04] Marcelo Bertalmio, Pere Fort, and Daniel Sanchez-Crespo. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proc. of 3DPVT*, pages 767–773, 2004.

[BGH03] J.A. Bangham, S.E. Gibson, and R.W. Harvey. The art of scale-space. In *British Machine Vision Conference*, pages 569–578, 2003.

[BHM00] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial (2nd ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[BKTS06] Adrien Bousseau, Matt Kaplan, Joëlle Thollot, and François X. Sillion. Interactive watercolor rendering with temporal coherence and abstraction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 141 – 149, 2006.

[BNTS07] Adrien Bousseau, Fabrice Neyret, Joëlle Thollot, and David Salesin. Video watercolorization using bidirectional texture advection. *ACM TOG (Proc. of ACM SIGGRAPH 2007)*, 26(3):104, 2007.

[Bou09] Adrien Bousseau. Non-linear aperture for stylized depth of field. In *SIGGRAPH 2009 - Technical talk*, 2009.

[BPD09] Adrien Bousseau, Sylvain Paris, and Frédo Durand. User assisted intrinsic images. *ACM TOG (Proc. of ACM SIGGRAPH Asia 2009)*, 28(5), 2009.

[BSCB00] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proc. of ACM SIGGRAPH 2000*, pages 417–424, 2000.

[BSM+07] Simon Breslav, Karol Szerszen, Lee Markosian, Pascal Barla, and Joëlle Thollot. Dynamic 2d patterns for shading 3d scenes. *ACM TOG (Proc. of ACM SIGGRAPH 2007)*, 26(3):20, 2007.

[BT78] H.G. Barrow and J.M. Tenenbaum. Recovering intrinsic scene characteristics from images. *Computer Vision Systems*, pages 3–26, 1978.

[CAC+02] Yung-Yu Chuang, Aseem Agarwala, Brian Curless, David H. Salesin, and Richard Szeliski. Video matting of complex scenes. *ACM TOG (Proc. SIGGRAPH 2002)*, 21(3):243–248, July 2002.

[Can86] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[Car88] Stefan Carlsson. Sketch based coding of grey level images. *Signal Processing*, 15(1):57–83, 1988.

[CCSS01] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *CVPR*, 2001.

[CDH06] Liviu Coconu, Oliver Deussen, and Hans-Christian Hege. Real-time pen-and-ink illustration of landscapes. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 27–35, 2006.

[CGL+08] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather S. Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *ACM TOG (Proc. of SIGGRAPH 2008)*, 27(3), 2008.

[CH03] John P. Collomosse and Peter M. Hall. Cubist style rendering from photographs. *IEEE Transactions on Visualization and Computer Graphics*, 9(4), 2003.

[CH05] John P. Collomosse and Peter M. Hall. Genetic paint: A search for salient paintings. In *Proc. of EvoMUSART*, pages 437–447, 2005.

[CRH05] J. P. Collomosse, D. Rowntree, and P. M. Hall. Stroke surfaces: Temporally coherent artistic animations from video. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):540–549, September 2005.

[CSD+09] Forrester Cole, Kevin Sanik, Doug DeCarlo, Adam Finkelstein, Thomas Funkhouser, Szymon Rusinkiewicz, and Manish Singh. How well do line drawings depict shape? *ACM TOG (Proc. of SIGGRAPH 2009)*, 28(3), 2009.

[CTP+03] Matthieu Cunzi, Joëlle Thollot, Sylvain Paris, Gilles Debunne, Jean-Dominique Gascuel, and Frédo Durand. Dynamic canvas for immersive non-photorealistic walkthroughs. In *Graphics Interface*, 2003.

[Dan99] Eric Daniels. Deep canvas in disney's tarzan. In *ACM SIGGRAPH 99: Sketches and applications*, page 200, 1999.

[DCFR07] Christopher DeCoro, Forrester Cole, Adam Finkelstein, and Szymon Rusinkiewicz. Stylized shadows. In *NPAR*, pages 77–83, 2007.

[DD02] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. *ACM TOG (Proc. of SIGGRAPH 2002)*, 21(3), 2002.

[DFRS03] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM TOG (Proc. of SIGGRAPH 2003)*, 22(3), 2003.

[DOM+01] Frédo Durand, Victor Ostromoukhov, Mathieu Miller, Francois Duranleau, and Julie Dorsey. Decoupling strokes and high-level attributes for interactive traditional drawing. In *EGSR 2001*, pages 71–82, 2001.

[DP73] David Douglas and Thomas Peucker. Algorithms for the reduction of the number of points required for represent a digitzed line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[DS02] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. *ACM TOG (Proc. of SIGGRAPH 2002)*, 21(3):769–776, 2002.

[Dur00] Frédo Durand. The art and science of depiction. Unpublished manuscript, 2000.

[EG01] J. H. Elder and R. M. Goldberg. Image editing in the contour domain. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):291–296, 2001.

[Eld99] James H. Elder. Are edges incomplete? *International Journal of Computer Vision*, 34(2-3):97–122, 1999.

[EZ98] James H. Elder and Steven W. Zucker. Local scale control for edge detection and blur estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(7):699–716, 1998.

[Fan06] Hui Fang. Rototexture: Automated tools for texturing raw video. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1580–1589, 2006.

[FAR07] Raanan Fattal, Maneesh Agrawala, and Szymon Rusinkiewicz. Multiscale shape and detail enhancement from multi-light image collections. *ACM TOG (Proc. of SIGGRAPH 2007)*, 26(3), 2007.

[Fat08] Raanan Fattal. Single image dehazing. *ACM TOG (Proc. of SIGGRAPH 2008)*, 27(3), 2008.

[FDL04] Graham D. Finlayson, Mark S. Drew, and Cheng Lu. Intrinsic images by entropy minimization. In *ECCV*, pages 582–595, 2004.

[FF87] M. A. Fischler and O. Firschein. *Intelligence: The Eye, the Brain and the Computer*. Addison-Wesley, 1987.

[FH04] Hui Fang and John C. Hart. Textureshop: Texture synthesis as a photograph editing tool. *ACM TOG (Proc. of SIGGRAPH 2004)*, 23(3):354–359, 2004.

[FHD02] Graham D. Finlayson, Steven D. Hordley, and Mark S. Drew. Removing shadows from images. In *ECCV*, 2002.

[FLW02] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. *ACM TOG (Proc. of SIGGRAPH)*, 21(3):249–256, 2002.

[FMH97] Pamela Frorer, Michelle Manes, and Orit Hazzan. Revealing the faces of abstraction. *International Journal of Computers for Mathematical Learning*, 2(3):217–228, 1997.

[FMS01] Bert Freudenberg, Maic Masuch, and Thomas Strothotte. Walk-Through Illustrations: Frame-Coherent Pen-and-Ink Style in a Game Engine. *Computer Graphics Forum (Proc. of Eurographics 2001)*, 20(3):184–191, 2001.

[GASP08] Floraine Grabler, Maneesh Agrawala, Robert W. Sumner, and Mark Pauly. Automatic generation of tourist maps. *ACM TOG (Proc. of SIGGRAPH 2008)*, 27(3), 2008.

[GVWD06] M. Grundland, R. Vohra, G. P. Williams, and N. A. Dodgson. Cross dissolve without cross fade: preserving contrast, color and salience in image compositing. *Computer Graphics Forum : Proc. of Eurographics 2006*, 25(3), 2006.

[GWL+03] Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys. A multigrid solver for boundary value problems using programmable graphics hardware. In *Graphics Hardware*, pages 102–111, 2003.

[HE04] James Hays and Irfan Essa. Image and video based painterly animation. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 113–120, 2004.

[Her98] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH '98*, pages 453–460, 1998.

[HMP+08] Eugene Hsu, Tom Mertens, Sylvain Paris, Shai Avidan, and Frédo Durand. Light mixture estimation for spatially varying white balance. *ACM TOG (Proc. of SIGGRAPH 2008)*, 27(3), 2008.

[Hod03] Elaine R. S. Hodges. *Guild Handbook of Scientific Illustration*. John Wiley & Sons, Inc, 2003.

[Hor86] Berthold K. Horn. *Robot Vision*. MIT Press, 1986.

[HP00] Aaron Hertzmann and Ken Perlin. Painterly rendering for video and interaction. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 7–12, 2000.

[HRRG08] Charles Han, Eric Risser, Ravi Ramamoorthi, and Eitan Grinspun. Multiscale texture synthesis. *ACM TOG (Proc. of ACM SIGGRAPH 2008)*, 27(3):1–8, 2008.

[HSZ87] R. M. Haralick, S. R. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(4):532–550, 1987.

[HZ00] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *ACM SIGGRAPH 2000*, pages 517–526, 2000.

[IMG00] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically reparameterized light fields. In *ACM SIGGRAPH 2000*, pages 297–306, 2000.

[JEH01] Bruno Jobard, Gordon Erlebacher, and M. Yousuff Hussaini. Lagrangian-eulerian advection for unsteady flow visualization. In *VIS '01: Conference on Visualization '01*, pages 53–60, 2001.

[KCC06] Hyung W. Kang, Charles K. Chui, and Uday K. Chakraborty. A unified scheme for adaptive stroke-based rendering. *The Visual Computer (Proc. of Pacific Graphics 2006)*, 22(9), 2006.

[KD79] J. J. Koenderink and A. J. Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32(4):211–216, 1979.

[KFCO+07] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2d exemplars. *ACM TOG (Proc. of ACM SIGGRAPH 2007)*, 26(3):2, 2007.

[KL08] H. Kang and S. Lee. Shape-simplifying image abstraction. *Computer Graphics Forum (Special issue on the Pacific Graphics 2008)*, 27(7), 2008.

[KLK+00] Allison W. Klein, Wilmot W. Li, Michael M. Kazhdan, Wagner T. Correa, Adam Finkelstein, and Thomas A. Funkhouser. Non-photorealistic virtual environments. In *ACM SIGGRAPH 2000*, pages 527–534, 2000.

[Koe84] Jan J. Koenderink. The structure of images. *Biological Cybernetics*, 50(5):363–370, 1984.

[Kok98] Anil C. Kokaram. *Motion Picture Restoration: Digital Algorithms for Artefact Suppression in Degraded Motion Picture Film and Video*. Springer-Verlag, London, UK, 1998.

[Kra07] Jeff Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, 2007.

[KRFB05] Erum Khan, Erik Reinhard, Roland Fleming, and Heinrich Bülthoff. Image-based material editing. *ACM TOG (Proc. of SIGGRAPH 2005)*, 24(3):148, 2005.

[KST08] Michael Kolomenkin, Ilan Shimshoni, , and Ayellet Tal. Demarcating curves for shape illustration. *ACM TOG (Proc. of ACM SIGGRAPH ASIA 2008)*, 2008.

[KWT87] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

[LD06] Thomas Luft and Oliver Deussen. Real-time watercolor illustrations of plants using a blurred depth test. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 11–20, 2006.

[LHM09] Yu-Kun Lai, Shi-Min Hu, and Ralph R. Martin. Automatic and topology-preserving gradient mesh generation for image vectorization. *ACM TOG (Proc. of SIGGRAPH)*, 28(3), 2009.

[Lin98] Tony Lindeberg. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):117–154, 1998.

[Lit97] Peter C. Litwinowicz. Processing images and video for an impressionist effect. In *SIGGRAPH 97*, pages 407–414, 1997.

[LL06] Gregory Lecot and Bruno Levy. Ardeco: Automatic Region DEtection and COnversion. In *Proc. of EGSR*, pages 349–360, 2006.

[LLW04] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM TOG (Proc. of SIGGRAPH 2004)*, 23(3):689–694, 2004.

[LLW08] Anat Levin, Dani Lischinski, and Yair Weiss. A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2008.

[LM71] Edwin H. Land and John J. McCann. Lightness and retinex theory. *Journal of the optical society of America*, 61(1), 1971.

[LM01] Eric B. Lum and Kwan-Liu Ma. Non-photorealistic rendering using watercolor inspired textures and illumination. In *Pacific Graphics*, pages 322–331, 2001.

[LMHB00] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 13–20, 2000.

[LW07] Anat Levin and Yair Weiss. User assisted separation of reflections from a single image using a sparsity prior. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(9):1647–1654, 2007.

[LWQ+08] Xiaopei Liu, Liang Wan, Yingge Qu, Tien-Tsin Wong, Stephen Lin, Chi-Sing Leung, and Pheng-Ann Heng. Intrinsic colorization. *ACM TOG (Proc. of ACM SIGGRAPH Asia 2008)*, 27(5), 2008.

[Mar82] David Marr. *Vision*. Freeman, 1982.

[MB95] Nelson Max and Barry Becker. Flow visualization using moving textures. In *Proc. of the ICASW/LaRC Symposium on Visualizing Time-Varying Data*, pages 77–87, 1995.

[McC94] Scott McCloud. *Understanding comics : the invisible art*. Harper Perennial, 1994.

[Mei96] Barbara J. Meier. Painterly rendering for animation. In *SIGGRAPH 96*, pages 477–484, 1996.

[MH80] D. Marr and E. C. Hildreth. Theory of edge detection. *Proc. of the Royal Society of London. Biological Sciences*, 207:187–217, 1980.

[MIA+08] Ross Maciejewski, Tobias Isenberg, William M. Andrews, David S. Ebert, Mario Costa Sousa, and Wei Chen. Measuring stipple aesthetics in hand-drawn and computer-generated images. *IEEE Computer Graphics and Applications*, 28(2):62–74, 2008.

[MP08] James McCann and Nancy S. Pollard. Real-time gradient-domain painting. *ACM TOG (Proc. of SIGGRAPH)*, 27(3), 2008.

[MTC07] Ankit Mohan, Jack Tumblin, and Prasun Choudhury. Editing soft shadows in a digital photograph. *IEEE Computer Graphics and Applications*, 27(2):23–31, 2007.

[Ney03] Fabrice Neyret. Advected textures. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*, pages 147 – 153, july 2003.

[NLB+05] Ren Ng, Marc Levoy, Mathieu Brédif, Gene Duval, Mark Horowitz, and Pat Hanrahan. Light field photography with a hand-held plenoptic camera. Technical report, 2005.

[OBBT07] Alexandrina Orzan, Adrien Bousseau, Pascal Barla, and Joëlle Thollot. Structure-preserving manipulation of photographs. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 103–110, 2007.

[OBS04] Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. Ridge-valley lines on meshes via implicit surface fitting. *ACM TOG (Proc. of ACM SIGGRAPH 2004)*, pages 609–612, 2004.

[OBW+08] Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. *ACM TOG (Proc. of ACM SIGGRAPH 2008)*, 27(3), 2008.

[Ola05] Marc Olano. Modified noise for evaluation on graphics hardware. In *Graphics hardware*, pages 105–110, 2005.

[OST93] Mehmet K. Ozkan, M. Ibrahim Sezan, and A. Murat Tekalp. Adaptive motion-compensated filtering of noisy image sequences. *IEEE transactions on circuits and systems for video technology*, 3(4):277–290, 1993.

[OW04] Ido Omer and Michael Werman. Color lines: Image specific color representation. In *CVPR*, pages 946–953, 2004.

[Pal99] Stephen E. Palmer. *Vision Science: Photons to Phenomenology*. The MIT Press, 1999.

[PB06] Brian Price and William Barrett. Object-based vectorization for interactive image editing. *Visual Computer (Proc. of Pacific Graphics)*, 22(9):661–670, 2006.

[PCPN07] Giuseppe Papari, Patrizio Campisi, Nicolai Petkov, and Alessandro Neri. A biologically motivated multiresolution approach to contour detection. *EURASIP Journal on Advances in Signal Processing*, 2007:Article ID 71828, 28 pages, 2007.

[Per85] Ken Perlin. An image synthesizer. *Computer Graphics (Proc. of ACM SIGGRAPH 85)*, 19(3):287–296, 1985.

[PFS03] Oscar Meruvia Pastor, Bert Freudenberg, and Thomas Strothotte. Real-time animated stippling. *IEEE Computer Graphics and Applications*, 23(4):62–68, 2003.

[PGB03] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM TOG (Proc. of SIGGRAPH)*, 22(3):313–318, 2003.

[PHWF01] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *ACM SIGGRAPH 2001*, pages 579–584, 2001.

[PM90] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

[RIY04] Ramesh Raskar, Adrian Ilie, and Jingyi Yu. Image fusion for context enhancement and video surrealism. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 85–152, 2004.

[Rom03] Bart ter Haar Romeny. *Front-End Vision and Multi-Scale Image Analysis*. Kluwer Academic Publishers, 2003.

[SA93] Pawan Sinha and Edward Adelson. Recovering reflectance and illumination in a world of painted polyhedra. In *ICCV*, pages 156–163, 1993.

[SCS90] T. Simchony, R. Chellappa, and M. Shao. Direct analytical methods for solving poisson equations in computer vision problems. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 12(5):435–446, 1990.

[SD04] Anthony Santella and Doug DeCarlo. Visual interest and npr: an evaluation and manifesto. In *NPAR*, pages 71–150, 2004.

[Sel03] Peter Selinger. *Potrace: a polygon-based tracing algorithm*, 2003.

[She64] Roger N. Shepard. Circularity in judgments of relative pitch. *The Journal of the Acoustical Society of America*, 36(12):2346–2353, 1964.

[Sim92] Karl Sims. Choreographed image flow. *The Journal of Visualization and Computer Animation*, 3(1):31–43, 1992.

[SL08] Yael Shor and Dani Lischinski. The shadow meets the mask: Pyramid-based shadow removal. *Computer Graphics Forum (Proc. of Eurographics)*, 27(3), 2008.

[SLWS07] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM TOG (Proc. of SIGGRAPH)*, 26(3):11, 2007.

[Sta99] Jos Stam. Stable fluids. In *SIGGRAPH 99*, pages 121–128, 1999.

[STL08] Li Shen, Ping Tan, and Stephen Lin. Intrinsic image decomposition with non-local texture cues. In *CVPR*, 2008.

[SV92] Jean Serra and Luc Vincent. An overview of morphological filtering. *Circuits Syst. Signal Process.*, 11(1):47–108, 1992.

[TFA05] Marshall F. Tappen, William T. Freeman, and Edward H. Adelson. Recovering intrinsic images from a single image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9), 2005.

[TFF08] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV*, page 839, 1998.

[VBTS07] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. Dynamic point distribution for stroke-based rendering. In *EGSR 2007*, pages 139–146, 2007.

[vDT96] M. van Droogenboeck and H. Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recognition Letters*, 17:1451–1460, 1996.

[WA94] J. Y. A. Wang and E. H. Adelson. Representing Moving Images with Layers. *The IEEE Transaction on Image Processing Special Issue: Image Sequence Compression*, 3(5):625–638, 1994.

[Wan95] Brian A. Wandell. *Foundations of Vision*. Sinauer Associates, 1995.

[Wei01] Yair Weiss. Deriving intrinsic images from image sequences. In *ICCV*, pages 68–75, 2001.

[Wei06] Ben Weiss. Fast median and bilateral filtering. *ACM TOG (Proc. of SIGGRAPH 2006)*, 25(3):519–526, 2006.

[WJFC02] Allison W.Klein, Peter-Pike J.Sloan, Adam Finkelstein, and Michael F. Cohen. Stylized video cubes. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*, pages 15–22, 2002.

[WLL+06] Fang Wen, Qing Luan, Lin Liang, Ying-Qing Xu, and Heung-Yeung Shum. Color sketch generation. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*, pages 47–54, 2006.

[WOBT09] Holger Winnemöller, Alexandrina Orzan, Laurence Boissieux, and Joëlle Thollot. Designing and draping textures in 2d images. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)*, 2009.

[WOG06] Holger Winnemöller, Sven C. Olsen, and Bruce Gooch. Real-time video abstraction. *ACM TOG (Proc. of SIGGRAPH 2006)*, 25(3):1221–1226, 2006.

[WS94] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. *Computer Graphics*, 28:91–100, 1994.

[WTBS07] Tai-Pang Wu, Chi-Keung Tang, Michael S. Brown, and Heung-Yeung Shum. Natural shadow matting. *ACM TOG*, 26(2):8, 2007.

[WXSC04] Jue Wang, Yingqing Xu, Heung-Yeung Shum, and Michael Cohen. Video tooning. *ACM TOG (Proc. of SIGGRAPH 2004)*, 23(3):574 – 583, 2004.

[WY04] Qing Wu and Yizhou Yu. Feature matching and deformation for texture synthesis. *ACM TOG (Proc. of ACM SIGGRAPH 2004)*, 23(3):364–367, 2004.

[YM98] Yizhou Yu and Jitendra Malik. Recovering photometric properties of architectural scenes from photographs. In *ACM SIGGRAPH 98*, pages 207–217, 1998.

[Zen86] Silvano Di Zenzo. A note on the gradient of a multi-image. *Computer Vision, Graphics, and Image Processing*, 33(1):116–125, 1986.