IMAGE EDITING AND INTERACTION TOOLS FOR VISUAL EXPRESSION

BY

ZICHENG LIAO

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

Doctoral Committee:

Professor David Forsyth, Chair
Dr. Hugues Hoppe, Microsoft Research
Professor John Hart
Assistant Professor Derek Hoiem
Assistant Professor Svetlana Lazebnik

ABSTRACT

Digital photography is becoming extremely common in our daily life. However, images are difficult to edit and interact with. From a user's perspective, it is important to *interact* freely with the images on his/her smartphone or ipad. In this thesis we develop several image editing and interaction systems with this idea in mind. We aim for creating visual models with pre-computed internal structures such that interaction is readily supported. We demonstrate that such interactable models, driven by a user's hand, can render powerful visual expressiveness, and make static pixel arrays much more fun to play with.

The first system harnesses the editing power of vector graphics. We convert raster images into a vector representation using Loop's subdivision surfaces. An image is represented by a multi-resolution feature-preserving sparse control mesh, with which image editing can be done at semantic level. A user can easily put a smile on a face image, or adjust the level of scene abstractness through a simple slider. The second system allows one to insert an object from image into a new scene. The key is to correct the shading on the object such that it goes consistently with the scene. Unlike traditional approach, we use a simple shape to capture gross shading effects and a set of shading detail images to account for visual complexities. The high-frequency nature of these detail images allows a moderate range of interactive composition effects without causing alarming visual artifacts. The third system is on video clips instead of a single image. We proposed a fully automated algorithm to create video loops from short (5-second) videos. We then introduce a novel media format called progressively dynamic video, which encodes a wide spectrum of looping videos, ranging from a static scene to a highly animated video. A user can adjust the level of dynamism in a scene, or locally adjust motion configuration, based on personal taste or mood.

To my mother and father.

ACKNOWLEDGMENTS

PhD is a long journey. People all start similarly; it is the ends that diversify along the path. "Would you still choose to take the PhD, had time flied back to the beginning of it?" Asked by a friend one day in my fifth year. "Yes." I answered, with a clear and firm mind.

That answer would not have been that easy without all the people I met during the years.

I am most in debt to my academic advisor, David Forsyth, who consistently inspires me with brilliant ideas and enormous enthusiasm every time I run into his office with frustration, who is generous on my stubbornness and my many childish blunders, whose willingly taking me as his student at the end of my second year had saved me from quitting PhD.

I am also in debt to Professor Yizhou Yu. Professor Yu provided me the precious chance to start at this prestigious department. His strictness and perseverance laid down the standard for me in research and influenced me all over the years. Years after his leaving from UIUC, I still cannot help turning to his advise when I am in doubt. Upon he leaving for HKU, he introduced me to Hugues Hoppe.

Meeting Hugues is no doubt one of the luckiest things that have ever happened to me. I learn from every aspect of him: his extremely high standards, his critical thinking, his concise and precise writing style, his disciplined way of organizing a research project, his amiable and mild character, etc. Even his unlearnable smartness taught me to be a humble person, which I was nothing like before.

It has been very rewarding to have Professor John Hart on my committee, whose in-depth comments demonstrate to me a philosophical way of looking into a problem without having to know every detail of it; and Professor Derek Hoiem, whose solidness and mild character

makes an image of a respectful professor. His computer vision course material turned out be to what I often look up to for technical clarity. It is a regret I could not have grasped every opportunity to talk to them in the hallway. It has been a valuable experience to learn from Professor Jiawei Han in a research project for his powerful communication skills and his character of "sovereignty". Professor Chengxiang Zhai, whom I have rarely spoke with face to face, would greet me with an amiable smile and leaves me an image of an established professor who is just as friendly as anyone in the street. It has also been a valuable experience to work on the same floor with Richard Seleski. He would listen attentively to every person's talk and make good points out of it. From him I see a perfect embodiment of an excellent researcher and a great manager. I also owe my thanks to Neel Joshi, without whom my first SIGGRAPH paper could not have been the way it is. Neel also acted as an "older brother" when I was at difficult occasions in the internships at MSR, which I would never forget.

Dennis DeCoste, Eric Sun, Zheng Shao, Rong Yan, these are my mentors and people I learned a lot during my two internships at Facebook. Dennis, whose hired me as his intern in the first year of my PhD, opened me to an extraordinarily valuable journey to this great company in her early time, and to Silicon valley. It was also through those experience I realized the valley is not the place I should stay long. Without those early experience I would have made very different career decisions then. My facebook internships were an irreplaceable page of my happy memories of the years, through which I made many great friends: Zhenhui, Yuntao, Jinfeng, Zuoning, Aileen, Yue, Chi, etc. Of course I shall not forget to mention the friends I made during my internships at MSR: Hongning, Hui, Jia-Bin, Jingjing, Yuwei, Hongwei, Eric, Francisco, Fangbo, etc. Thank you all for the precious friendship and bringing to me those memorable days.

I would like to thank the friends of my last year in Champaign: Xiaohui, Ziyi, Ting, Tianxiao, Jiajun, Kaihao. It was the luckiest thing that I came to share an apartment of warmth and friendship with Xiaohui, who also gave me encouragement and advice in the time I need them the most.

Last but not least, I owe my sincere thanks to collaborators at UIUC: Kevin Karsch, Yang Wang, Ali Fahadi, Jason Rock, Ian Endres. I am in debt to Professor Qunsheng Peng who

guided me along my academia path and consolidated my mind of returning to China. I am in debt to Professor Zhangye Wang who first introduced me to the prestigious CAD & CG lab. It was where the journey started.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

We are entering a new era of visual media. Pervasive mobile devices make image acquisition effortless at almost no cost. With social networks, people create online albums for personal travels, parties, advertising and even campaigns. As a new visual media, digital image plays an increasingly important role in our communication, interaction and expression with the outer world. To keep up with its rapid expansion, the research community has endeavored to develop accompanying tools that assist user editing with the media. We follow this line of research and demonstrate a few systems that enhance the expressiveness of the media and empower user interaction with it.

The first system extends *image vectorization* research. Previous work includes binary image conversion with a piecewise-constant representation and parametric curves [2], piecewise-linear trangulation [3], and higher-order representations (gradient mesh [5], diffusion curves [6], patch-based thin plate splines [7]). None of the existing works offers a representation that handles curvilinear features efficiently, preserves sharp edges and guarantees non-edge color continuity. We propose a subdivision-based approach that achieves all of these. We also expand the control mesh of the vector image into a pyramid structure (analogous to Laplacian pyramid but on meshes). This representation supports image signal processing directly on a vector image, in addition to feature-based object deformation, colorization, abstraction and stylization.

The second system offers a hybrid approach to *image relighting*. We often encounter ap-

plications of compositing two images into a new one. Relighting is necessary to close the gap of illumination inconsistency. However, physically-based relighting requires a strict rendering environment setup: object shape, reflectance, illumination, etc. (or a laborious process to capture the reflectance function [11] in image-based rendering approach). Shape and reflectance (BRDF, or Lambertian albedo) estimation from a single image are notoriously known unsolved inverse problems in computer vision. We propose an approximate relighting solution that circumvents these barriers, yet achieves plausible visual realism. The approach is based on observations on human visual perception and the low-dimensional illumination cone theory [12, 13]. To the end, we use a coarse shape to capture the essential shading effects and combine image-based composition for detail manipulation.

The third system offers the first fully automated solution to video looping and cinemagraph creation, following [14, 9, 10, 15, 16, 17]. Video looping uses a short video sequence to synthesize an infinitely loopable video, similar to texture synthesis in the temporal domain. A cinemagraph [18] combines looping elements and static background to capture a living moment within. An automated solution to the two has promising applications in creating animated background for desktops, slideshows, or homepage of commercial platforms (e.g., bing.com). More important, we invent a new media format: *progressively dynamic video*. The media format bridges a still image and a highly animated scene with a series of progressively activated video loops. It is concise in storage and supports interactive user control. With a slider a user could adjust the level of scene dynamism based on personal state or mood. A user could also arbitrarily freeze or activate a local element.

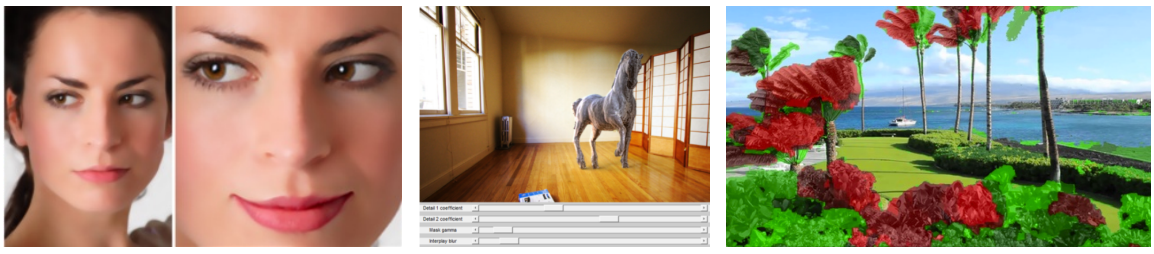


Figure 1.1: Demo projects. Left: face editing with image vectorization; Middle: object relighting and apparent detail editing with a plausible re-shading model; Right: video looping and adjustable scene dynamism with a nested segmentation.

1.2 Connecting the dots

Underlying these work is a pursuit of *building interactable visual objects and scenes*. We believe that visual media should represent scenes in more lively ways, that the projection of the world on screen should be manipulated easily and naturally in ways people would like to. This is certainly am ambitious goal. The three works demonstrate our efforts towards it: the first project allows a viewer to adjust the level of abstractness of a scene, or pick up an object and change its color or shape at semantical level; the second project allows an artist to pick up an object from a single image and insert it into a new scene. The inserted object is re-shaded to match the target scene. The apparent material and surface details can be interactively enhanced or smoothed to reach a desired effect; in the third work, we capture scene activity with a video sequence and transform it into a progressive structure that allows a user to seamlessly turn the scene from highly dynamic to completely static and vice versa.

The term "interactable" is used to emphasize the intrinsic property of a scene representation that supports user interaction. To build interactable models from images or videos is a challenging task. Three challenges are identified. (1) at constructional level, how to *extract semantic features* from pixels and organize the information in an appropriate *representation*; (2) at perception level, how to build a model that renders the desired *visual impression*; and (3) at interaction level, how to create model that embodies design ideology and allows users to *express internal emotion* through interaction with it. We expand on these three items as follows:

Structure extraction and representation Structure recovery is the basis for high-level scene interaction. We mainly rely on existing techniques for this part. Of equal importance is to put the extracted structural information into an appropriate representation, which usually goes hand-in-hand with the former and is crucial for functionality.

Different representational schemes are tried out. In the image vectorization work, we convert discrete color signal into a *parametric* surface representation – a variant of Loop's subdivision surface [19]. Such a vector representation supports superresolution and facili-

tates several image editing operations. In the relighting work, image signal is *decomposed* into several intrinsic components, or factors that explains the image. These intrinsic components can then be used to re-generate new images by modifying one parameter or another. In the progressively dynamic video work, the representation is based on temporal and spatial *partitioning* (or segmentation). The temporal partitioning produces the per-pixel loop structure; the spatial partitioning produces the nested looping structure that defines a series of video loops.

There is one representational scheme found particularly useful in all the works: the *level-of-"detail"* representation. In the vectorization work, the control mesh of the vector image is expanded into a base mesh plus a sequence of detail vectors sorted by frequency. This representation supports signal processing and multi-resolution editing on the control mesh, resulting in interesting image processing effects (but over a vector representation). In the relighting work, the shading of an object is decomposed into a coarse base layer, plus two detail layers of increasing spatial frequency. The detail layers, due to its high frequency nature, can be used in an image-based composition for tuning visual effects in a moderate range without corrupting visual perception. In the progressively dynamic video work, "detail" refers to scene dynamism. A novel level-of-dynamism video looping structure is presented and offers interesting visual effects as well as user interaction.

Visual impression Since an image is to be viewed through the eyes and perceived by the brain, how is visual information processed, interpreted and represented in human brain must be taken into consideration. After all, the goal of image depiction is to synthesize the essence of our visual experience. Neuroscientists and psychologists have found that a physically correct image (neutral image) may not be the most effective one for making a visual impression (e.g., photograph VS caricatured faces), and that the brain is insensitive to certain types of inconsistent information in a picture (e.g., shadows, highlights, or shading details).

Two points can be exploited here: (1) *how to extract essential features of a scene;* (2) *what bits can we safely neglect without corrupting visual perception.* For the first point,

essential features of a scene are defined differently in different visual formats. In the video looping work, we depict a scene by identifying temporally looping elements in the scene (e.g., swinging flowers, dancing person, pulsing lights) while deliberately freezing the rest. As a result, attention is drawn towards the looping elements and freed from distraction of the background. The inspiration was taken from art work of cinemagraphs by Beck and Burg [18].For the second point, we illustrate it in the object relighting work. In this work, we use a rough shape-from-contour model plus a set of shading detail images to reshade an surface in new environments. The reshading model is physically incorrect but visually plausible. It is based on the fact that human vision system checks shadow and shading signal with a simpler mechanism than true physics [20]. Especially, it handles cast shadows rapidly in early vision process and discard them so that they do not interfere in further processing [21].

Emotional expression

The organization of interactable visual modeling by representation, impression and expression was inspired by Itten's color aesthetics theory (The Art of Color, 1974):

> *Color aesthetics may be approached from these three directions:*
>
> *Impression (visually)*
>
> *Expression (emotionally)*
>
> *Construction (symbolically)*
>
> *... Symbolism without visual accuracy and without emotional force would be mere anemic formalism; visually impressive effect without symbolic verity and emotional power would be banal imitative naturalism; emotional effect without constructive symbolic content or visual strength would be limited to the plane of sentimental expression.*

Emotional expression was the key that drew my interest, as it unveiled my longtime wondering of what underlies the motion control part of the video looping work: the motion control tool allows a user to adjust the level of dynamism (imagine a peaceful mind turns lively scene to a tranquil state and an excited one does the opposite) and arbitrarily override

local motion status (e.g., activating a swaying palm tree or freezing a rotating Pinocchio) based on personal taste or mood. The very nature of motion as an emotion carrier offers effective means for users to express their inner states through their configuration with a scene. And for the very basic human need, the significance of such expressive tool cannot be overstated.

An expressive tool should accommodate and exhibit user's internal emotion through interaction. Identifying the emotion carrier is the key, like color and form to painting, tone and scale to music.

With these concepts in mind, it is natural to distinguish two types of interactability: *functional* and *expressive*. Functional interactability is by those who lack an emotion carrier; expressive interactability by those who possess it. Without strict verification, my conjecture is that expressive interactability is scarce in visual modeling and is under-acknowledged by the community. Most existing interaction/editing tools are functional. A tool, existing or not, that allows a user to turn a view of summer into fall, or winter into spring, etc., would entitle to the latter, as people would like to view a scene that reflects their state of being, and the four seasons are nature's most vivid expression of life states in a cycle.

It is only when expression becomes possible does interactable visual modeling meets its higher level goal, where science and engineering meets the principles of art, and the color aesthetics theory applies to applications of visual modeling.

CHAPTER 2

IMAGE VECTORIZATION

2.1 Introduction

There has been a recent resurgence of vector-based graphical content in personal computers and on the Internet. For example, major operating systems have increasingly adopted vector graphics in their user interface, and Adobe Flash has strengthened support for vector graphics in rich internet applications. Vector-based drawing tools, such as Adobe Illustrator and CorelDRAW, enjoy immense popularity among artists and designers. Such a wide range of applications is made possible by the fact that vector graphics is both editable and scalable. Editability is a high priority for artists and designers who wish to conveniently produce visual content with user interaction.

Since imaging devices typically produce raster images, image vectorization remains an important means for generating vector-based content. A recent trend in vector graphics research focuses on developing scalable (resolution-independent) representations of full-color raster images. One long-lasting challenge on this front is to make vectorized images easily editable so that artists and designers can incorporate them into their artwork. Since a full-color raster image typically has significant pixel-level detail and not all of this detail needs to be preserved in the abstracted version, a second challenge is to let users easily choose a desired level of detail for a vectorized image.

We introduce a vector image representation to meet the aforementioned challenges. In our representation, the image plane is decomposed into a set of triangular patches with potentially curved boundaries, and the color signals over the image plane are treated as height

Figure 2.1: A raster image converted to a piecewise smooth vector-based representation with curvilinear features. Guided by the feature curves, a multiresolution vector image pyramid enables intuitive editing of the resulting vector graphics. Left: Original image. Middle left: Subdivision surface control mesh for vectorization. Middle right: Magnified (4x) local view of the vectorized image. Right: Combined effects of shape editing, color editing and stylization on the vector representation.

fields. A subset of the curved patch boundaries are automatically aligned with curvilinear features. The geometry of the patch boundaries as well as the color variations over the patches are represented using a piecewise smooth Loop subdivision scheme. Such a simplicial layout of patches avoids T-junctions and better supports feature-sensitive patch boundary alignment. With properly defined subdivision masks, the patch boundary curves are C^2 everywhere, and the color function is at least C^1 everywhere except across features where it is discontinuous.

To offer the flexibility of multiple levels of abstraction, we also design a multiresolution vector image representation. Different resolutions in this representation contain progressively coarser meshes, each one acting as the control mesh of a piecewise smooth subdivision surface. Because image features play a crucial role in vector image representations, our multiresolution representation is feature-centric. Features are sorted and distributed to different resolutions according to their saliency scores. When switching between different resolutions, we "downsample" or "upsample" features rather than pixels. Multiple resolutions allow the user to choose a desired level of abstraction during image vectorization or vector image editing.

Using the piecewise smooth subdivision representation, we develop techniques to facilitate a variety of vector image editing operations, including shape editing, color editing, image stylization, and vector image processing. Such editing operations effectively create novel

vector graphics by reusing and altering existing vectorized images. While shape editing, color editing and image stylization can be applied to any single resolution, vector image processing involve inherently hierarchical operations that affect multiple levels simultaneously.

We summarize our contributions as follows.

- We introduce a new vector image representation based on piecewise smooth subdivision surfaces. It is the first work that applies subdivision surfaces to modeling image with discontinuity curves. The fact that this representation automatically provides the desired continuity conditions is particularly useful for both vectorization and subsequent vector editing operations. Due to its simplicity and elegance, this representation is a unified and flexible framework that may find many other uses in vector image representations.

- This work supports a novel feature-oriented multiresolution vector image representation. Unlike traditional multiresolution mesh representations for shape editing, the most important motivation of our multiresolution representation is not facilitating vector image editing, but providing multiple levels of visual abstraction. According to their own preferences, users may choose different levels as the final vectorization result.

- This is also the first work that focuses extensively on vector image editing and processing. Our representation lets us process vector images directly, and achieves novel results different from such operations on raster images. Research in this direction is significant because it directly processes vector images without the need to go through any intermediate raster image representations. We expect this work to stimulate further research on processing of vector image representations.

2.2 Background

2.2.1 Raster images VS vector images

Roughly speaking, vector image refers to the type of image representation by geometric primitives, such as polygons, parametric curves or surfaces. The geometric primitives cover the image domain in a *nonuniform* manner. Pixel values are defined *parametrically* in terms of them. In contrary, a raster image is a 2D array of pixels, which are *explicit, uniform* color samples of the image domain. Despite of its simplicity, the majority of existing images are stored as raster images. Why is it so?

A uniform non-parametric representation seems to be the only justifiable scheme to record and display complex visual signals in a 2D space without any assumptions to be made. That is why raw image data is acquired as discrete optical samples in digital cameras, and at display end, images are always rasterized before sent to a screen or printer. Second, in the human visual system, the eyes receive optical signal by nearly randomly scattered rod and cone cells in the retina. The retina image is then de-mosaiced and separated as intensity and color signal for higher level visual processing – so the visual image we take at the very early stage is also in "raster" format. Besides, the brain has developed mechanisms to interpret discrete visual signals rather delicately. One famous example is that Pointillism painters use very rough dots to depict objects of a scene. While through human eyes, such a painting always appears to exhibit more details than it actually does.

Despite its prevalence, raster image lacks internal structures of the content and is limited for editbility. This is a major reason vector image comes into play. The implicit representation of an image by a set of geometric primitives offers much greater flexibility to edit the image content at semantical level. A notable downside of vector images, on the other hand, is the difficulty to represent high frequency signals, e.g., textured regions. Raster image are more suited in this case.

There are other aspects for the comparison of the two image types. One is resolution

dependency. Vector graphics renders crispy edges regardless of resolution while raster image does not. In raster image vectorization, resolution independence requires (1) faithful representation of curvilinear feature by high order continuous curves and (2) discontinuity-preserving representation for sharp color contrast across feature. Another aspect is compactness. Raster image is intrinsically redundant while vector image encodes image features adaptively according to the underlying complexity of the signal. However, raster image compression techniques, e.g., JPEG (discrete cosine transform), JPEG2000 (wavelets), fractal compression [22], etc., have been extensively studied and successfully narrowed the gap. Quantitative comparison of some vector image representations to raster image is found in a few documents [5, 23].

2.2.2 A brief review

Early vector graphics were mainly used in graphical design, e.g., fonts, logos, cartoon animation, diagram plotting, etc. Recent techniques have extended them to photographic images.These systems generally fall into two types of applications: (1) systems that converts existing raster images into vector images – known as image vectorization, and (2) systems that allows artists to create novel vector graphics from scratch. Below we review a few representative works.Sorted by the characteristics of underlying representation, we have *piecewise-constant parametric curves* [2], *Triangulation* based vectorization [3], and *higher order* representations, such as gradient mesh [5], diffusion curves [6], thin-plate splines [7, 24] and subdivision surfaces described (chapter 3).

Piecewise constant parametric curves

One type of vector graphics system were designed for non-photographic images vectorization such as fonts, logos, clip arts, or line drawings [25, 26, 2, 27, 28]. Edge tracing generates a set of closed regions. Each region is assigned a constant color. Boundary of the regions is modeled by parametric curves. For example, the *Potrace* system by Selinger [2] traces binary

images with optimized Bezier curves and line segments (Figure 2.2). Piecewise constant regions are most limited for color representation and can not be used for photographic image reproduction. However, they are found particularly well suited for image abstraction and stylization [3, 29].

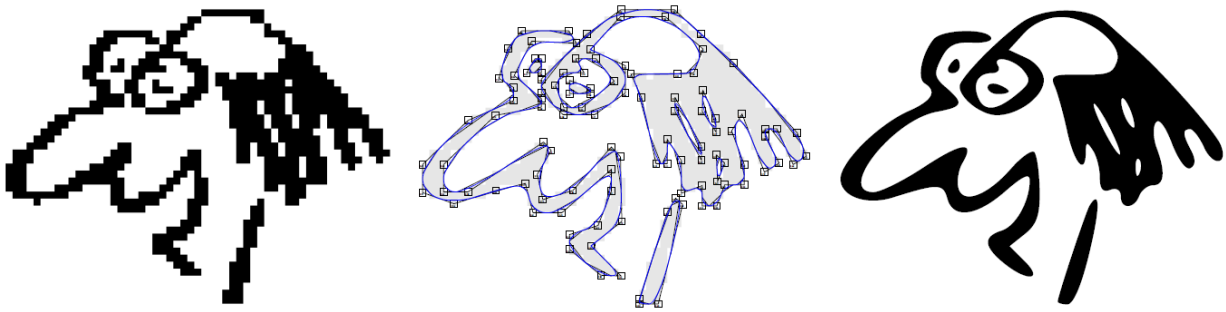


Figure 2.2: Potrace system illustration. Left: input raster image; Middle: edge trace and control vertices optimization; Right: output vector image. Source: Selinger et al. [2].

Triangulation

Several other early image vectorization algorithms were proposed for image compression purposes based on Delaunay trangulation [30, 31, 3]. The *Ardeco* system by Lecot and Levy is one of the first examples. At the core of this method is a region segmentation algorithm by energy minimization with a color fitting term and a competing region smoothness term. The regions are defined as groups of trixels obtained from a saliency-aware image domain trangulation. Image intensity is approximated in each region using a linear combination of zero-, first- or second-order basis functions. The limitation of trangulation-based representation is that each curvilinear feature is approximated by many short line segments. Such polyline feature representation is not genuinely resolution independent because the difference between a smoothly curved feature and the polyline with only C^0 continuity at the vertices will become more obvious when magnified. Figure 2.3 shows the triangulation-based vectorization result with 0, 1 and 2-order color basis. Notice the obvious polyline edge artifacts.

For more faithful photographic image vectorization or photo-realistic vector graphics authoring, higher order surfaces must be used with parametric curves for color and feature

Figure 2.3: Triangulation based vectorization. Upper left: trangulation; Upper right: 0-order color basis (piecewise constant); Bottom left and right: 1- and 2-order color basis. Source: Lecot and Levy [3].

representation.

Gradient mesh

Gradient mesh is a popular vector graphics edit tool available in Adobe Illustrator and Corel CorelDraw. A user can manually create, edit and animate vector arts at photorealistic level with it. A gradient mesh consists of topologically planar quad patches with curved boundaries, any kind of curve can be used. Each vertex in the patch contains location, color and color gradient information (Fig. 2.4). Colors are interpolated inside the patch guided by

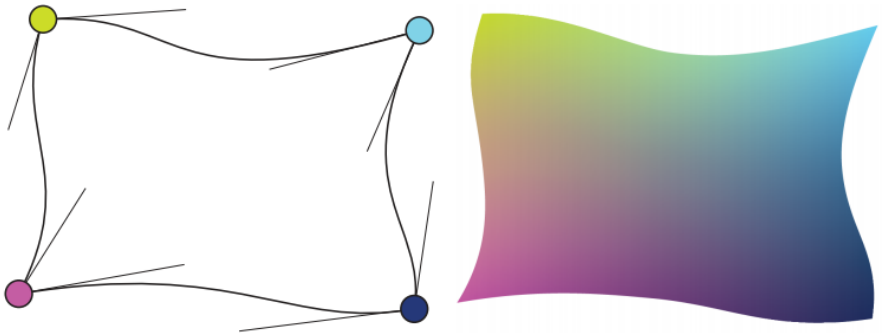


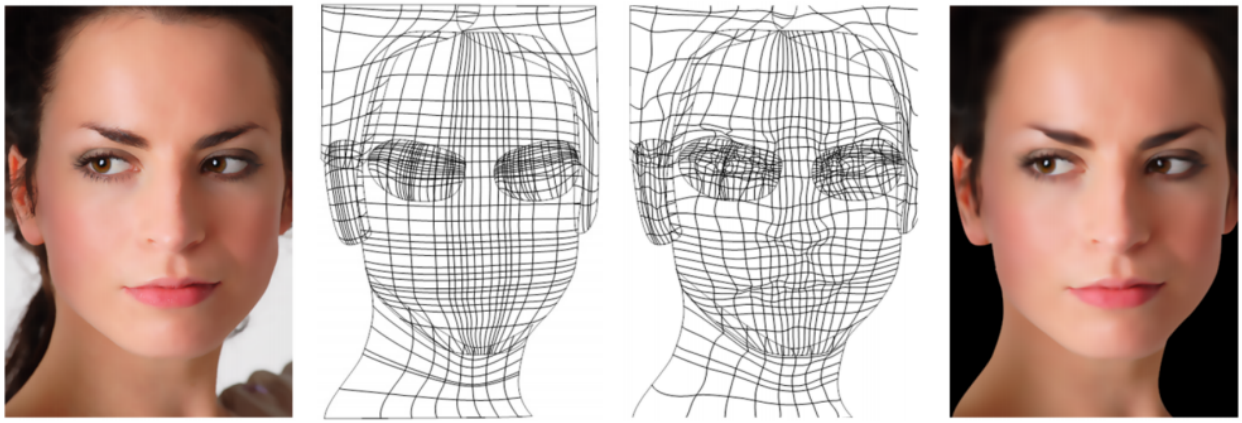
Figure 2.4: Gradient mesh definition. Source: Barla and Bousseau [4]

the color gradients. This representation provides a flexible control over structure of image gradient.

Sun et al. [5] introduce a vectorization technique that optimize vertex color and position of a gradient mesh structure to approximate an input image, where manual mesh initialization is required to align mesh boundaries with salient image features (Figure 2.5). A fully automated and topology-preserving gradient mesh optimization algorithm is proposed by Lai et al. [8]. One limitation of gradient mesh is that it is defined to be smooth everywhere. Color discontinuity for sharp edges is approximated by degenerate quads or fold-overs. In addition, the rectangular arrangement of patches in gradient mesh hinders a highly adaptive spatial layout. The following diffusion curve, TPS and subdivision representations avoid these two problems in a more principled way.

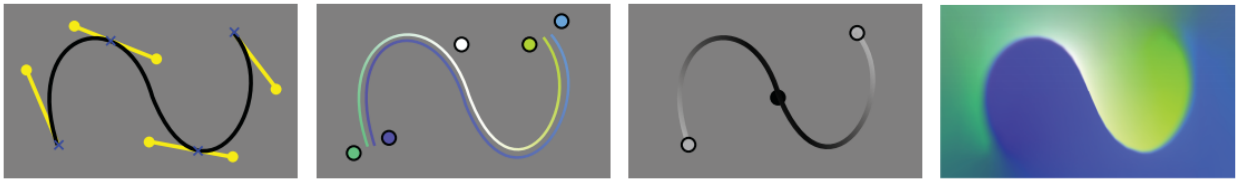
Diffusion curves

Diffusion curves [6] provide a mesh-free representation. It is especially well suited for interactive authoring of vector graphics. Different from gradient meshes, by which users need to manually determine the number of patches and sketch mesh lines along both image features and interior regions of an object, diffusion curve sketches an object by Bezier curves for image feature only. Color values are assign at both side and interpolated along the curve. A curve blur property is set to determine the mix-up (blurriness) of colors across curve loca-



(a) Input bitmap (b) Initial gradient mesh (c) Optimized gradient mesh (d) Resulting image

Figure 2.5: Image vectorization by optimized gradient mesh. Source: Sun et al. [5]



(a) Geometry (b) Color (c) Blur (d) Final image

Figure 2.6: Diffusion curve definition. Source: Orzan et al. [6]

tion. Given curve geometry, color and blur, a heat diffusion process diffuses the source color over the whole image plane.

Write image gradient as $[I_x, I_y]^T$, and second order derivative as $[I_{xx}, I_{xy}; I_{xy}, I_{yy}]$, the diffusion process minimizes sum of squared gradients $\int\int (I_x^2 + I_y^2) dxdy$ by solving the Laplace equation:

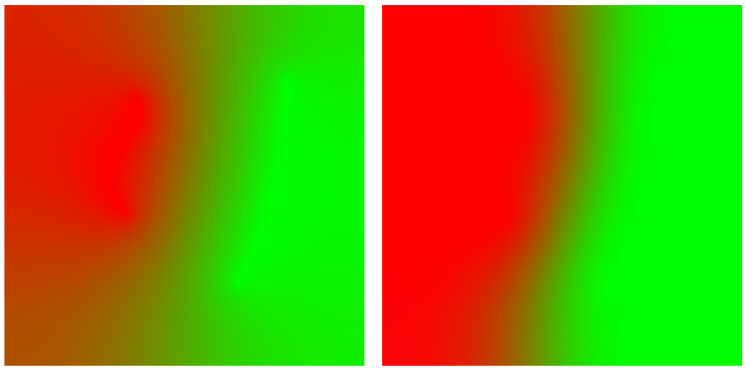
$$\Delta I(x, y) \quad = \quad I_{xx} + I_{yy} = 0$$

with constraints (the source colors), where ΔI is known as the Laplace operator.

Discretizing the equation over the image grid using finite differences results in a sparse linear system that can be solved directly or using iterative solver such as Jacobi relaxation or Gauss-Seidel method. To achieve real-time performance, they adopt the multigrid algorithm [32] that applies Jacobi iteration in a coarse-to-fine scheme.

In follow-up works, Takayama et al. [33] generalize the diffusion curves to diffusion surfaces for volumetric rendering. Jeschke et al. [34] and Hnaidi et al. [35] apply diffusion curves to geometric modeling and terrains and displaced surfaces.

One limitation of diffusion curves is the lack of control over color propagation. Bezerra et al. [36] introduce diffusion constraints that allows users to control the speed and direction of color diffusion. Another problem with diffusion is derivative discontinuity at the curves. Because the solution's Laplacian is nonzero at the curves, their color constraints yield creases, or tent-like artifacts along the diffusion curves (Fig. 2.7). In image vectorization, due to its lack of "stiffness", diffusion may fail to reproduce color over the interior regions, especially when edges fall far apart. The problem is avoided in extension works by [7, 24] using thin plate splines with extra color sample off the curves.



(a) Laplace equation (b) Bi-Laplace equation

Figure 2.7: Smoothness comparison of diffusion (Laplace equation) and thin plate splines (Bi-Laplace equation). Source: Barla and Bousseau [4]

Thin plate splines

The thin plate spline scheme adopted by [7] and [24] minimizes bending energy in terms of second-order derivatives

$$\int\int (I_{xx}^2 + 2I_{xy}^2 + I_{yy}^2) dxdy$$

to provide higher order smoothness. The solution to this energy has a physical analogy of bending a *thin sheet of metal* subject to constraints, while the diffusion process analogies to

deforming a *membrane*. The characteristic difference is demonstrated in Figure 2.7. Notice the smoother color interpolation and smoothness across the constraining curves in the right figure.

By variational calculus, the minimizer of the objective reduces to the Bi-Laplace equation:

$$\Delta^2 I(x,y) = (\frac{\partial^2 I}{\partial x^2})^2 + 2(\frac{\partial^2 I}{\partial xy})^2 + (\frac{\partial^2 I}{\partial y^2})^2 = 0$$

with constraints (source colors). Discretizing the equation gives a large sparse linear system and can be solved by similar techniques for the Laplace equation as mentioned above.

In image vectorization [7] where the target color values are known, TPS is estimated in its functional form. A TPS that interpolates a set of N constraint points $\{(x_i, y_i)\}^N$ has a known functional form expressed as

$$f(x,y) = b_0 + b_1 x + b_2 y + \sum_{i=1}^{N} \alpha_i \phi(||(x_i, y_i) - (x,y)||) \tag{2.1}$$

where ϕ is the thin plate radial basis function $\phi(s) = s^2 log(s)$, and $\sum_{i=1}^{N} \alpha_i = 0$ and $\sum_{i=1}^{N} \alpha_i x_i = \sum_{i=1}^{N} \alpha_i y_i = 0$ in order for $f(x,y)$ to have square integrable second derivatives [37]. Notice that the functional has only $N+3$ parameters, which can be estimated with a small set of sample colors by least squares. Once α and b are solved, any point in the parametric domain can be evaluated using equation 2.1.

To support complex editing power and expressiveness, Finch et al. [24] define a variety of features (crease, contour curves, slope curves and critical points) and compound feature types. In [7], the whole image is decomposed into a set of non-overlapping Bezier patches, thin plate spline color fitting is done over individual patches. As a result, color continuity is not maintained across the patch boundaries. Again, this creates creases similar to those of diffusion curves. Our follow-up work [23] use a global subdivision surface to model image color and guarantee continuity except for sharp features, where tear edges were used to maintain C^{-1} continuity (Figure 2.8). Chapter 3 describes this work in detail.

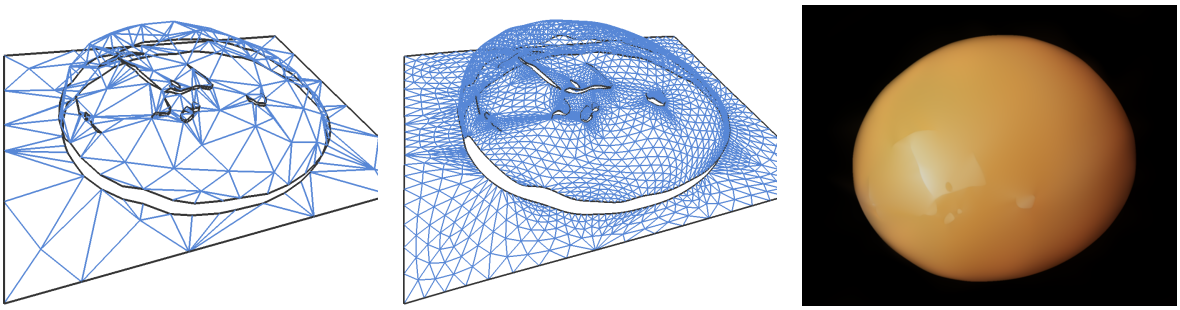


Figure 2.8: Subdivision-based vector image representation.

PDE-Based VS Mesh-Based

Part of the aforementioned representations are based on a control mesh (triangulation, gradient mesh, Bezier patch-based decomposition [7], subdivision surfaces [23]). The rest of them use a mesh-free representation, e.g., diffusion curves [6] rely on curves with color and blur attributes as boundary conditions of a diffusion process. The final solution of this diffusion process, by solving a partial differential equation, defines the color variations of a vector image. The free-form vector graphics defines higher-order color variation in a similar way (thin-plate splines). This technique is particularly well suited for interactive authoring of vector graphics. However, it has a few limitations. First, the diffusion curves of a PDE-based representation are not coupled together by definition, which makes it hard to perform some vector image editing operations like region-based color or shape editing. In comparison, mesh-based approach builds a network of objects and regions to better support vector image editing and signal processing. Second, PDE-based representations focus primarily on discontinuity curves – they only have very low degree of control of color propagation between those sharp discontinuities. In contrast, a mesh-based representation can approximate arbitrary color variations between the curves in a non-parametric manner. From this perspective, the mesh-based representation can be regarded as a semi-vector representation that lies between the traditional definition of a vector image (parametric curves and surfaces) and a raster image (non-parametric uniform sampling).

Vectorization and interactive authoring Mesh-based representations are generally more suited for vectorization and less convenient for vector image authoring. A mesh is

intrinsically an over complete representation for object structure, as non-feature edges need to be added and the topological constraints be maintained. The Ardeco system and the subdivision surface based method fall into this category. The gradient mesh tool by Adobe Illustrator and Corel CorelDraw is an exception. Still, authoring a figure from scratch requires a professional expertise. The artists need to manage several meshes for a single object and draw mesh lines that may not align with object features. Working with triangular meshes is even harder.

The mesh-free feature-based curve representations, such as the diffusion curves [6] and freeform vector graphics tool [24], are more user friendly for interactive authoring. Artists draw an object by sketching its silhouettes and specify their colors. While the diffusion curve provides no control over interior regions, the freeform vector graphics tool allow users to add point features in interior regions for additional color constraints.

Vector graphics authoring tool can also be used to vectorize existing images. For example, the optimized gradient mesh algorithm [5] asks a user to initialize a mesh that is aligned with image features and optimize the mesh position, vertex color and color gradients to generate the final gradient mesh. The diffusion curves is also demonstrated as a vectorization tool. Given an input image, the diffusion curves are formed by edge detection and color sampling. The edge blur values are computed by Gaussian scale space analysis.

Comparison of representations Table 2.1 summarizes the representational characteristics of the 7 methods. The gradient mesh implementation detail by Adobe Illustrator and Corel CorelDraw is not made public. We use the optimized gradient mesh standard by [5] in the table.

2.3 Subdivision-based vector image representation

Below we describe in detail of our subdivision-based image vectorization method.

Subdivision approach We consider color variations in a raster image from a geometric

method	*Edge representation*	*Cross-edge discontinuity*	*Surface smoothness*
Potrace	Bezier	Yes	piecewise-constant
Ardeco	polyline	No	No
gradient mesh	Bezier	Inherently Not	Yes
diffusion curves	Bezier	Yes	No
TPS [Xia'09]	Bezier	Yes	No
TPS [Finch'11]	quadratic B-spline	Yes	Yes
subdivision surfaces	subdivision curve	Yes	Yes

Table 2.1: Comparison of several vector graphics representations.

perspective, treating each color channel as a height field over the 2D image domain. Thus, an image with three color channels is associated with a 2D surface in 5D space. Because an image has color discontinuities (i.e. features), we adopt a piecewise approximation. The image domain is partitioned into regions, each defining a locally smooth surface patch. Specifically, we define the complete piecewise smooth surface (spanning the full image domain) by adapting a piecewise smooth subdivision scheme [19, 38] as follows.

The subdivision scheme of Loop [19] defines a smooth (C^1) surface as the limit of a subdivision scheme applied to a control mesh $M = M^0$. The subdivision step $M^r \to M^{r+1}$ refines the mesh M^r by (1) replacing each triangle by four triangles and (2) computing vertex positions of M^{r+1} as affine combinations of nearby vertices in M^r, according to a set of subdivision *masks*. Each vertex in M^{r+1} is either a *vertex point* or *edge point*, depending on whether it corresponds to a vertex or edge in M^r, and the associated subdivision masks are shown in Figure 2.9(a,e).

In our setting, the control mesh is a 2D triangulation of the image domain, in which each vertex is a 5-dimensional vector (x, y, r, g, b). The effect of subdivision is to smooth both the 2D geometric positions and the 3D color coordinates. After subdivision, each triangle in the control mesh M^0 becomes a triangular region, generally with curved boundaries, and the color function is at least C^1 across all such boundaries.

The scheme of [38] extends subdivision to allow surface creases and corners, where the surface is continuous but not smooth. This is achieved by tagging control mesh edges as

either *smooth* or *crease*[1]. However, for our purposes this is insufficient because the resulting surface is still everywhere continuous.

Discontinuous subdivision To model discontinuous functions, we further extend subdivision by introducing a third type of edge, a *tear*, which has the effect of splitting each adjacent vertex into two vertices (Figure 2.9(d,g)). These two vertices share the same x, y spatial coordinates, so that the triangulation maintains a bijection onto the image domain. However, the two vertices may have different r, g, b color coordinates, so as to break color continuity.

A chain of tear edges is called a *tear feature*, and a chain of crease edges is called a *crease feature*. We consider only tear features, because their associated discontinuities form the most prominent elements in vector graphics images. Vectorizing crease features, which are more subtle, is left as future work.

In our scheme, vertices have four types: *smooth*, *crease*, *tear*, and *corner*. A smooth vertex is a vertex incident only to smooth edges; crease and tear vertices are adjacent to exactly two crease and tear edges respectively; corner vertices are located at all other configurations, including feature endpoints. To fix the rectangular image boundary, the four corners are marked as corner vertices, and all perimeter edges are marked as crease edges.

Figure 2.9 shows the complete set of subdivision masks. The corner vertex mask ensures its position does not move after subdivision. The crease and tear masks both subdivide the feature curve to produce a cubic B-spline curve. The tear masks differ in that they act independently on the duplicated vertices across the tear.

In practice we apply two or three subdivision steps and then push the subdivided vertices to their limit positions (using a set of limit masks, not shown). Although the mesh could be further subdivided, we find that it already starts to form a sufficiently accurate piecewise linear approximation.

The goal of vectorization (Section 2.4) is to (1) optimize the vertex positions along this feature to align the resulting subdivided feature curve with the raster image discontinuities,

[1]We use the terminology "crease" rather than "sharp" to make clearer our further generalization.

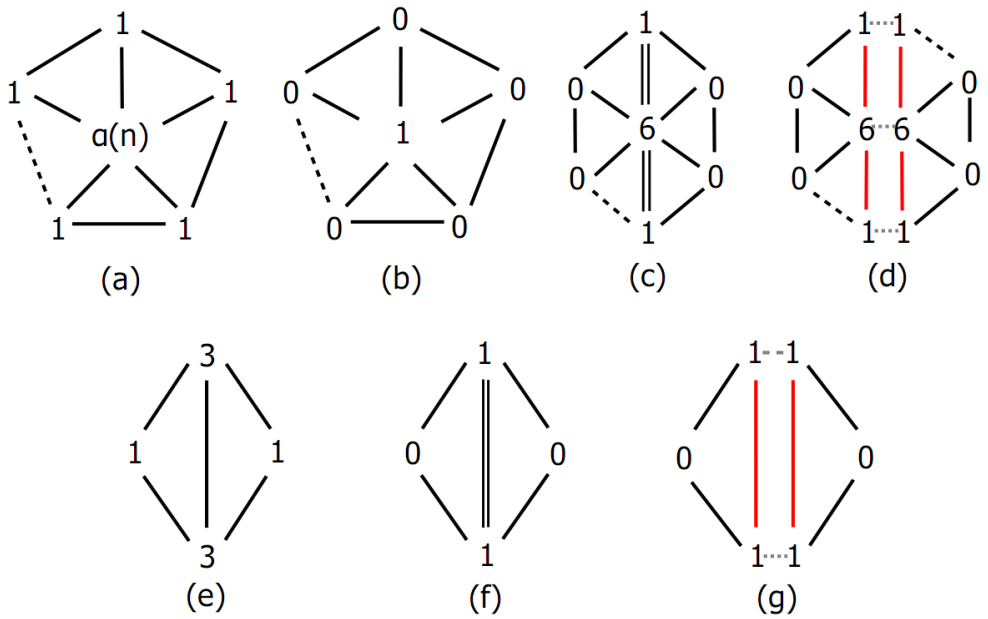


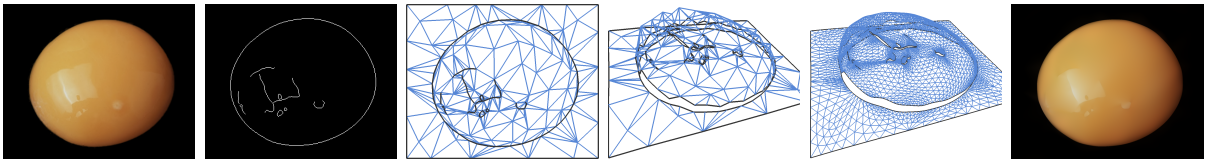
Figure 2.9: Subdivision masks. (a-d) Subdivision vertex masks for smooth, corner, crease and tear vertices. (e-g) Subdivision edge masks for smooth, crease and tear edges. In (d) and (g) the parallel vertex-pairs each connected by a gray dashed line are "split" vertices along the tear feature. In (a), $\alpha(n) = (\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n})^2 + \frac{3}{8}$ where n is the vertex valence.

and (2) optimize the vertex colors such that the piecewise smooth subdivided mesh best fits the image color function.

In this vector graphics setting, the piecewise smooth subdivision approach offers a number of benefits. First, it represents both the shapes of image features and the variations of color signals in a unified, resolution-independent representation. Second, it achieves the desired spatial and color continuity conditions by construction, without requiring constraints over the degrees of freedom, namely: (1) the subdivided feature curves are everywhere C^2, and (2) the color function is everywhere C^1, except across feature curves where it is C^{-1} (Actually, it is also C^2 away from extraordinary vertices, which are those with valence other than 6.). Because the vector image will be subject to interactive user manipulation, these properties guarantee that no matter how the user deforms the control meshes, feature curves will remain geometrically smooth, and the color field will remain smooth everywhere except across features. In comparison, the piecewise color representation in [7] may give rise to undesired visible seams across region boundaries.

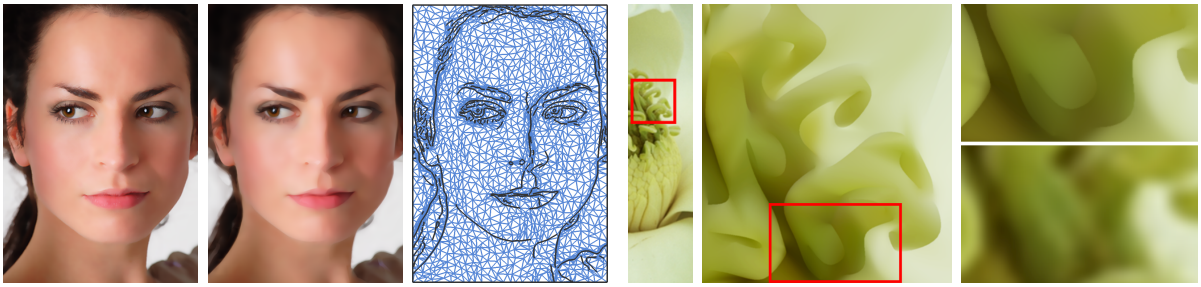
Multiresolution representation While our vector image representation involves a sequence of progressively finer meshes, these meshes all share the same set of features — the same amount of detail. In Section 2.5, we extend this with a multiresolution structure, in which each resolution level is itself a vector image, and contains a different level of detail.

2.4 Single-level image vectorization



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

Figure 2.10: Vectorization pipeline. (a) Original image. (b) Detected curvilinear features. (c) Control mesh of the reconstructed subdivision surface. (d) 3D view of the optimized control mesh. (e) Optimized control mesh subdivided twice. (f) Rasterization result of the reconstructed vector image (1.40/pixel mean reconstruction error using the control mesh of 303 (0.3%) vertices and 369 triangles).



(a) input (b) reconstruct (c) ctrl mesh (d) input (e) magnify (8x) (f) comparison

Figure 2.11: Two vectorization examples. The left example uses a control mesh of 1725 (1%) vertices and 2470 triangles with mean reconstruction error 1.48; the right example shows a magnified view (8x) of a local region of a flower pin using our vector representation and a comparison to the same scale magnification of the raster image using bicubic interpolation.

Our image vectorization pipeline consists of four major stages: feature detection, initial control mesh construction, mesh simplification, and color optimization.

Feature detection is performed through Canny edge detection and image segmentation. Detected Canny edges are thinned to 1-pixel wide, and broken pieces are linked together to form longer features [39]. If image segmentation (we use GrabCut [40] in our experiments) is performed to partition an image into regions, region boundaries are always closed and are also treated as features. The initial control mesh is created as a regular grid, with one vertex per image pixel. Additional vertices are introduced in the mesh at subpixel locations based on the detected features, and the mesh is locally retriangulated appropriately. All edges along features are marked as tear edges. Initial mesh construction and subsequent feature-preserving mesh simplification follow the work of [7] except that we use subdivided feature curves to fit image features. Note that more advanced feature detection method, such as the one in [41], could be adopted to achieve better subpixel accuracy without affecting our overall vectorization pipeline. We would like to leave this as a topic for future investigation.

Mesh simplification Because the mesh is initially very dense, for efficiency we perform simplification using the quadric error metric of [42], treating the color channels as geometric height fields. To preserve the topology of feature tears, each tear vertex is only permitted to collapse with an adjacent vertex on the same tear. And to carefully preserve the geometric fidelity of the feature curves, after each collapse involving a tear vertex we solve an optimization to locally refit the subdivided feature curve to its associated feature in the raster image. This geometric optimization is formulated to minimize the summed squared distances between vertices of the densely subdivided mesh and their target positions:

$$E = \sum_{j=1}^{N_s} \|\mathbf{x}_j - \mathbf{V}\mathbf{y}_j\|^2, \tag{2.2}$$

where $\mathbf{V}$ is a $2\text{x}N_c$ matrix of the N_c unknown control vertices, and N_s is the number of affected vertices in the subdivided mesh. Vector $\mathbf{x}_j$ is the target position of the j-th subdivided vertex, and $\mathbf{V}\mathbf{y}_j$ is the expression for the limit position of the j-th subdivided vertex in terms of the control vertices. The target position $\mathbf{x}_j$ for a tear vertex is its projection onto the original feature curve; for all remaining vertices it is their current position.

Minimizing E is a sparse linear least-squares problem since the local nature of subdivision

rules ensures that each $\mathbf{y}_j$ is a sparse vector. If the maximum fitting error along the new curve exceeds one pixel, the edge collapse is rolled back. Also, we prevent foldovers by disallowing edge collapses that result in flipped triangles. Once the number of vertices in the control mesh has been reduced to a predefined threshold, mesh simplification terminates and the structure of the control mesh becomes final.

Color optimization Because the mesh simplification process is greedy and heuristic, the solution is far from optimal. In fact, the colors in the simplified mesh do not take into account subdivision at all. The final step globally optimizes the colors of the control mesh vertices. We use a formulation similar to (2.2), but this time over 3D colors rather than 2D positions. Thus $\mathbf{V}$ becomes a $3\text{x}N_c$ matrix containing all control vertex colors, and N_s is the total number of vertices in the subdivided mesh. The target color $\mathbf{x}_j$ is the bilinearly filtered image color at the 2D location of the corresponding subdivided vertex.

Because color values may vary significantly across image features, missampling near features in the original raster image can result in disturbing results. Thanks to the one-pixel error bound in the earlier feature fitting, we need only pay special attention to vertices in

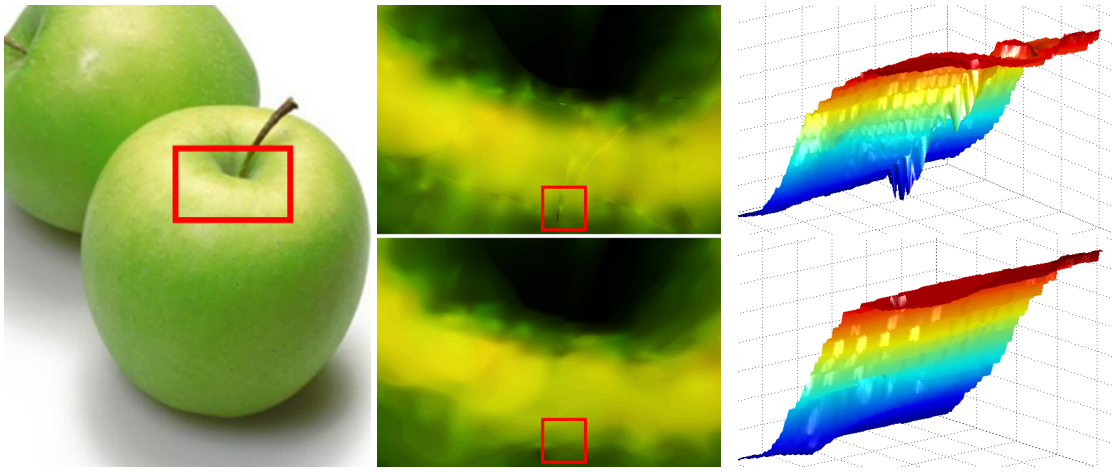


Figure 2.12: Cross-boundary continuity: comparison with [7]. Left: Original image. Middle: Contrast-enhanced view of the vectorization of the local rectangular region by [7] (upper) and our method(bottom). Right: 3D reconstructed surface (gray-scale as height) of the indicated local regions from the images in the middle. Note the color and geometric gradient discontinuities across patch boundaries from the result by [7] in the upper middle and upper right.

Figure 2.13: Representation compactness compared to [8]. Left: Original image. Right: Vectorization result with mean error 2.13 using our method. Our subdivision-based representation takes up 14.0KB of storage after zip compression; the gradient mesh representation ([8]) needs 9.4KB storage with the same mean error, while JPEG compression with a comparable quality requires 20KB.

a one-pixel band adjacent to the features. We obtain the target colors of these vertices as follows. For tear vertices themselves, the target color is assigned from the closest pixel on the feature. The remaining vertices that lie within one pixel from the features are referred to as *border vertices*. Their target colors are initially set to be undefined, and we perform hole filling to propagate correctly sampled colors from nearby interior vertices and tear vertices. Hole filling starts from the boundary of the holes and iteratively extends into the interior of the holes. The target color value of a vertex, whose color is previously undefined, is interpolated from the target values of its neighboring known vertices.

We solve the resulting large sparse linear system using TAUCS [43].

2.4.1 Results and comparisons

Examples of vectorization and magnification can be found in Figures 2.10 and 2.11. To demonstrate the quality and compactness of our vector image representation, we have compared our method with those in [8, 7]. As shown in Figure 2.12, our result is at least C^1 across non-feature patch boundaries whereas the result by [7] exhibits color discontinuities across such boundaries. Figure 2.13 indicates that the amount of storage required by our

method is comparable to gradient meshes.

2.5 Multiresolution vector images

There are no universal criteria regarding the optimal density of features in a vectorized image. Denser features make the vectorized version more faithfully represent the original raster image while sparser features provide a higher level of abstraction, which could be more visually appealing. We introduce a feature-oriented multiresolution vector image representation to address this problem. Such a representation contains different levels of details at different resolutions, and thus provides vector-based approximations of a raster image over a spectrum of granularity and abstraction. It has the flexibility that users can choose their preferred level of abstraction in a vector image.

Our multiresolution vector images are based on features as basic building blocks due to their importance in vector representation. Thus each resolution represents a distinct level of abstraction of the original raster image (Figure 2.14). The multiresolution vector images are constructed as follows.

We first gather the set of features in the original image. Each feature f is assigned a *saliency score* that is a weighted summation of its length $P(f)$ and the average contrast (gradient magnitude) $C(f)$ across the feature:

$$S(f) = P(f) + w\, C(f), \tag{2.3}$$

The user-configurable parameter w determines the relative importance of length and contrast. In addition, we allow users to interactively adjust the saliency of semantically important features. by interactively overriding their assigned resolution.

We uniformly group features into L subsets in descending order of saliency. This lets us define a sequence of *nested feature sets* $\{F_i\}_{i=0}^{L}$, where $F_j \subset F_i$ if $j > i$. In our multiresolution representation, we generate a single-level vector representation S_i for each feature

set $F_0 \dots F_{L-1}$ such that S_i has C^1 continuity everywhere except for the subset of region boundaries aligned with F_i across which it has C^{-1} continuity.

Thus, we begin at level 0 with the finest control mesh, which contains all features. Level l is constructed from level $l-1$ by first removing the subset of features $F_{l-1} \setminus F_l$. Recall that every control vertex along a feature is paired with another vertex on the opposite side of the feature, and these vertices have the same x- and y-coordinates but different color coordinates. When a feature is eliminated, the open boundary it creates becomes sealed, and every pair of vertices on the boundary is merged into a single vertex with an averaged color. Second, mesh simplification is performed to eliminate a certain percentage of the vertices. In the current implementation, we remove 50% of the vertices between two consecutive levels by default. During this stage of simplification, each merged vertex on a just-eliminated feature is allowed to collapse with any other vertex, while a vertex on a remaining features is constrained to collapse only with other vertices on the same feature.

Note that w and L are heuristic parameters. In our experiments we always use default values, $w = 2$ and $L = 3$. However, users can choose to assign them alternative values through an interface.

2.6 Vector image editing

Editability is the main reason that vector graphics is widely used in content design. Traditional vector graphics is represented with high-level geometric primitives with adjustable parameters so that editing operations can be conveniently achieved. In this section, we demonstrate that our new vector representation for photographic images also exhibits such an advantage and supports a variety of editing operations. Note that even though most editing operations addressed here can already be performed on raster images, our goal is to perform direct vector image processing without going through any intermediate raster images.

2.6.1 Shape editing

Shape editing of an image object is achieved by deforming a part of the control mesh corresponding to the image object at an appropriate level of the multiresolution vector images. We use the as-rigid-as-possible shape manipulation technique in [44] to solve for a new configuration of the control vertices given user-supplied deformation constraints. A deformation constraint is a pair of original and new control vertex positions. Given one or more deformation constraints, the technique in [44] is able to solve for new positions of the remaining control vertices by minimizing the overall mesh distortion.

We have implemented a simple shape editing interface. Users can provide deformation constraints by dragging a single vertex or a feature. When a feature is selected, the user can partially deform the feature or completely relocate the entire feature. In the former case, the mouse click position is the center of deformation and the closest feature is selected. The displacement of any vertex on the selected feature is based on its initial distance to the center of deformation using a Gaussian kernel. Users can specify the variance (σ^2) of the Gaussian kernel to adjust the region of influence. In the latter case, the user can translate and/or rotate a selected feature to define the new positions of the vertices on the feature.

Figure 2.15 shows large-scale shape editing of an object silhouette to convincingly alter the perception of its 3D shape. Figure 2.17 shows shape editing (and color editing, introduced in the next subsection) of multiple features in the same vector image to create a new facial expression. Note that such intuitive feature-oriented shape editing cannot be conveniently achieved with previous vector representations for photographic images. Diffusion curves [6] and individual gradient meshes in [5] are not coupled together by definition. Modern shape editing techniques, such as the one in [44] cannot be easily applied without significant enhancements to such representations. The automatic technique in [8] performs adaptive grid subdivision near image features but does not exactly align vertices with features, making high-precision feature selection and relocation hard to achieve. Although there is a base mesh holding all the Bézier patches together in [7], patch boundaries are individual Bézier curves. During shape deformation, the continuity between adjacent curve segments cannot

be guaranteed without enforcing additional constraints among their control vertices.

2.6.2 Color editing

With the mesh representation and explicit feature structures, color editing can be conveniently performed by defining region selection tools and then manipulating the color channels of the selected control vertices. Similar to the vertex and feature selection tool in shape editing, we support selecting a single vertex or an entire feature. Users can further specify a propagation radius to select a local region around the selected vertex or feature.

For color manipulation, users specify the rgb values of a new color that will affect the color of the selected vertex or feature. There are of course many transformation operators that one can apply. In our prototype we have explored two such operators, *BLEND* and *TRANSFORM*. In the BLEND mode, the final color is computed as a linear blend of the original and new colors. In the TRANSFORM mode, a seed vertex closest to the mouse click location is first chosen and a 3x3 diagonal color transform matrix is computed using the new color and the original color of the seed vertex. This transform matrix is then applied to all vertices within the selected local region. Both color editing modes preserve the original color variations in the selected region. Figure 2.16 shows color editing results achieved with BLEND and TRANSFORM operators.

2.6.3 Abstraction and stylization

Our multiresolution vector images provide a sequence of control meshes with progressive density. These control meshes generate subdivision surfaces that approximate the original raster image at different levels of details. Finer levels more faithfully represent the original raster image while coarser levels provide a higher level of abstraction with the removal of edges with low salience. This structure gives a natural solution to edge-aware multilevel image abstraction, which allows users to choose an appropriate abstraction level to display an image for various purposes.

We further generate stylized images from multiresolution vector images by drawing freestyle strokes along a subset of features (Figure 2.18). Stylization requires a user-selected abstraction level and interactively selected regions of interest where features are going to be emphasized with strokes. In comparison with [29], where the input image is segmented into regions, each of which is filled with a constant color, our results put more emphasis on sharp image features, which are aligned with partial region boundaries, and preserve weakened color variations within local regions. Both methods show visually interesting results, but with different stylization emphases. In our results, strokes are only used to enhance features within regions of interest. Within a region of interest, features with saliency scores higher than a threshold are always enhanced with strokes while features with saliency scores below the threshold are randomly chosen to be enhanced. The width of a stroke varies according to the length of the feature. Both ends of a stroke are linearly tapered.

Abstraction and stylization represent another novel application of our vector image representation. There have been no previous attempts to use vector image representations for such a purpose. Feature alignment and preservation as well as the removal of high-frequency details in our vector-based approximations are consistent with the goal of abstraction and stylization. Our results demonstrate that abstraction and stylization based on vectorization can be quite effective. Our technique also suggests a way to make smoother but edge-preserving base images for other methods that rely on a base- and detail-layer decomposition.

2.6.4 Signal processing

It is desired to perform image processing tasks directly on a vector-based image representation, which eliminates the need to convert vector images back to raster images. Different levels of a multiresolution vector image, as introduced in Section 2.5, are mutually independent. Such a multilevel structure becomes inadequate for vector image processing tasks, such as filtering and enhancement, which need to work with all frequency bands simultaneously. We further enhance our multiresolution vector image representation by storing inter-level details. The resulting data structure is called a vector image pyramid.

Our representation for inter-level detail in the pyramid shares similarities with the multiresolution mesh hierarchy proposed in [45], which was in turn inspired by the Burt-Adelson image pyramid [46]. (Interestingly, a vector image representation combines characteristics of both meshes and images.) The idea is that during the simplification of the original control mesh using a sequence of elementary coarsening operations (i.e. edge collapses), we record for each operation a detail vector that expresses the position (or data) of the removed vertex relative to the resulting coarse neighborhood. Specifically, the removed vertex is predicted as a weighted combination (relaxation) of the coarse neighboring vertices, and the detail vector is the difference from this prediction. Interested readers are referred to [45, 46] for more details.

Some differences between our vector image pyramid and the multiresolution mesh hierarchy in [45] are summarized as follows.

• **Relaxation** The relaxation operation R used to predict a vertex from its one-ring neighborhood has weights from [47]:

$$R(\mathbf{v}) = \sum_{i=1}^{n} w_i \mathbf{v}_i, \quad w_i \propto 1/\|\mathbf{v}' - \mathbf{v}'_i\| \text{ and } \sum_{i=1}^{n} w_i = 1, \tag{2.4}$$

where $\mathbf{v}'$ is the projection of $\mathbf{v}$ in the XY plane, which provides a perfect parameterization of our 2.5D color signal. The Fujiwara weights usually produce higher-quality results in our experiments than the second-order divided differences in [45].

• **Local Frames and Detail Vectors** We store 2D position displacement vectors with respect to local frames in the simplified meshes. For color displacements, we simply use per-channel differences with respect to the global frame whose z-axis is perpendicular to the image plane.

The detail vectors and scalars in the pyramid construction process store the differences between actual signals and their smoothly predicted version from the relaxation operation. Within a vector image pyramid, detail signals at finer levels accommodate relatively high-frequency details while those at coarser levels accommodate low-frequency details. As in [45],

signal processing operations such as low-pass, high-pass, and band-pass filtering can be performed conveniently by appropriately editing such detail signals. Filtering and enhancement based on editing detail signals can be formulated as:

$$\mathbf{v}' = R(\mathbf{v}) + \eta d(\mathbf{v}), \tag{2.5}$$

where the edited vertex $\mathbf{v}'$ is obtained from its relaxed prediction $\mathbf{v}$ and by scaling the precomputed detail signal in 2D geometric coordinates and/or 3D color coordinates. Smoothing is achieved by setting $0 < \eta < 1$, and enhancement is achieved by setting $\eta > 1$. Setting η as a function of pyramid level achieves filtering effects dependent on frequency bands.

Figure 2.20 and Figure 2.21 show two signal processing examples. Figure 2.19(d) shows a combined effect of filtering and stylization. These results demonstrate that standard signal processing operations can be directly performed on a vector image without the need to convert it to a raster image first.

Note that signal processing operations have not been supported in previous vector image representations. Unlike our multiresolution vector representation, they were not originally designed for signal processing tasks. Comparing to raster image processing, our cut-open mesh structure along sharp image features leads to perfect edge-preserving smoothing without the need of any extra treatment while the bilateral filter or other edge-preserving raster image filtering algorithms only partially preserve contrast across sharp edges.

2.7 Discussion

GPU-based rasterization We rely on GPU-based rasterization of subdivision surfaces to achieve real-time vector image display. Recent work on real-time surface subdivision can be found in [48, 49]. In our experiment, we implemented rasterization using CUDA [50] on nVidia Geforce GTX275. For a display window with a moderate size (512x512) our GPU-based rasterization yields 60 frames per second. We do uniform subdivision on the control mesh and terminate when the total number of triangles exceeds the number of pixels in the

display window. Note that a zoomed view only requires a portion of the control mesh to be subdivided. Thus the rendering speed is determined by the display window size rather than the image size.

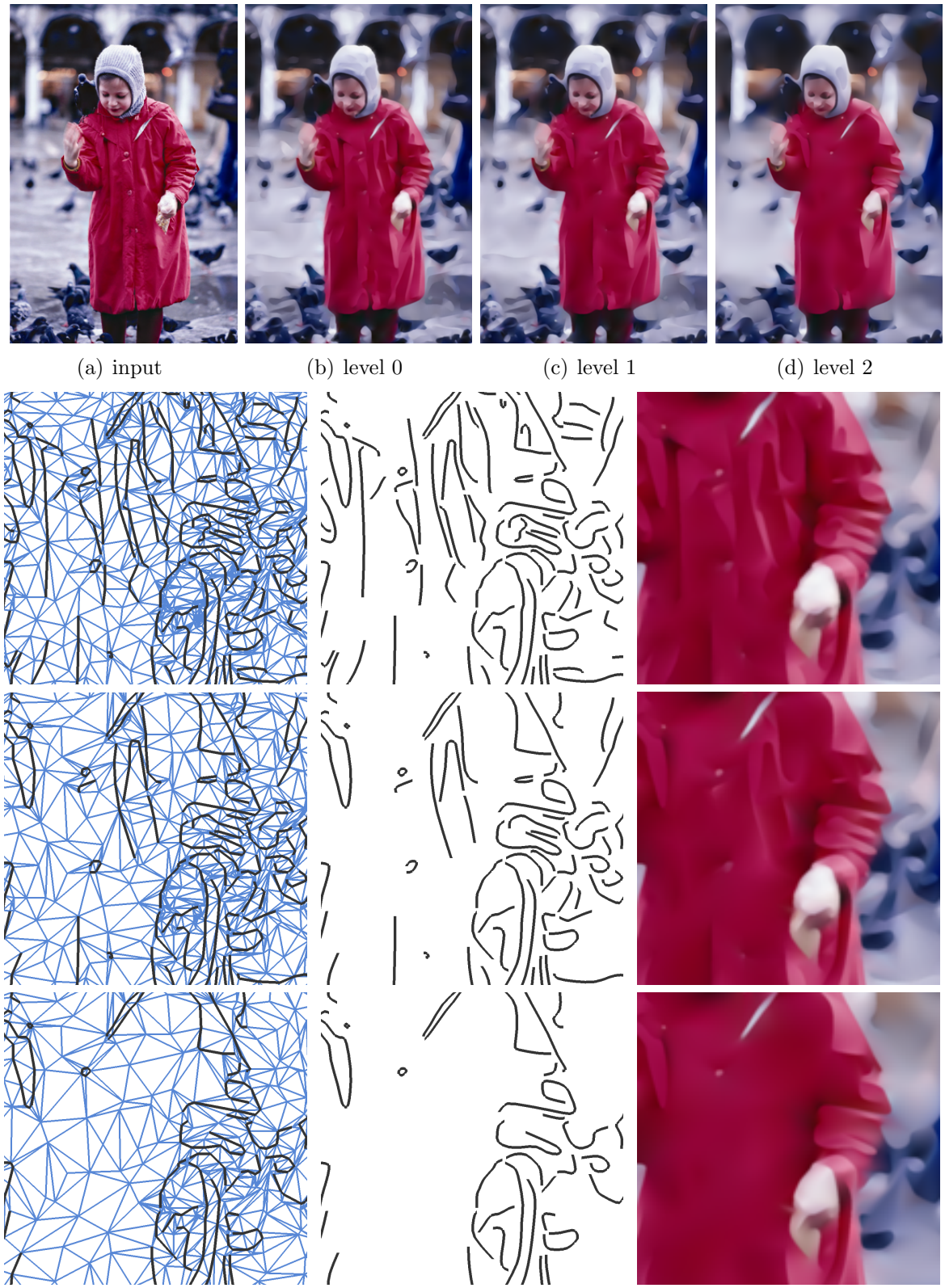
A triangle with its ordered one-ring neighborhood is the atomic unit in our parallel implementation. Multiple iterations of subdivision are performed on the initial control mesh. Each iteration subdivides each of the triangles from the previous iteration into four smaller triangles each associated with an ordered one-ring neighborhood itself. To avoid heavy data swapping between the CPU and the GPU, we allocate sufficient global memory on the GPU at the beginning and manage the memory layout to make only one data swap during the whole subdivision process. Shared memory is utilized to achieve high speed data access. The per-block shared memory size is the major hurdle to achieving a high level of parallelization. In our experimental configuration, 32 threads are created and executed simultaneously with synchronization per block and a total of 240 blocks are allocated.

Vector representation statistics Table 2.2 summarizes the control mesh complexity of the vector images used except for the ones that have been mentioned in the context.

Table 2.2: Complexity of vector image control meshes: ratio of vertices relative to original unsimplified mesh; number of vertices, triangles, and features; optimization time in seconds. Statistics for images *girl*, *apple*, *horse*, *grapes* are at the finest level. The number of vertices is halved at each level from the preceding finer level.

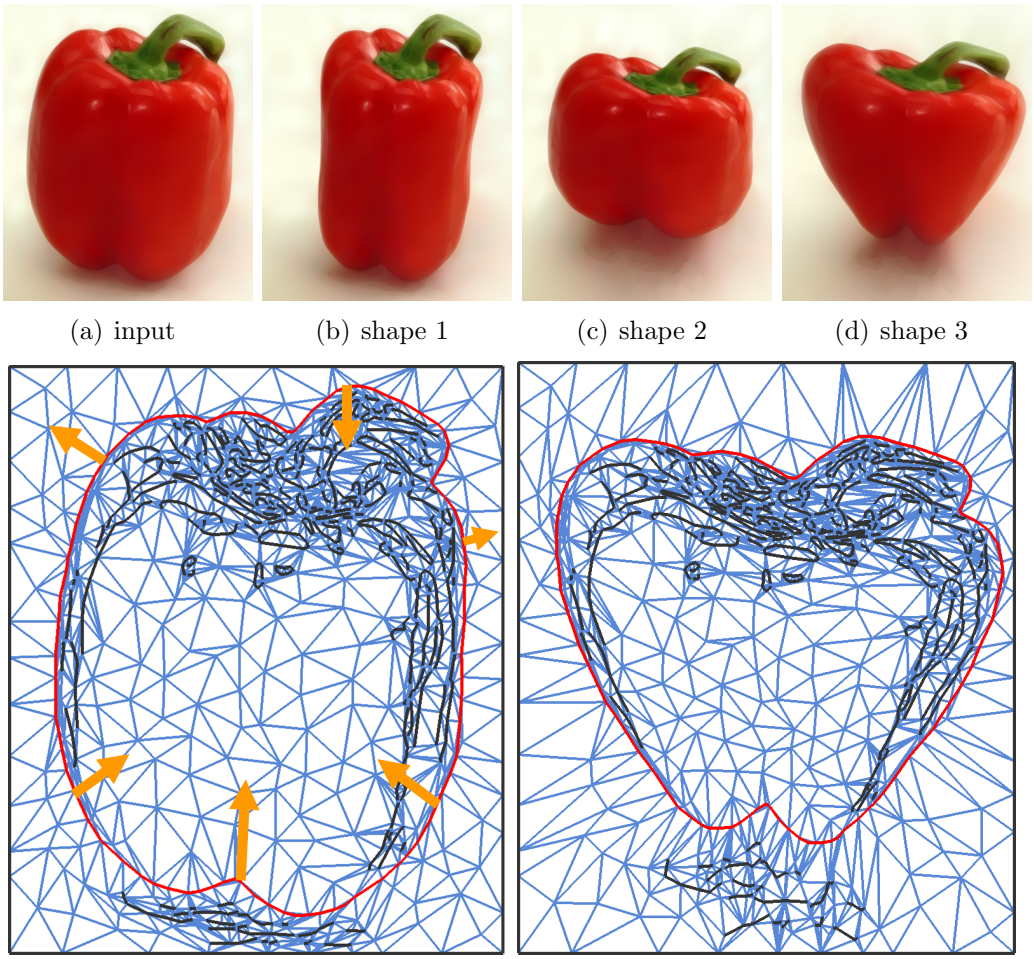
Image (Fig. No.)	Ratio	Vert	Tri	Feat	Opt time
flowers (1)	0.015	3000	3612	174	3.0
flower pin (4e)	0.01	207	239	17	0.37
peppers (6)	0.0125	2294	3731	48	9.8
girl (7)	0.05	6453	9356	306	5.9
pepper2 (8)	0.015	1960	2518	182	6.3
banana (9a)	0.01	883	1258	20	3.1
tulip (9c)	0.03	3313	5342	63	3.0
flower2 (10b)	0.0125	426	570	15	4.3
face (10 bot.)	0.05	3462	5998	54	2.9
Italy (11b)	0.04	5741	7807	303	5.1
goldfish (11d)	0.025	6766	8753	594	6.1
grapes (12)	0.08	9210	15325	153	8.2
apple (13))	0.08	11308	21267	24	11.7
horse (14)	0.08	12966	22366	113	13.1

Limitations There exist a few aspects about our algorithm and implementation that deserve further investigation. Our vector image representation currently only supports a single foreground layer. While this assumption does not negatively impact most of the operations, it does affect shape editing. Separating different objects in an image onto distinct layers enables a user to alter the shape of each object independently. Issues related to multiple layers include how to automatically or semi-automatically recognize layers in an image and how to fill gaps created when two overlapping layers are altered differently. Another limitation is that our current implementation does not support the insertion of new features into a vectorized image. We expect this can be accomplished in a straightforward way by first performing intersection tests between the new features and the triangles in the control mesh followed by re-triangulation around the intersections.



(a) input (b) level 0 (c) level 1 (d) level 2

Figure 2.14: Multiresolution abstraction. Top (a-d): Original raster image, the finest, intermediate, and coarsest levels of abstraction. Bottom (left to right): Cropped views of the control mesh, subdivided features, and vectorized image in three levels of abstraction;

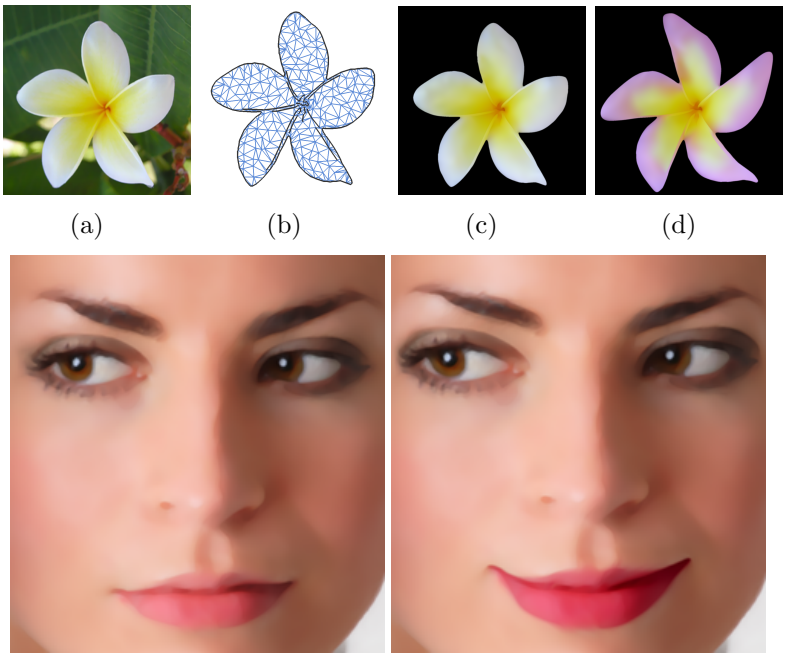


(a) input (b) shape 1 (c) shape 2 (d) shape 3

Figure 2.15: Shape editing. Top row: Three shape editing results on a given vector image. Bottom row: Original control mesh for (a) and its deformation for (d) using six indicated mouse interactions.

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

Figure 2.16: Color editing. (a) Input vector image. (b) Color editing in the BLEND mode. (c) Input vector image 2. (d)-(e) Color editing in the TRANSFORM mode.



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

Figure 2.17: Combined shape and color editing. Upper: (a) raster image; (b) control mesh of the extracted foreground layer; (c) foreground object vectorization; (d) shape and color editing to the object. Bottom left: Vector image input. Bottom right: Shape and color editing on the vectorized image. Shape editing includes deforming the mouth and eye brows, and enlarging the eyes. Color editing is performed on the lips.

(a) raster image (b) vector image stylization (c) raster image (d) vector image stylization

Figure 2.18: Vector image stylization examples.

(a) raster image (b) vectorization of (a) (c) detail enhancement (d) filtering and stylization (e) difference map: (c) - (b)

Figure 2.19: Combined stylization and vector image processing results.

Figure 2.20: Signal processing using our vector image representation. Upper left: Raster image. Bottom left: Vector approximation. Upper middle: Low-pass filtered vector approximation. Bottom middle: High-frequency enhanced vector approximation. Upper right: Difference map of the smoothed image and vector image. Bottom right: Difference map of the enhanced image and the vector image.

Figure 2.21: Left: Original raster image. Middle: Vector image detail enhancement. Right: Difference map due to vector enhancement.

CHAPTER 3

IMAGE RELIGHTING

3.1 Introduction

A important task of image composition is to take an existing *image fragment* and insert it into another scene. This task is appealing because 3D models are difficult to build, and image fragments carry real texture and material effects that achieve realism in a data-driven manner.

Relighting is generally necessary in the process. To relight the object, we need to know its shape and material properties. Image-based composition methods [46, 51, 52], on the other hand, avoid the relighting process, totally relying on the artist's discretion for determining shading-compatible image fragments and limiting the range of data that can be used for a particular scene. Despite such limitations, image-based methods have been largely preferred to relighting-based methods, because shape estimation (for the latter) remains an extremely challenging problem. State-of-the-art algorithms, such as the SIRFS method of Barron et al. [1] still produce weak shapes and do not work well on complex materials. Is there a compromise between these two spaces that allows for improved image editing?

We propose such an approach by exploring an *approximate shading model*. The model circumvents the formidable 3D reconstruction problem, yet is reshadable and allows a much wider range of objects to be inserted into a target scene.

There are good reasons to consider approximate shading models in image relighting. Evidence shows that human visual system (HVS) can tolerate certain degrees of shading inaccuracy [53, 54, 55]. Psychologists describe this phenomenon with the term *alternative*

physics [20], explaining that the brain employees a set of rules that are not strict physics when interpreting a scene from an image. When these rules are violated, a perception alarm is fired, or recognition is negatively effected[**?**]. Otherwise, visual plausibility is maintained without having to adhere strictly to physical correctness. Our model exploits the inherent ambiguity of HVS from this line of reasoning; methods bearing the same spirit have been found in material editing [56] and illumination estimation [57].

Our approximate shading model works as follows: shading is decomposed into a smooth component captured by a coarse shape h, and a linear combination of two naturally distinct shading detail layers S_p and S_g:

$$S(h, S_p, S_g) = shade(h, L) + w_p S_p + w_g S_g \tag{3.1}$$

where L is illumination and w_p and w_g are scalar weights (illustration in Fig. 3.1).

The coarse shape is purely based on the contour constraint (e.g. surface normals at the silhouette are perpendicular to the viewing direction), easy to construct and robust to moderate perturbation of view direction. It produces smooth shading capturing directional and long-scale illumination effects that are critical for perceptual consistency. The two "detail" layers (S_p and S_g) encode the middle and high frequencies of the shading signal left out by the smooth shading component and account for visual complexity of the object. While image-based composition of the detail layers is not physically-based, in practice it yields surprisingly good results for a variety of object and material types. With this model, we implement an image relighting system that supports object insertion with little user input. Figure 3.1 shows our model and pipeline.

To evaluate the model, we compare our relighting results with a state-of-the-art shape reconstruction method [1] in two tasks: (a) re-rendering MSE on the MIT intrinsic image dataset, and (b) a user study in which we ask subjects on Mechanical Turk to rate the relative realism of results by both methods as well as against real scenes. In the first task, our method yields slightly lower MSE on the MIT Lab illumination dataset. The user study is more compelling as qualitative realism is our primary focus when inserting objects into

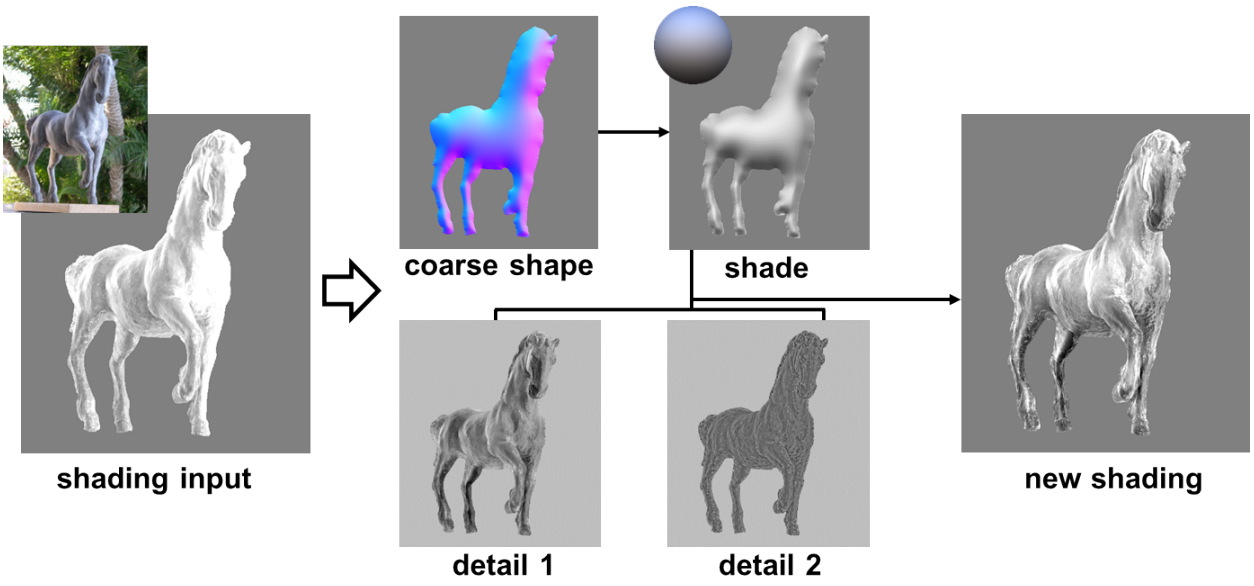


Figure 3.1: Given a shading image estimated from a single image of an object (left), our approximate shading model (middle) can reshade the object under new illumination and produce a new shading on the right. Shape/detail images are rescaled for visualization.

images, and we found that subjects preferred our insertions over that over Barron and Malik by a margin of 20%.

3.2 Background

3.2.1 Image formation and the inverse problem

The rendering equation describes the equilibrium radiance leaving a point as the sum of self emission and reflected radiance integrated over a hemisphere:

$$L(x,y) = L_e + \int_\Omega f_r(\omega_i,\omega_o)L_i(\omega_i)cos\theta_i d\omega_i$$

For non-emitting objects, $L_e = 0$. ω_i and ω_o are incident irradiance and outgoing radiance direction. $L_i(\omega_i)$ is the irradiance from the incident direction. θ_i take account the foreshortening effect of irradiance with respect to surface normal. L_e is a ambient light. $f_r(\omega_i,\omega_o)$

is the *bidirectional reflectance distribution function* (BRDF) that defines how incident light from ω_i is reflected in outgoing direction ω_o. Each direction variable is parameterized by azimuth angle and zenith angle with respect to the tangent plane, so BRDF as a whole is 4-dimensional. To account for spatial variation, the reflectance model is generalized to a 6-dimensional function. To model subsurface scattering effects, the model is further generalized to a 8-dimensional function.

The rendering equation is the fundamental model that governs image formation. The inverse of the rendering equation, is to infer object material and geometry information from images. Once we have these information, we can insert the object into a new environment and relight it. The inverse problem is well known to be ill-posed. The BRDF function is a major source of intractability. While many solutions are proposed to handle complex BRDFs [58, 59, 60, 61, 62, 63], the Lambertian reflectance is a commonly taken assumption for reflectance model in the inverse problem.

Lambertian reflectance is the simplest BRDF model. It assumes a surface has uniformly distributed microfacet structure that reflects light evenly in all outgoing directions. With the Lambertian assumption, the rendering equation (omitting the emitting term and dropping the indices) reduces to:

$$\begin{aligned} I &= A \int_{\Omega} L_i(\omega_i) cos\theta_i d\omega_i \\ \text{or} \quad I &= AS(Z, L) \end{aligned} \tag{3.2}$$

where A is the lambertian reflectance term, or *albedo*, the integral is the shading term S that is parameterized by Z, the surface geometry, and L, the illumination.

Equation 3.2 decomposes an image into two separable terms: albedo and shading. Separating these two terms is a classical image decomposition problem (section 3.2.2). Given the shading image and with a simplified illumination model (point light source or a low dimensional spherical harmonics representation), we can further infer the depth information, known as shape from shading (section 3.2.3).

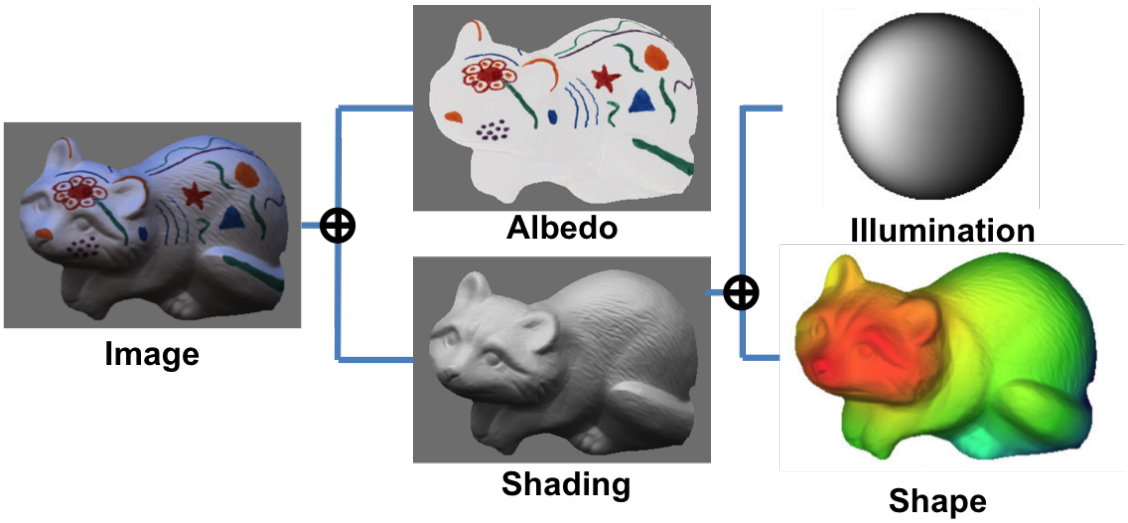


Figure 3.2: Image decomposition example: an input image is decomposed into albedo and shading. Shape and illumination is further inferred from the shading image. The illumination is visualized by its shading on a sphere.

3.2.2 Albedo-Shading decomposition

With the Lambertian reflectance model we have shown that an image is formed by an albedo term times a shading term. Barrow and Tenenbaum [64] proposed the more general idea of decomposing an image into a set of *intrinsic images*, e.g., depth, reflectance, shading, shadow, etc., that "explain away" the image. Albedo-shading decomposition is a classical problem in this line of research.

Land's influential Retinex model [65] assumes effective albedo displays sharp, localized changes (which result in large image gradients), and that shading has small gradients; important variants include [66, 67, 68]. While many variants and new algorithms have been tried, the simplest version of the Retinex algorithm is still among the most robust ones. It consists of three steps: (1) compute gradient (mostly done in log image space); (2) gradient thresholding (small gradients are shading edges, large gradients are albedo edges); and (3) recovering albedo and shading images from their gradient images. Thresholding is tricky and

may vary from dataset to dataset. Step 3 can be formulated as solving a Poisson equation:

$$\Delta f = div\ \nabla \mathbf{f}$$

where f is the unknown albedo or shading, the right hand side is the divergence of the gradient field.

The Retinex assumption does not hold for sharp shadow boundaries or shading changes at normal discontinuities. Funt et al. [69] use chromaticity to help eliminate the ambiguity based on the observation that chromaticity channel change across albedo edges but not for shading edges. Weiss [70] use multiple images under different lighting conditions to eliminate the ambiguity, as albedo edges stay consistent under illumination change while shading edges does not. Tappen et al. [71] train a classifier that discriminate albedo and shading edges using color and gray-scale features. Classification labels is propagated from high confidence region to ambiguous region by Generalized Belief Propagation. Bousseau et al. [72] incorporate user input (indication of regions of constant albedo or shading) to guide the process. The problem is formulated as an optimization based on the assumption that reflectance values are low-rank in local windows and solved in closed form. Shen et al. [73, 74] use non-local texture cues and the sparsity prior of albedo for intrinsic image decomposition.

Another way the Retinex assumption does not hold is in surface geometric details, which causes abrupt sharp shading changes too. Recent work by [75] propose a new intrinsic image decomposition formula that separates high frequency shading signal caused by geometry detail from coarse shading. At the core of the algorithm is a non-parametric filter based on dictionary learning and reconstruction. The shading detail image is demonstrated as a good material presentation for image editing and shows better performance in material classification than the commonly used detail image by Bilateral filtering.

3.2.3 Shape from shading and X

One common assumption in shape from shading is local shading model with point light source on a Lambertian surface, based on which equation 3.2 reduces to:

$$I(x, y) = A(x, y)S \cdot N(x, y) \tag{3.3}$$

where S is the (single) point light source whose magnitude is the light intensity and N is surface normal. Another shading model is to assuming light is isotropic and distant from object. Then we can describe the lighting as a non-negative function on the surface of a sphere. A low dimensional spherical harmonics, an analogy of Fourier analysis but on the surface of the sphere, can be used to represent the environment light sphere. Shading with the spherical harmonics light is simply a quadratic polynomial of the surface normal:

$$S(x, y) = N(x, y)^T M N(x, y) \tag{3.4}$$

where M is a 4×4 symmetric matrix for the 9-coefficient spherical harmonics [76].

Since the first shape from shading technique was introduced by Horn in the early 70's [77], many different approaches have been proposed. Ikeuchi, Brooks and Horn [78, 79] model the problem as an energy minimization problem with a brightness constraint that the reconstructed shape should produce the same brightness as the input image, and a smoothness terms that regularizes the under-constrained system (3 unknowns for normal estimation but one constraint at each pixel). Frankot and Chellappa [80] add integrability constraint in order to recover integrable surfaces. Leclerc and Bobick [81] solve directly for depth, which avoids the integrability issue. Besides, for direct depth recovery, the system is well-posed, so the smoothness term is only used for driving convergence. All of the above mentioned methods deal with local illumination model. Nayar et al. [58] take into account the inter-reflection in illumination using photometric stereo, so does [82]. Wu et al. [83] introduce a interactive interface called *rotation palette* that allows a user to correct long scale surface normals, as SFS typically produces faithful reconstruction for high frequency component but

fail to recover the long-scale depth correctly due to error accumulation during integration. Other user-guided SFS methods include [84, 85]. A comprehensive literature survey can be found in [86] and [87].

Beside of shape from shading, there is a series of shape from X algorithms, where X could be texture, shadow, specularity, stereo, motion, contour, etc. Shape from contour interpolates an as-smooth-as-possible surface assuming an orthogonal camera and surface normal being perpendicular to the view direction [88, 89]. In our relighting project, we use a variant of the shape from contour shape estimate to capture coarse-scale shading in relighting.

Joint optimization Barron and Malik [1] recently propose a system that infers depth, albedo and illumination in a joint optimization process:

$$\begin{aligned} \text{minimize} \quad & g(A) + f(Z) + h(L) \\ \text{subject to} \quad & I = A + S(Z, L) \end{aligned}$$

Here I, A and S are in the log space. Z is the depth. $g(A), f(Z)$ and $h(L)$ are albedo, depth and illumination priors learned from data. Spherical harmonics illumination is used for shading. The optimization is done in a coarse-to-fine framework using L-BFGS.

It is the first unified solution of reflectance, shading and illumination estimation from a single image. The advantage of the joint optimization solution is the capability of utilizing more prior knowledge and the flexibility in explaining image signal with all of the three factors together, though it runs at higher risk of getting none of the components right while together they explain the image (when the priors do not fit or optimization stuck with a poor initialization). The system is trained with the MIT intrinsic images dataset [90] and produces significantly better results on a synthetic dataset generated with the same local shading model by the training. For the original MIT Lab illumination images it still suffers from strong shadows.

3.2.4 Object insertion and relighting

Once we have albedo and depth estimates of an object, we can insert it into a new scene. The scene could be a real 3D graphics scene or reconstructed from images. Karsch et al. [57] introduces a technique that reconstructs a 3D scene from a single indoor image and allows synthetic objects to be inserted and animated. Our algorithm (chapter 4) build an object model from a single image for relighting. We can utilize the technique from [57] to support image to image object insertion. Khan et al. [56] introduces a straightforward method that inserts an object from image to a target image and simulate changes in the apparent material of the object, given an approximate normal field and environment map. Lalonde et al. [91] and Chen et al. [92] introduce pure image-based object insertion systems. Shading consistency is dealt with by a data driven approach, searching in a large database for compatible source. The Poisson image composition method [51] uses gradient domain information to blend source and target image. Maintaining shading consistency is at the artist's discretion. We argue that a relighting procedure would significantly expand the range of images to composite with.

3.2.5 Approximate relighting

The major difficulty of object relighting from images is that the albedo and shape models are not easy to get, even for satisfactory approximates. Material estimation methods only demonstrated reasonable performance on lab environment images, many are under restricted conditions (known geometry [93], multiple illuminations [70, 60], etc). Methods that recover shape from shading are unstable as well as inaccurate, particularly in the absence of reliable albedo and illumination. So, physically-accurate object relighting from a single image should be put as a long term goal, if worth pursuing at all.

However, there are fairly encouraging evidence suggesting an approximate relighting approach might be able to achieve visual realism while circumventing the formidable inverse

problem.

Visual perception facts First, human visual system is tolerant to certain inconsistencies in an image. While how exactly does our brain interpret visual signal is still a puzzle, neuroscientists and psychologists have found physical correctness is not required for visual realism. An "alternative physics", as suggested by Cavanagh [20], may be what is employed by the brain. For example, Conway and Livingstone [55] show human are tolerant to multiple points of view in a single image. Ostrovsky et al. [53] find it is hard to spot inconsistent shadow directions in a single image. as long as gross shading is correct. Highlights are important material cues for humans [94], but observers are not perturbed if the highlight is somewhat in the wrong place (see [95], experiment 3). These facts allows us to generate approximate relighting that somewhat deviates from the true physics model as long as it does not corrupt our visual perception. The image-based material editing work by [56] is a successful example of this.

Second, perceptual studies show that physically-correct image, or the *neutral image*, is not the most effective one for visual impression: caricatured face is more effective for recognition [54]; mice that are rewarded for discriminating a rectangle from a square will respond more vigorously to a rectangle that is longer and skinnier than the prototype; graphics designers often adjust local contrast or shift image hue to render a clearer theme, even though the final result is known to be incorrect (e.g., [96, 97]). The effect, known as the peak shift phenomenon [98], is a universal law of perception. This suggest that relighting system not only need not to pursue photorealism, but also should incorporate extra editability for compositor to adjust (or shift) the neutral result towards an exaggerated effect, e.g., increasing contrast or enhance details.

Illumination cone The illumination cone theory states that the set of images of an object lit by all lighting in a fixed pose lies in a convex cone [12]. Basri and Jacobs [13] show that the illumination cone lies close to a low dimension space (e.g., the top 9 dimensional spherical harmonics representation accounts for 98% of shading variability). This suggests

(a) normal (b) 3D shape (c) view2 (d) view3

Figure 3.3: Normal field and shape reconstruction. Our reconstructions are simple but typically robust to large errors that may manifest in state-of-the-art SfS algorithms (see Fig. 3.11). This benefit is key to our goal of image fragment insertion.

a new shading image can be approximated rather well with a linear combination of a few basis images – a low dimensional image-based reshading method is feasible. Georghiades et al. [99] describe a standard method to estimate a low dimensional representation of this cone to model appearance variation of human face.

Our image relighting system (chapter 4) can be seen as a hybrid of a shape-based and an illumination cone representation. The shape component captures the crucial coarse-scale shading effects for visual perception. The image components account for the variable detail effects under illumination changes and support interactive user composition for enhanced visual effects.

3.3 Our approximate model

Our object model has four components. We compute a coarse 3D shape estimate, then compute three maps: the albedo, a parametric shading residual, and a geometric detail layer. We refer to the "coarse shading" by the shape, the "parametric shading residual" and the "geometric detail" as the three shading components.

3.3.1 Coarse shape

We assume the object to be inserted is an image fragment, and wish to estimate what its appearance under new illumination. Exact shape is ideal but unavailable. We need a representation capable of handling extreme shading effects. For example, a vertical cylinder with light from the left will be light on left, dark on right. Moving the light to the right will cause it to become dark on left, light on right. We also want our reconstruction to be consistent with a generic view assumption. This implies that (a) the outline should not shift too much if the view shifts, and (b) there should not be large bumps in the shape that are concealed by the view direction (Fig. 3.11 demonstrates these kinds of mistakes typically generated by more complicated SfS methods). To support these, we use a simple shape from contour method with stable outline and smooth surface (Fig. 3.3).

First, we create a normal field by constraining normals on the object boundary to be perpendicular to the view direction, and interpolating them from the boundary to the interior region, similar to Johnston's Lumo technique [88]. Let N be the normal field, S be the object mask, Ω and $\partial\Omega$ be the set of pixels in the mask and on boundary, respectively, and $N_\perp^i$ be the tangent of mask boundary at pixel i. We compute N by the following optimization:

$$\begin{aligned} \min_N \quad & \sum_\Omega ||\nabla \mathbf{N}||^2 + \alpha(||\mathbf{N}|| - 1)^2 \\ \text{subject to} \quad & N_z^i = 0 \text{ and } N^i \cdot N_\perp^i = 0, \quad \forall i \in \partial\Omega \end{aligned} \tag{3.5}$$

We then reconstruct a height field from the normal. Reconstructing an exact shape with vertical boundary is tricky; Wu et al. [83] (section 3.1, Fig. 6) describes a method for it. Instead, we reconstruct an approximate height field h by minimizing:

$$\sum_\Omega ||(\frac{\partial h}{\partial x} - \frac{N_x}{max(\epsilon, N_z)})||^2 + ||(\frac{\partial h}{\partial y} - \frac{N_y}{max(\epsilon, N_z)})||^2 \tag{3.6}$$

subject to $h_i = 0$ for boundary pixels (stable outline). The reconstructed height field is flipped to make a symmetric full 3D shape (Fig. 3.3). The threshold $\epsilon = 0.1$ avoids numerical issues near the boundary and forces the reconstructed object to have a crease along its

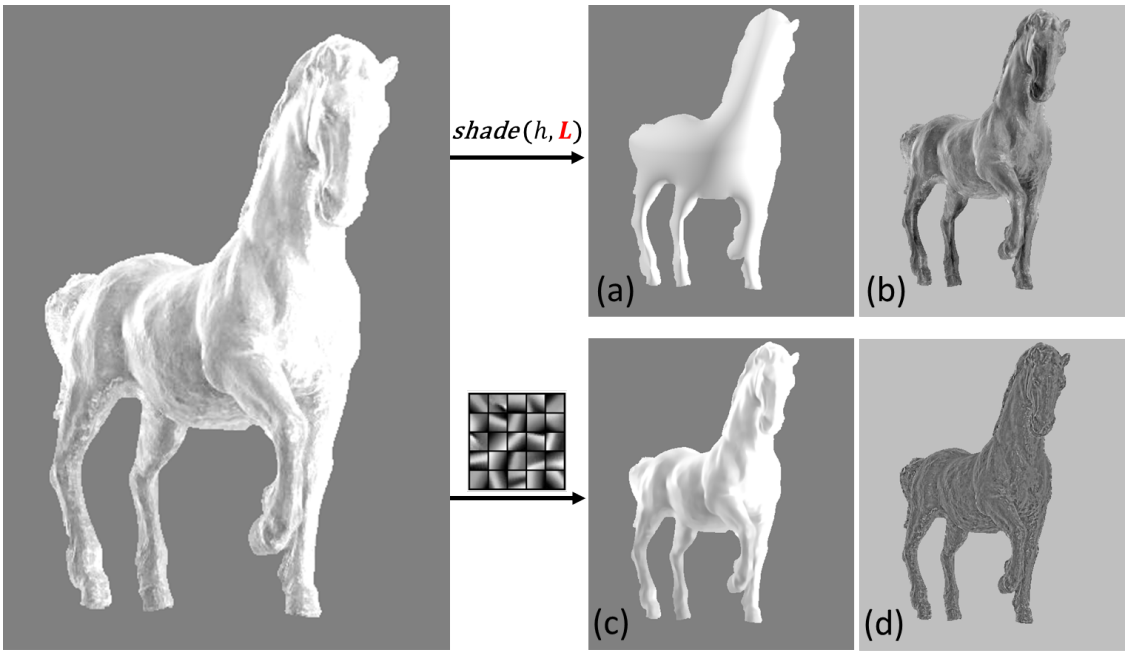


Figure 3.4: From a shading image (left), the upper right row shows the parametric fitting procedure to compute the best fit shading (a) from the shape and the parametric shading residual (b); the bottom right row shows the non-parametric patch-based filtering procedure to compute the filtered shading image (c) and the residual known as geometric detail (d).

boundary. This crease is very useful for the support of generic view direction, as it allows slight change of view direction without exposing the back of the object and causing self-occlusion.

3.3.2 Albedo and Parametric shading residual

The coarse shape can recover gross changes in shading caused by lighting. However, it cannot represent finer detail. We use shading detail maps to represent this detail. We define the shading detail maps as a representation of the residual incurred by shading the coarse shape with some model. We use two shading details in our model: *parametric shading residual* that encodes object level features (silhouettes, crease and folds, etc.), and *geometric detail* that encodes short scale effects.

First, we use a standard color Retinex algorithm [90] to get an initial albedo ρ and shading

S estimates. decomposition from the input image: $I = \rho \cdot S$. We then use a parametric illumination model $L(\theta)$ to shade the estimated shape model and compute the shading residual by solving:

$$\hat{\theta} : \underset{\theta}{\operatorname{argmin}} \sum ||S - \text{Shade}(h, L(\theta))||^2 \tag{3.7}$$

The optimized illumination $\hat{\theta}$ is substituted to obtain the parametric shading residual:

$$S_p = S - \text{Shade}(h, L(\hat{\theta})). \tag{3.8}$$

Many parametric illuminations are possible (i.e., spherical harmonics). We used a mixture of 5 point sources, the parameters being the position and intensity of each source, forming a 20-dimensional representation.

Figure 3.4 top shows an example of the best fit coarse shading and the resultant parametric shading detail. Note that the directional shading is effectively removed, leaving shading cues of object level features.

3.3.3 Geometric detail

The parametric shading residual is computed by a *global* shape and illumination parameterization, and contains all the shading details missed by the shape. Now we wish a compute another layer that contains only fine-scale details. We borrow a technique from Liao et al [75], in which they extract very fine-scale geometric details with a *local* patch-based non-parametric filter. The resultant geometric detail represents high frequency shading signal caused by local surface geometry like bumps and grooves and is insensitive to gross shading and higher-level object features such as silhouettes (see the difference to the parametric shading residual in Fig. 3.4).

The filtering procedure uses a set of shading patches learned from smooth shading images to reconstruct an input shading image. Because geometric detail signals are poorly encoded by the smooth shading dictionary, they are effectively left out. See Liao et al. [75] for more

details. In the experiment we use dictionary size of 500 with patch size 12×12.

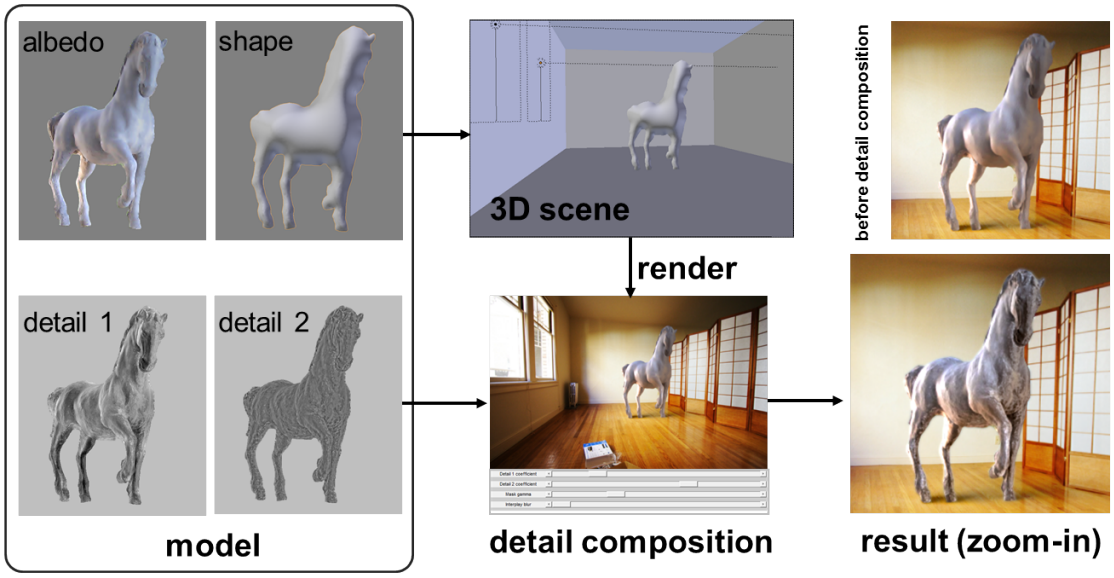


Figure 3.5: Given the object model, an artist place the object into a 3D scene, render it with a physically-based renderer, and then composite it with the detail layers to generate the final result. Notice the difference on the horse before and after the detail composition.

3.4 A Relighting system

With the object model, we develop a system that relights an object from image into a new scene. The system combines interactive scene modeling, physically-based rendering and image-based detail composition (Fig. 3.5).

3.4.1 Modeling and Rendering

We use the technique from Xia et al. [7] to build a sparse mesh object with proper boundary conditions from the height field. The target scene can be existing 3D graphics scenes, or built from an image (Karsch et al. [57], Hedau et al. [100], etc.). The artist then selects an object and places it into the scene, adjusting its scale and orientation, and making sure the view is roughly the same as that of the object in the original image. The model is then

Figure 3.6: Our relighting system adjusts the shading on the object for a variety of scenes with different illumination conditions. Detail composition simulates complex surface geometry and materials properties that is difficult to achieve by physically-based modeling. Best viewed in color at high-resolution.

rendered with the estimated albedo. For all the results, we use Blender (http://blender.org) for modeling and LuxRender (http://luxrender.net) for rendering.

Our shape model assumes an orthographic camera. However, most rendering systems use a perspective camera. This will cause texture distortion. We use a simple "easing" method to avoid it. Besides, the flipped shape model is thin along the base and can cause light leaks and/or skinny lateral shadows. We created a simple user-controllable extrusion procedure to handle such cases; refer to supplemental material for details.

3.4.2 Detail composition

We then composite the rendered scene with the two detail maps and original scene to produce final result. See Fig. 3.6 for examples (more in supplemental material).

First, we composite the two shading detail images with the shading field of the rendered

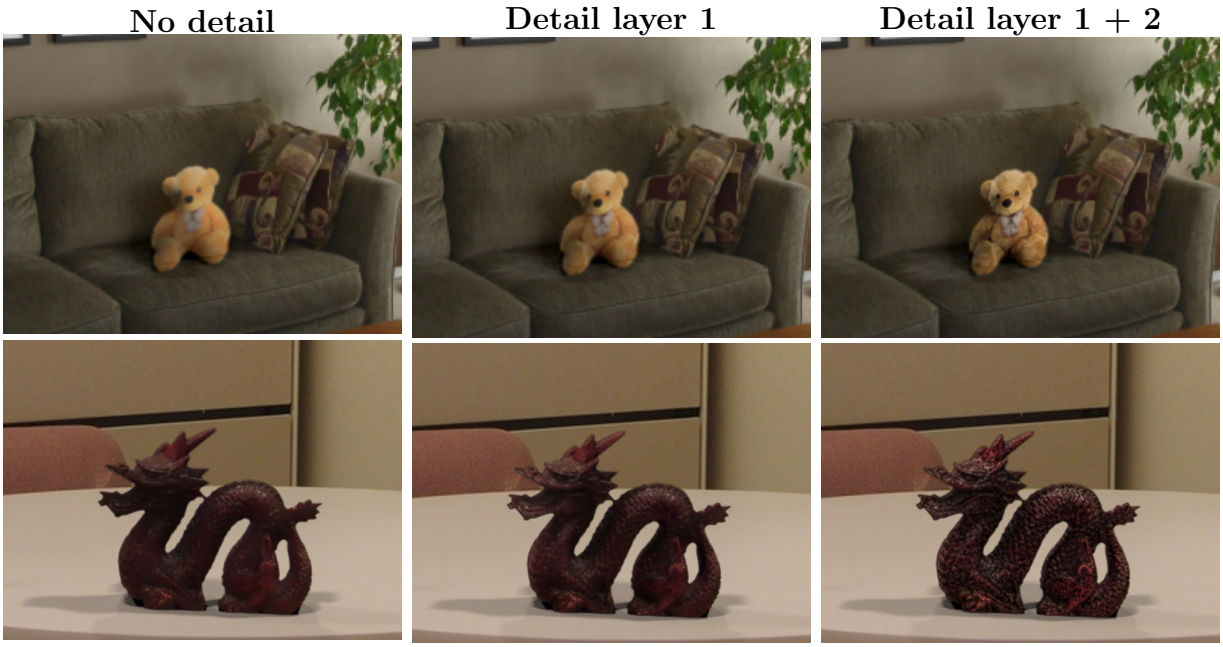


Figure 3.7: Relighting and detail composition. The left column displays relighting results with our coarse shape model and estimated albedo. The middle column displays results compositing with only the parametric shading residual. Notice how this component adds object level shading cues and improves realism of perception. The right column are results compositing with both detail layers. Fine-scale surface detail is further enhanced (see the dragon). Best viewed in color at high-resolution.

image (similar to the material editing technique by Liao et al. [75])

$$C = \rho(S + w_p S_p + w_g S_g) \tag{3.9}$$

where $S = R/\rho$ is the shading field, R is the rendered image, S_p and S_g are the parametric shading residual and geometric detail projected in the new image domain. w_p and w_g are weights adjustable by artist with a slider control for each. Compositing the two details significantly improves the visual realism of object with object level and fine-scale details and adds flexibility to the rendered image (Fig. 3.7).

Second, we use standard techniques (e.g. [101, 57]) to composite C with the original image of the target scene. This produces the final result. Write I for the target image, E for the empty rendered scene *without* the inserted object, and M for the object matte (0 where no

object is present, and $(0, 1]$ otherwise). The final composite image C is obtained by:

$$C_{\text{final}} = M \odot C + (1 - M) \odot (I + R - E). \tag{3.10}$$

3.5 Evaluation

Our assumption is that the shading decomposition model can capture major effects of illumination change of an object. To evaluate this, we compare our representation with state-of-the-art shape reconstructions by Barron and Malik [1] on a re-rendering metric (Sec. 3.5.1). We also conducted a user study to evaluate the realism of our relighting results (Sec. 3.5.2).

3.5.1 Re-rendering Error

The re-rendering metric measures the error of relighting an estimated shape. On a canonical shape representation (a depth field), the metric is defined as

$$\text{IMSE}_{rerender} = \frac{1}{n}||I - k\hat{\rho}\text{ReShade}(\hat{h}, L)||^2 \tag{3.11}$$

where $\hat{\rho}$ and $\hat{h}$ are estimated albedo and shape, I is the re-rendering with the ground truth shape h^* and albedo ρ^*: $I = \rho^*\text{ReShade}(h^*, L)$, n is the number of pixels, k is a scaling factor.

With our model, write $S_c = shade(h, L), S_p, S_g$ for the coarse shading, parametric shading detail and the geometric detail, respectively, and rewrite Equation 3.1 as $\text{ReShade}(S(L), w) = S_c + w_p S_p + w_g S_g$ for some choice of weight vector $w = (1, w_p, w_g)$. The re-rendering metric is:

$$\text{IMSE}'_{rerender} = \frac{1}{n}||I - k\hat{\rho}\text{ReShade}(S(L), w)||^2 \tag{3.12}$$

We offer three methods to select w. An *oracle* could determine the values by least square fitting that leads to best MSE. *Regression* could offer a value based on past experience. We

Method	"Natural" Illumination (No strong shadows)		Lab Illumination (With strong shadows)	
Barron 2012	0.0172		0.0372	
Ours	*LSQ*	*Regression*	*LSQ*	*Regression*
(a) default	0.0329	0.0358	0.0586	0.0641
(b) Barron & Malik S	0.0274	0.0320	0.0341	0.0360
(c) GT S	0.0206	0.0243	0.0228	0.0240
(d) GT S&L	0.0149	0.0219	NA	NA

Table 3.1: Re-rendering error of our method compared to Barron & Malik [1]. Our automatic weights (regression) can generate slightly lower MSE on the real Lab illumination dataset.

learn a simple linear regression model to predict the weights from illumination. Lastly, an artist could *manually* choose the weights, as demonstrated in our relighting system (Sec. 3.4.2).

Experiment We run the evaluation on the augmented MIT Intrinsic image dataset [1]. To generate the target images, we re-render each of the 20 object by 20 randomized monochrome (9×1) SH illuminations, forming a 20×20 image set. We then measure the re-rendering error using our representation and the shape estimation by Barron and Malik. For our method, we compare models built from the Natural Illumination dataset and Lab Illumination dataset separately. The models are built (a) in the default setting, (b) using Barron and Malik's shading and albedo estimation, (c) using the ground truth shading, and (d) using both ground truth shading and illumination. See Table 3.1 for the results. To learn the regression model, for each object we draw 100 nearest neighbors (in terms of Illumination) from the other 380 data points (leave-1-out scheme), and fit a linear model to their LSQ coefficients.

The result shows that when the shape estimation is accurate (on the "Natural" Illumination dataset, a synthetic dataset by the same shading model used in their optimization), our approximate shading performs less as well. This is reasonable, because a perfect shape is supposed to produce zero error in the re-rendering metric. However, when the shape estimation is inaccurate (on the "Lab" Illumination dataset, real images with strong shadows taken in lab environment), our approximate shading model can produce lower error with both regressed weights and oracle's setting. With better detail layers (when ground truth

shading is used to derive them), our model achieves significantly lower errors, indicating space of improvement with a better intrinsic image decomposition algorithm or alternative detail layer definitions.

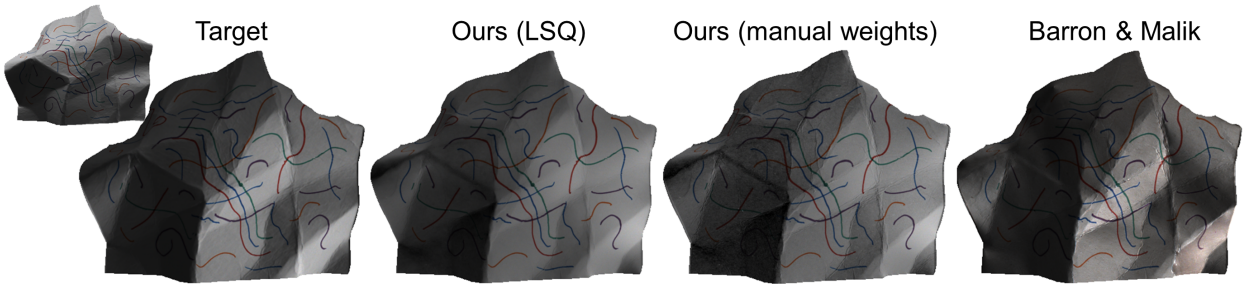


Figure 3.8: Left: target shading (original image in upper left); Middle left: our reshading by LSQ fitting. Middle right: our reshading by user adjusted weight (for the geometric detail); Right: reshading by shape estimation from [1]. Notice the user adjusted weight makes a more realistic result, though not in the measure of least MSE.

It is worthy noting that MSE is not geared toward visual realism (image features takes little weight; non-linearity of visual perception on light intensity, etc.). As a result, the shading images fit by LSQ or regression do not always emphasize the shading details as much as we expect (Fig. 3.8). To demonstrate the real potential of our model in an interactive object insertion setting, we employed a user study to evaluate the "realism" achieved by our insertion technique.

Figure 3.9: Example trial pairs from our user study. The top pair shows an insertion result and a real image (task 1), and the bottom pair shows insertion results from our method and the method of Barron and Malik (task 3). Users were instructed to choose the picture from the pair that looked the most realistic. For each row, which image would you choose? Best viewed in color at high-resolution.

Fraction of times subjects chose an insertion result over a real image in the study

Subpopulation	# of trials	ours	Barron and Malik [1]
all	1040	**0.440±0.015**	0.418±0.016
expert	200	**0.435±0.034**	0.362±0.037
non-expert	840	**0.442±0.017**	0.429±0.018
male	680	**0.447±0.019**	0.426±0.018
female	360	**0.428±0.024**	0.390±0.035
age ($\leq$ 25)	380	**0.432±0.025**	0.417±0.025
age (>25)	660	**0.445±0.019**	0.419±0.020
passed p-s tests	740	**0.442±0.018**	0.405±0.021
failed p-s tests	300	0.437±0.027	**0.439±0.025**
first half	520	**0.456±0.022**	0.430±0.023
second half	520	**0.425±0.020**	0.418±0.021

Table 3.2: Overall, users confused our insertion results with real pictures 44% of the time, while confusing the results of Barron and Malik with real images 42% of the time. Interestingly, for the subpopulation of "expert" subjects, this difference became more pronounced (44% vs 36%). Each cell shows the mean standard deviation.

3.5.2 User study

In the study, each subject is shown series of two-alternative forced choice tests, where the subject chooses between a pair of images which he/she feels the most realistic. We tested three different tasks: (1) our method against real images, (2) the method of Barron and Malik against real images, and (3) our method against Barron and Malik. Figure 3.9 shows example trials from the first and third tasks.

Experiment setup For each task, we created 10 different insertion results using a particular method (either ours or Barron and Malik), ensuring the same object was used by each method, inserted at roughly the same location in the same scene. We also collected 10 real scenes (similar to the ones with insertion) for the tasks involving real images. Each subject viewed all 10 pairs of images for one but only one of the three tasks. For the 10 results by our method, the detail layer weights were manually selected (it is hard to apply the regression model as in Section 3.5.1 to the real scene illuminations) while the method of Barron and Malik does not have such options. Besides, we observe that the error introduced by Barron and Malik's shape tends to fall outside of the object region (e.g., wrong shadows casted in the scene, Fig. 3.11), which cannot be fixed by manipulating the details on the object.

We polled 300 subjects using Mechanical Turk. In an attempt to avoid inattentive subjects, each task also included four "qualification" image pairs (a cartoon picture next to a real image). Subjects who incorrectly chose any of the four cartoon picture as realistic were removed from our findings (6 in total, leaving 294 studies with usable data). We would like to make the userstudy image set and the collected data publicly available.

At the end of the study, we showed subjects two additional image pairs: a pair containing rendered spheres (one a physically plausible, the other not), and a pair containing line drawings of a scene (one with proper vanishing point perspective, the other not). For each pair, subjects chose the image they felt looked most realistic. Then, each subject completed a brief questionnaire, listing demographics, expertise, and voluntary comments. These answers allowed us to separate subjects into subpopulations: **male/female**, **age** < 25 / ≥ 25, whether or not the subject correctly identified both the physically accurate sphere *and* the proper-perspective line drawing at the end of the study (**passed/failed perspective-shading (p-s) tests**), and also **expert/non-expert** (subjects were classified as experts only if they passed the perspective-shading tests *and* indicated that they had expertise in art/graphics). We also attempted to quantify any learning effects by grouping responses into the **first half** (first five images shown to a subject) and the **second half** (last five images shown).

Results and discussion Overall, our user study showed that subjects confused our insertion result with a real image 44% of 1040 viewed image pairs (task 1); an optimal result would be 50%. We also achieve better confusion rates than the insertion results of Barron and Malik (task 2, 42%), and perform well ahead of the method of Barron and Malik in a head-to-head comparison (task 3, see Fig. 3.10).

Table 3.2 demonstrates how well images containing inserted objects (using either our method or Barron and Malik) hold up to real images (tasks 1 and 2). See Figure 3.9 (bottom) for an example trial from task 1. We observe better confusion rates (e.g. our method is confused with real images more than the method of Barron and Malik) overall and in each subpopulation except for the population who failed the perspective and shading

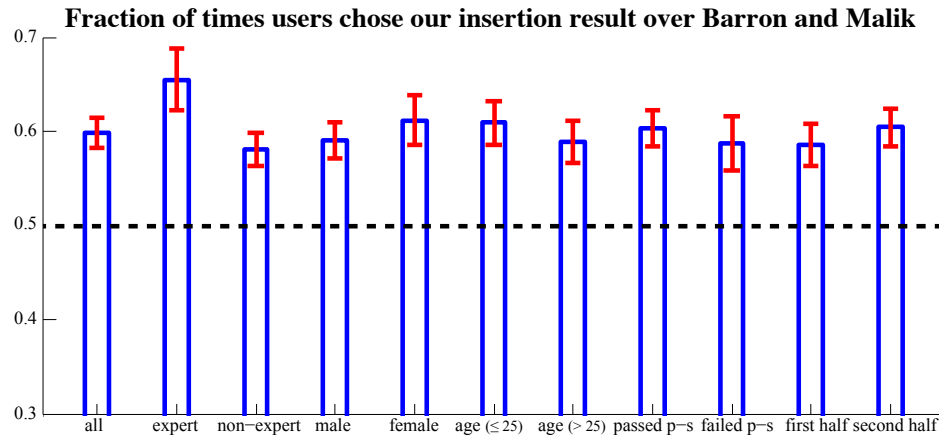


Figure 3.10: In a comparison of our results against that by the method of Barron and Malik. our results were chosen as more realistic in 60% of the trials ($N = 1000$). For all subpopulations, our results were preferred well ahead of the other as well. All differences to the dotted line (equal preference) are greater than two standard deviation. The "expert" subpopulation chose our insertion results most consistently.

tests in the questionnaire.

Karsch et al. [57] performed a similar study to evaluate their 3D synthetic object insertion technique, in which subjects were shown similar pairs of images, except the inserted objects were synthetic models. In their study, subjects chose the insertion results only 34% of the time, much lower than the two insertion methods in this study. While the two studies were not identical and performed by different populations, the results are nonetheless intriguing. We postulate that this large difference is due to the nature of the objects being inserted: we use *real* image fragments that were formed under real geometry, complex material and lighting, sensor noise, and so on; they use 3D models in which photorealism can be extremely difficult. By inserting image fragments instead of 3D models, we gain photorealism in a data-driven manner.

We also compare our method and the method of Barron and Malik head-to-head by asking subjects to choose those most realistic image when shown two similar results side-by-side (see Fig. 3.9 top for an example trial). Figure 3.10 summarizes our findings. Overall, users chose our method as more realistic in a side-by-side comparison on average 60% of the time in 1000 trials. In all subject subpopulations, our method was preferred by a large margin to the method of Barron and Malik; each subpopulation was at least two standard deviations

away from being "at chance" (50% – see the red bars and black dotted line in Fig. 3.10). Most interestingly, the expert subpopulation preferred our method by an even greater margin (66%), indicating that our method may appear more realistic to those who are good judges of realism.

(a) Input (b) Shape (c) Result of B & M (d) Our result

Figure 3.11: In this example, we built models from the cube in the input image (cyan box) and inserted it back into the scene. Sophisticated SfS methods (in this case, Barron and Malik [1]) can have large error and unstable boundaries that violates the generic view assumption. For object insertion, lighting *on* the object is important, but it is equally important that cast shadows and interreflected light look correct; shape errors made by complex SfS methods typically exacerbate errors both *on* and *around* the object (see cast shadows in c). Our shape is simple but behaves well in many situations and is typically robust to such errors (d). Best viewed in color at high-resolution.

Figure 3.12: A failure example under extreme lighting conditions. The left group shows a 3D model lit under four lighting directions: 3-quarter (3Q), left, front, top. The right group shows our results. Our model appears realistic when the lighting is not strongly directed (3Q; front), but looks unnatural in harsh conditions (left; top).

3.6 Conclusion and future work

We have proposed a new representation suitable for relighting image fragments and an effective workflow for inserting objects photorealistically into new images. Our models are simple yet robust to errors that make existing SfS methods infeasible for relighting tasks.

Through both quantitative and most importantly human subject studies, we found that our method is preferable to other methods for object relighting, and images created with our system are confusable (nearly at chance) with real images.

Due to the simple nature of our shape representation, the model can fail under extreme lighting conditions, i.e., strong point light source from extreme directions, or in the case of complex shapes (e.g. arm chairs or people in certain poses). See Figure 3.12 for a failure example on human faces. Large areas of strong shadow or highlight in the input image can also cause performance degrade for two reasons: (a) the current albedo-shading procedure works poorly for these cases, (b) the strong shadows and highlights will be (partly) kept in the shading detail layers and appear in subsequent relighting results. Fortunately, the visual system is very insensitive to inaccuracies of small shadows and highlights. Nonetheless, the model is best extracted from input images under diffuse multi-source lighting environments.

CHAPTER 4

PROGRESSIVELY DYNAMIC VIDEO LOOPING

4.1 Introduction

Many mobile devices now acquire high-definition video just as easily as photographs. With increased parallel processing, the gap in resolution between these two media is narrowing. It should soon become commonplace to archive short bursts of video rather than still frames, with the aim of better capturing the "moment" as envisioned by Cohen et al., [102].

Several recent techniques explore new ways of rendering short videos. Examples include cinemagraphs [18, 15, 103] and cliplets [16], which selectively freeze, play, and loop video regions to achieve compelling effects. The contrasting juxtaposition of looping elements against still backgrounds helps grab the viewer's attention. The emphasis in these techniques is on creative control.

We focus on automating the process of forming looping content from short videos. The goal is to render subtle motions in a scene to make it come alive, as motivated by Schödl et al., [14]. Many such motions are stochastic or semi-periodic, such as swaying grass, swinging branches, rippling puddles, and pulsing lights. The challenge is that these moving elements typically have different looping periods, and moreover some moving objects may not support looping at all. Previous techniques rely on the user to identify spatial regions of the scene that should loop and determine the best period independently for each such region.

Instead, we formulate video loop creation as a general optimization in which each pixel determines its own period. An important special case is that the period may be unity, whereby a pixel becomes static. Therefore the optimization automatically segments the

scene into regions with naturally occurring periods, as well as regions that are best frozen, to maximize spatiotemporal consistency. A key aspect that makes this optimization more tractable is to parameterize looping content so as to always preserve phase coherence.

Our other main contribution is to explore the concept of progressive dynamism (Figure 4.1). We extend the optimization framework to define a spectrum of loops with varying levels of activity, from completely static to highly animated. This spectrum has a compact encoding, requiring only a fraction of the storage of the input video. We show that this underlying structure also permits local selection of dynamism, for efficient runtime control of scene liveliness based on personal preference or mood. Applications include subtly animated desktop backgrounds and replacements for still images in slide shows or web pages.

Our contributions are:

- Using optimization to automatically segment video into regions with naturally occurring periods as well as static regions.
- Formulating video loop creation using 2D rather than 3D graph cut problems.
- Introducing the progressive video loop, which defines a nested segmentation of video into static and dynamic regions.
- Using the resulting segmented regions to enable interactive, seamless adjustment of local dynamism in a scene.
- Demonstrating an extremely compact encoding.

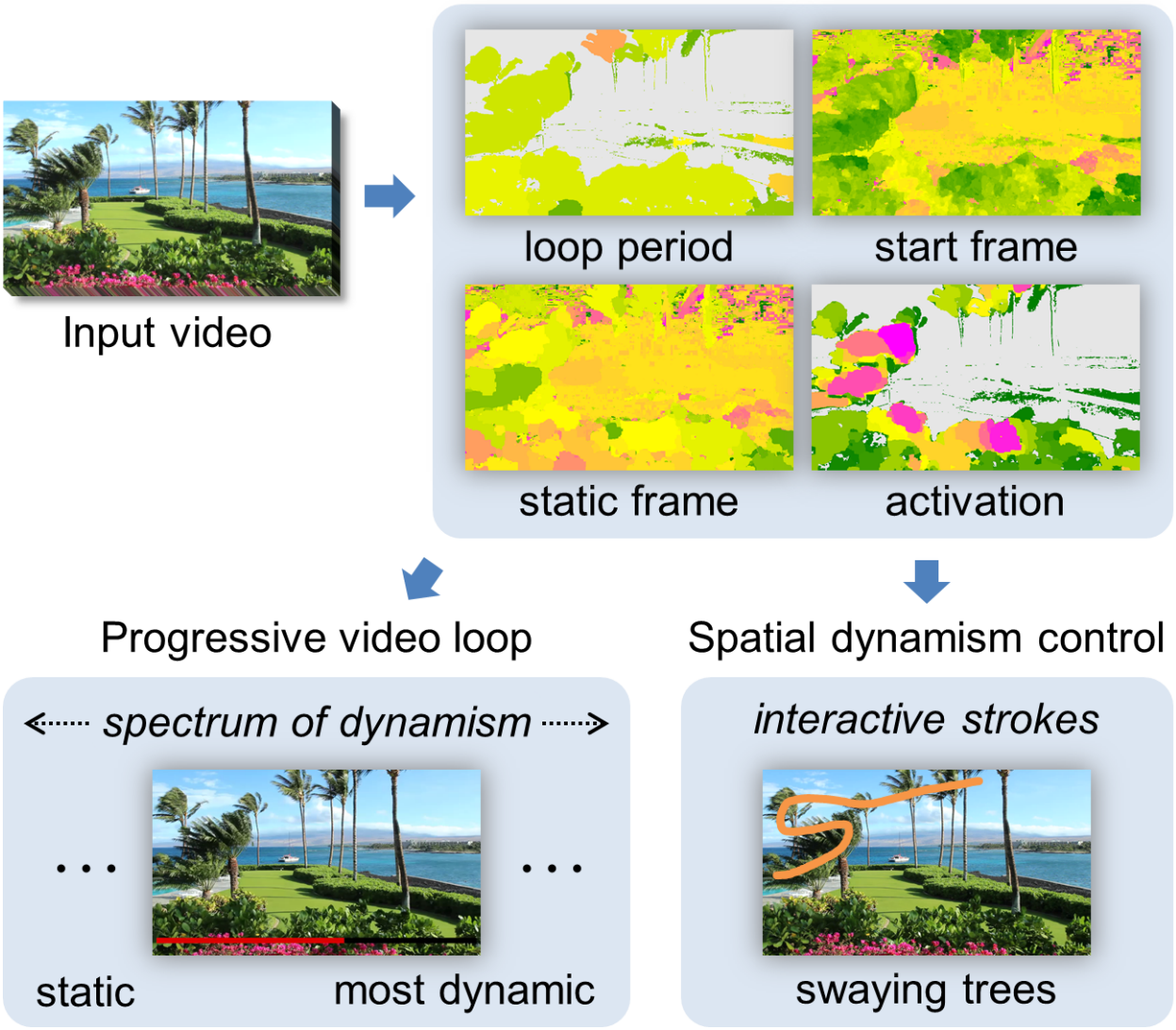


Figure 4.1: We optimize a set of video looping parameters. This compact encoding defines a spectrum of loops with varying activity and enables fast local control over scene dynamism.

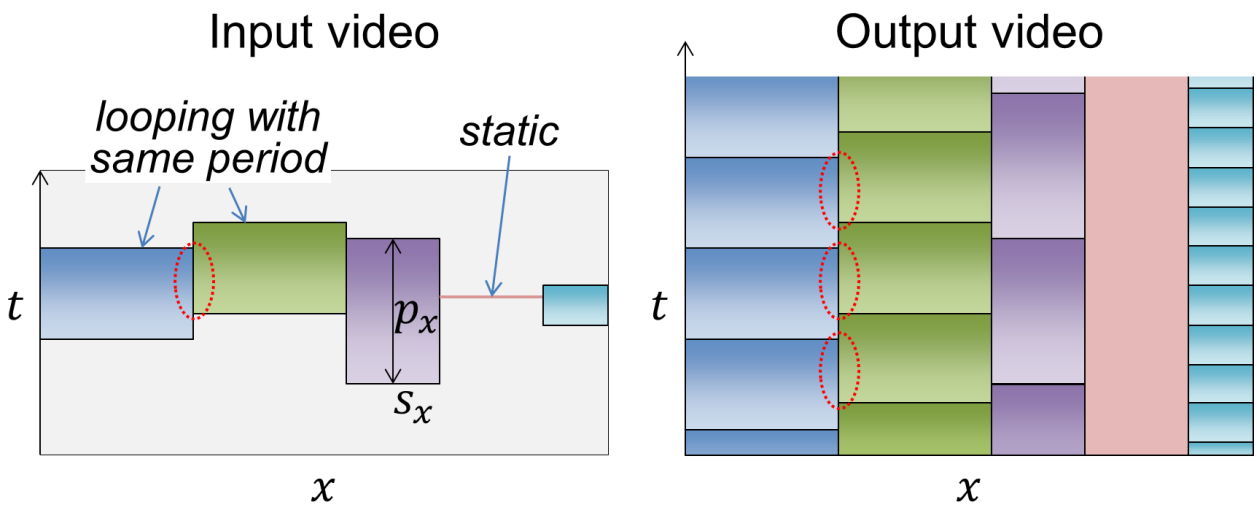


Figure 4.2: The video loop is specified by assigning each pixel a start time s_x and period p_x. Static pixels have $p_x = 1$. The time-mapping function locks phase to help maintain spatial consistency across pixels with the same period, as shown by the red ellipses.

4.2 Background

4.2.1 Creating video loops

Video textures

Schodl et al. [14] are the first to introduce the idea of synthesizing an looping video from a video sequence. The resultant video loop is called video texture in analogy to texture systhesis. A loop is formed by identifying temporally compatible (similar) frames for loop transition. For example, if frame j and frame k are similar and $k < j$, then transition from frame $j-1$ to frame k would be smooth and form a loop between frame k and frame $j-1$. A temporal window can be used in computing frame to frame compatibility to increase the temporal smoothness. By identifying multiple compatible transition pairs, the system can generate infinite, non-repeating video by stochastic traversal of the transition graph.

The method can generate interesting loops from simple scenes, like candle flame or water ripples. For scenes with more complex motions or motions of independent elements, there may not exist two frames that are compatible to each other. Several methods (below) define

more flexible schemes to form video loops at higher computational cost.

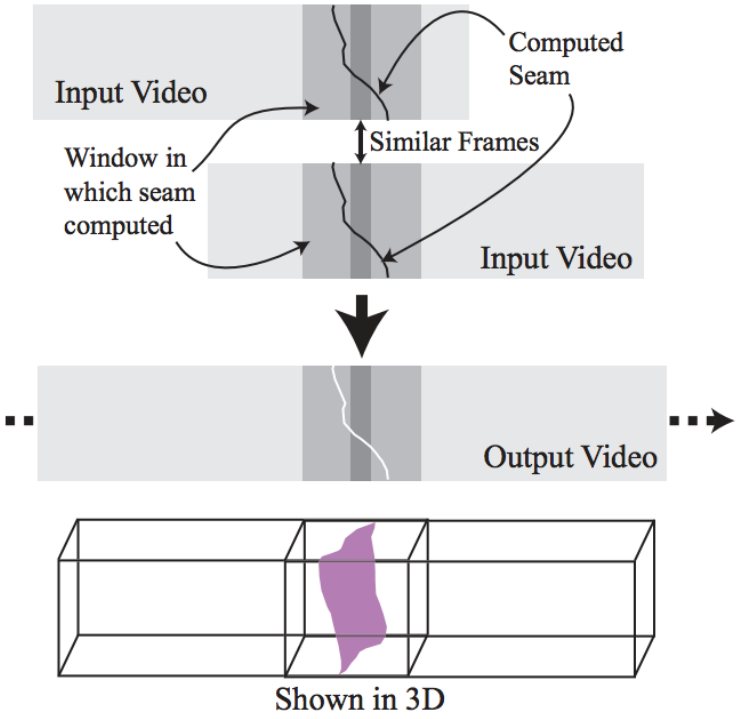


Figure 4.3: Graphcut video texture. Source: Kwatra et al. [9]

Graphcut video textures

Kwatra et al. [9] introduce graph cut [104] for texture synthesis and video looping.

To synthesize a larger texture, a new texture patch is placed over existing patch with overlap, and an optimal seam is computed by a binary graph cut over the 2D overlap grid. The graph-cut based texture synthesis is an advance from the image quilting algorithm by Efros and Freeman [105] in the sense of accounting for old seams. When a new patch is placed over a region where old seams exist, the old seam costs can be easily incorporated into the new graph cut problem and determine if any pixels from the new patch should cover over some of the old seams. In image quilting, the seam is computed by dynamic programming, which is known as a memoryless optimization procedure that does not keep track of old solutions.

Video loop synthesis is a natural extension from texture synthesis from 2D to 3D. To To form a loop, a video is placed to the end of itself with overlap. An optimal transition surface is computed by a binary graph over the 3D grid, see Figure 4.3. Compared to the original video texture method, in which transition happens at a whole frame, the transition surface provide a more flexible way to form a loop.

Because pixels do not transition at the same point, they may be placed next to new pixels. So in addition to the temporal compatibility (defined similarly as video texture, but at pixel level), spatial compatibility need also to be accounted. The basic principle for spatial compatibility is that if a new neighbor is similar to the neighbor in the original video, the cost is low; otherwise it is high. The temporal compatibility and spatial compatibility terms correspond to the unary and pairwise cost of the graph cut optimization, respectively. Standard graph cut optimization package are available online (e.g., http://vision.csd.uwo.ca/code/) to solve the problem.

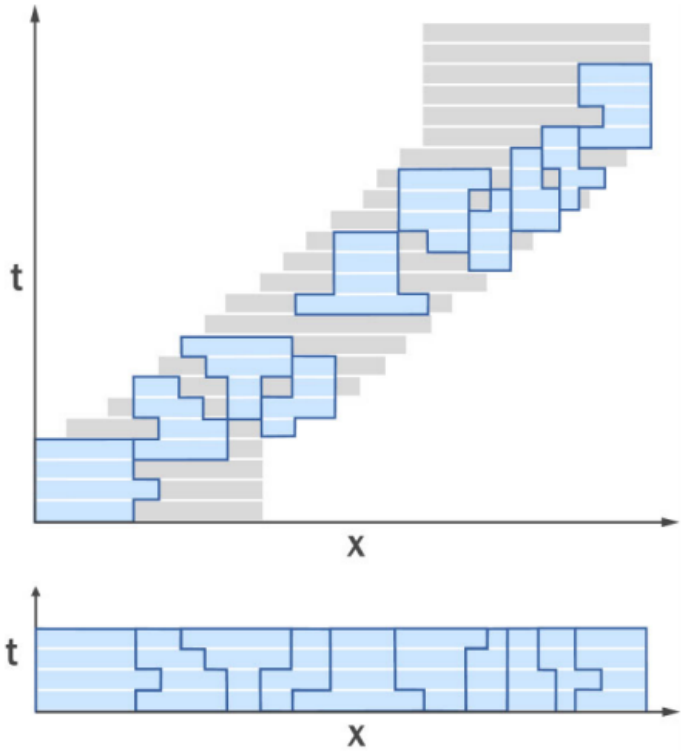


Figure 4.4: Panoramic video texture. Source: Agarwala et al. [10]

Panoramic video textures

Agarwala et al. [10] create panoramic video textures from a panning video sequence. As is shown in Figure 4.4, the input data, with registration, is incomplete in the whole spatiotemporal volume. Every location is captured during the interval it exists in the panning camera view. A wideview looping video is formed by taking pixels from the same spatial location but different temporal locations (not necessarily a continuous interval, nor do temporal order is conserved). Similar temporal and spatial costs are defined as is in [9]. All pixels are constrained the have the same period p, whose value is pre-determined with heuristic searching. The problem boils down to a multi-label graph cut defined over the 3D spatiotemporal grid, the labels being one of the p indices in the loop, or not used. Local minimum solution is obtained by the alpha-expansion algorithm [106].

Notice both the Graphcut video texture and Panoramic video texture solves for uniform period loops in each single cut. This is limited because for many independent moving elements may appear in the same scene, each element having a different looping period. Manually identifying such regions and compute the loops individually would be laborious. A procedure that automatically discover per-region, or per-pixel loop period is desirable.

Another issue with the Panoramic video texture method is that it does not constrain a loop at a location to be a continuous temporal interval, which are constraint implicitly made in both video texture and the graph cut video texture. Even though without the constraint loops are allowed to be formed more flexibly, most of them will be of high cost when temporal coherence is broken. So chances of finding better loops without the temporal constraint are low, while the benefit is that the 3D graph cut problem is then reduced to 2D.

The following method successfully avoid these two problems.

Progressively dynamic video looping

We introduce a new video looping algorithm that (1) automatically discover the natural looping period for each pixel and (2) formulate a graph cut optimization on a 2D grid. The

Method	Transition	Temporal	Period	Graph	#labels	optimality
Schödl'00	together	interval	single*			
Kwatra'03	not	interval	single	3D($H \times W \times F$)	2 (binary)	global
Agarwala'05	not	shuffled	single	3D($H \times W \times F$)	F	local
Liao'13	not	interval	multiple	2D($H \times W$)	$F \times P$	local

Table 4.1: Loop definition and optimization comparison.

computational complexity of the graph cut is similar to that of the Panoramic video texture. The details are described in chapter 4.

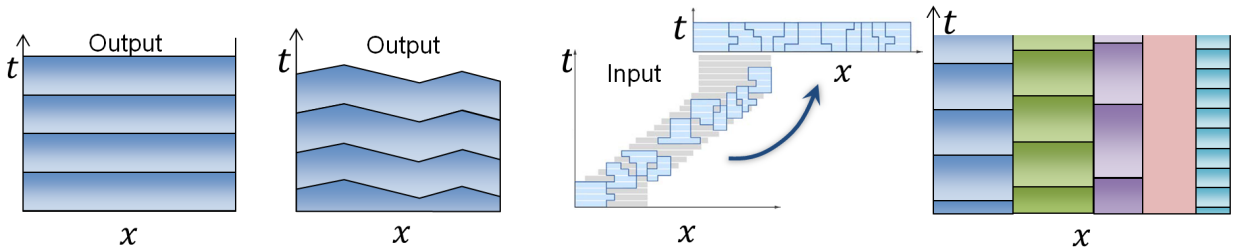


Figure 4.5: Different loop definition from the way output video loops are formed. From left to right: video texture, Graphcut video texture, Panoramic video texture, our approach.

Comparison of loop definition and optimization We now compare the above four video looping algorithm in terms of how the loop is defined and the complexity of optimization.

Figure 4.5 shows the four different loop definition schemes. Table 4.1 list a comparison over a few attributes. Video texture loops are formed most inflexibly, thus work only simple scenes. It is the only one that all pixels transition together, so no spatial cost is needed. The Graphcut video texture and Panoramic video texture formulate loop optimization in more complicated ways. However, they still produce single period loops. The Panoramic video texture method is the only one who loops are not constrained to be a continuous temporal interval, which would have reduced the graph complexity without much damage to the loop quality.

4.2.2 Cinemagraphs

The cinemagraph visual art invented by Beck and Burg [18] opens a brand new way to depict a scene. A cinemagraph consists of one or a few looping elements in a scene while the rest of the scene is deliberately frozen. Examples are flashing streetlights with still pedestrians, or a static face with the eyes blinking. The static part derives its power from imagination of what's implied beyond its spatial and temporal boundaries, while the looping part unfolds a temporal narrative. A proper combination of the two helps to capture the "moment" as envisioned by Cohen and Szeliski [102] and yields stronger visual impression than either image or video alone.

Tompkin et al. [15] present a tool for interactive authoring of cinemagraphs. Regions of motion in the video are automatically isolated. The user selects which regions to make looping and which reference frame to use for each region. Looping is achieved by finding matching frames.

Bai et al. [103] describe a method to selectively stabilize motions in video. The user sketches three types of strokes to indicate regions to be made static, immobilized, or fully dynamic. The method propagates the strokes across video frames using optical flow, warps the video to stabilize it, and solves a 3D MRF problem to seamlessly merge it with static content. Applications include cinemagraphs, motion visualization, and video editing.

Joshi et al. [16] develop a set of idioms (static, play, loop, and mirror loop) that let users to interactively juxtapose static and dynamic elements from a video to form a narrative. The juxtaposition of static and dynamic elements may be spatial (part of the frame is static while other parts are dynamic), temporal (a still followed by a dynamic and vice versa), or both. The temporal juxtaposition generates narratives that are temporally inconsistent with the original sequence but with novel visual effects. They generalize the media format from cinemagraphs and give it a new name *cliplets*, by allowing dynamic elements to not strictly form a loop, even though some of the idioms, like the mirror operator, strategically aims for loop creation.

4.2.3 User interaction

There are two types of user interaction when we use the term: (I) user assistance in model generation, and (II) user interaction *with* the model. The first type of interaction is tedious and we should always endeavor to avoid. The second type of interaction embraces creativity and is always appealing; the challenge is to identify what kind of interaction users would like and build it into the model during modeling stage.

A number of the aforementioned algorithms for video looping and cinemagraph creation require certain amount of user assistance. In panoramic video textures, user needs to manually select individual target regions for loop computation. The cinemagraphs creation work [15] asks user to select regions as well as reference frame. The selectively de-animating video work [103] takes three types of user strokes to guide de-animation and composition. The cliplets work [16] entirely rely on user specification to create an output. The user interaction in cliplets, however, is slightly different from those of the other ones. It enjoys a certain degree of creation freedom, which makes it somewhat like the type II interaction. But it is not operating on a pre-computed model but directly on the raw video data directly. This leaves some computational burden to user and makes interaction less intuitive. For example, which region has looping motion and where the boundary should be? Which start and end frame to use for smooth temporal transition? The users have to determine on each of these by eyeballing the video sequence. The system would be much more fun to use if such computations have been done by the machine so that users can focus on the creative part.

We avoid the first type interaction in our progressively dynamic video looping work. It is the first fully automated algorithm for cinemagraph creation. Furthermore, we build in a few interesting type II interactions in the model. First, we generate a level-of-dynamism looping structure, which encodes a spectrum of looping videos ranging from a static image to a highly animated scene. Traversal between the levels of dynamism is seamless and interactive controlled by a slider. So a user could easily adjust the level of dynamic elements in a scene according to personal preference or mood. Second, we compute a segmentation of the scene, each segment corresponds to an independently looping object or part. The user, therefore,

can pick an object by a simple stroke or mouse click and toggle its looping status (static or loop). This simple binary operation allows a user to configure a cinemagraph that is different from any of the computed results and unique in all.

4.3 Our loop definition

The input video is denoted as a 3D volume $V(x,t)$, with 2D pixel location x and frame time t. In forming a video loop, we assume that the input video has already been stabilized. Stabilization can be performed automatically, e.g., [15] or with user guidance, e.g., [103]. For our results we use either the "Warp Stabilizer" automated tool in Adobe After Effects or a standard feature-based global alignment method as in [16].

The goal is to construct an infinitely looping output video $L(x,t)$ with good spatiotemporal consistency, i.e., the loop should avoid undesirable spatial seams or temporal pops which occur when its content is not locally consistent with the input video. Because the input is stabilized, we form L by retrieving for each pixel some content associated with the same pixel in the input, as illustrated in Figure 4.2. This content may be either *static* or *looping*. In either case, it is represented as a temporal interval $[s_x, s_x + p_x)$ from the source video, where s_x is the start time and p_x is the period. A static pixel thus corresponds to the case $p_x = 1$.

More precisely, we define

$$L(x,t) = V\big(x, \phi(x,t)\big), \quad t \geq 0,$$

where the unknown is the time-mapping function

$$\phi(x,t) = s_x + ((t - s_x) mod p_x) \tag{4.1}$$

(Note the C/C++ remander operator "%" differs from the modulo operator "mod" for negative nubmers.) The complicated modulo arithmetic in this formula deserves further

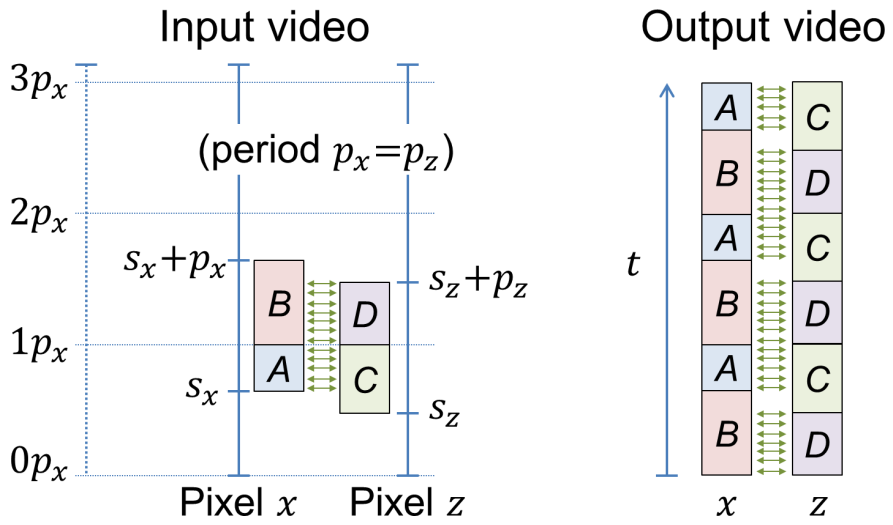


Figure 4.6: For adjacent pixels x and z with the same looping period $p_x = p_z$ and similar start times s_x, s_z, the in-phase time-mapping function of Equation (4.1) automatically preserves spatiotemporal consistency over a large portion of the output timeline. The green arrows show corresponding spatially adjacent pixels.

explanation. Intuitively, if two adjacent pixels are looping with the same period, it is usually desired that they be *in-phase* in the output loop. Figure 4.6 illustrate this important point. Two adjacent pixels x and z have the same looping period $p_x = p_z$ and retrieve content from input intervals $[s_x, s_x + p_x)$ and $[s_z, s_z + p_z)$ respectively. Although their start times s_x, s_z differ, their input time intervals have significant overlap. By wrapping these intervals in the output timeline using Equation (4.1), proper adjacency is maintained within the temporal overlap, and therefore spatial consistency is automatically preserved.

It is interesting to contrast this loop parameterization with that presented by Agarwala et al., [10, Fig. 5], which solves for time offsets between output and input videos. We instead assume these offsets are prescribed by phase coherence, and solve for start frames (Figure 4.2). As shown later in results, good video loops often have regions looping in-phase with a common optimized period but with many staggered per-pixel start times.

4.3.1 Construction overview

As in prior methods, we formulate video loop construction as an MRF problem. The goal is to find start times $\mathbf{s} = \{s_x\}$ and periods $\mathbf{p} = \{p_x\}$ that minimize the objective

$$E(\mathbf{s}, \mathbf{p}) = E_{\text{consistency}}(\mathbf{s}, \mathbf{p}) + E_{\text{static}}(\mathbf{s}, \mathbf{p}).$$

The first term encourages all pixel neighborhoods in the video loop to be consistent both spatially and temporally with those in the input video (Section 4.3.2). The second term penalizes the assignment of static loop pixels except in regions of the input video that are truly static (Section 4.3.3). Unlike in prior methods, the MRF graph is defined over the 2D spatial domain rather than the full 3D video volume. Another important difference is that the set of unknowns includes a looping period at each pixel.

4.3.2 Spatiotemporal consistency

In the generated video loop, each pixel's spatiotemporal neighbors should look similar to those in the input [10]. Because the domain graph is defined on the 2D spatial grid, the objective must distinguish spatial and temporal consistency:

$$E_{\text{consistency}}(\mathbf{s}, \mathbf{p}) = \beta E_{\text{spatial}}(\mathbf{s}, \mathbf{p}) + E_{\text{temporal}}(\mathbf{s}, \mathbf{p}).$$

Spatial consistency The term

$$E_{\text{spatial}} = \sum_{\|x-z\|=1} \frac{\gamma_s(x,z)}{T} \sum_{t=0}^{T-1} \begin{pmatrix} \|V(x, \phi(x,t)) - V(x, \phi(z,t))\|^2 + \\ \|V(z, \phi(x,t)) - V(z, \phi(z,t))\|^2 \end{pmatrix}$$

measures compatibility for each pair of adjacent pixels x and z, averaged over all time frames in the video loop. Thus the period T is the least common multiple (LCM) of all per-pixel periods. Equivalently, the objective can be formulated as $\lim_{T\to\infty} E_{\text{spatial}}$, the average spatial consistency over an infinitely looping video. We compute pixel value differences at

both pixels x and z for symmetry. Inspired by [9], the factor

$$\gamma_s(x,z) = 1/\Big(1 + \lambda_s \operatorname*{MAD}_t \big\|V(x,t) - V(z,t)\big\|\Big)$$

reduces the consistency cost between pixels when the temporal median absolute deviation (MAD) of their color differences in the input video is large, because inconsistency is then less perceptible. We use MAD rather than variance because it is less sensitive to outliers. (We set $\lambda_s = 100$ in all results.)

For efficient evaluation we distinguish four cases:

(1) When pixels x and z are both static, the energy reduces to

$$E_{\text{spatial}}(x,z) = \|V(x,s_x) - V(x,s_z)\|^2 + \|V(z,s_x) - V(z,s_z)\|^2.$$

(2) When only pixel x is static, the energy simplifies to

$$E_{\text{spatial}}(x,z) = \frac{1}{T}\sum_{t=0}^{T-1} \begin{pmatrix} \|V(x,s_x) - V(x,\phi(z,t))\|^2 + \\ \|V(z,s_x) - V(z,\phi(z,t))\|^2 \end{pmatrix}.$$

For each of the two summed vector norms and for each color coefficient $v_c \in V$, the sum is obtained as

$$\frac{1}{T}\sum_{t=0}^{T-1}\big(v_c(x,s_x) - v_c(x,\phi(z,t))\big)^2 = v_c^2(x,s_x) - \frac{2n_c(x,s_x)}{p_z}\sum_{t=s_z}^{s_z+p_z-1} v_c(x,t) + \frac{1}{p_z}\sum_{t=s_z}^{s_z+p_z-1} v_c^2(x,t).$$

We evaluate the two sums above in constant time by precomputing temporal cumulative-sum tables on V and V^2.

(3) When both pixels are looping with the same period $p_x = p_z$, the energy reduces to

$$E_{\text{spatial}}(x,z) = \frac{1}{p_x}\sum_{t=0}^{p_x-1}\begin{pmatrix}\|V(x,\phi(x,t)) - V(x,\phi(z,t))\|^2 + \\ \|V(z,\phi(x,t)) - V(z,\phi(z,t))\|^2\end{pmatrix}.$$

Moreover we detect and ignore the zero-valued terms for which $\phi(x,t) = \phi(z,t)$. As illustrated in Figure 4.6, for the common case where start times are similar, the large time intervals marked with green arrows can thus be ignored.

(4) Finally, when the pixels have different looping periods, we must generally compute the sum using $T = \text{LCM}(p_x, p_z)$. However, the apparent worst case when the two periods are relatively prime, i.e., $\text{LCM}(p_x, p_z) = p_x p_z$, is computed efficiently by recognizing that

$$\frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(a_i - b_j)^2 = \frac{1}{m}\sum_{i=0}^{m-1}a_i^2 + \frac{1}{n}\sum_{j=0}^{n-1}b_i^2 - \frac{2}{mn}\Big(\sum_{i=0}^{m-1}a_i\Big)\Big(\sum_{j=0}^{n-1}b_i\Big),$$

where $a_i = V_c(x,\phi(x,i))$ and $b_j = V_c(x,\phi(z,j))$. For symmetry reason, the expression of Φ includes a second set of terms $(a'_i - b'_j)^2$ where $a'_i = V_c(z,\phi(x,i))$ and $b'_j = V_c(z,\phi(z,j))$. We reuse the same precomputed cumulative-sum tables from case (2) to evaluate these terms in constant time.

We use this expected squared difference as an approximation even when the periods p_x, p_z are not relatively prime. This approximation provides an important ($6\times$) speedup, and we verified that it does not appreciably affect the quality of results.

Temporal consistency The objective term

$$E_{\text{temporal}} = \sum_x \begin{pmatrix}\|V(x,s_x) - V(x,s_x+p_x)\|^2 + \\ \|V(x,s_x-1) - V(x,s_x+p_x-1)\|^2\end{pmatrix}\gamma_t(x)$$

compares for each pixel the value at the loop start and the value right after the loop end, and for symmetry also the value just before the loop start and at the loop end.

Because looping discontinuities are less perceptible when a pixel varies significantly over

time in the input video, we attenuate the consistency cost using the factor

$$\gamma_t(x) = 1/\Big(1 + \lambda_t \operatorname*{MAD}_t \big\|V(x,t) - V(x,t+1)\big\|\Big),$$

which estimates the temporal variation at the pixel based on the median absolute deviation of successive pixel differences. (We set $\lambda_t = 400$ in all results.)

For any pixel assigned as static (i.e., $p_x = 1$), E_{temporal} computes the pixel value difference between successive frames and therefore favors pixels with zero optical flow in the source video. While this behavior is reasonable, in practice we find that it prevents interesting moving objects from being frozen in the static image. Instead, we let the temporal energy be zero for any pixel assigned to be static.

For looping pixels, we considered introducing a factor $1/p_x$ that would account for the fact that a shorter loop reveals a temporal discontinuity more frequently in the output. However, this was found undesirable as it over-penalizes loops with small periods relative to longer loops with equal energy.

4.3.3 Per-pixel dynamism prior

If all pixels are assigned to be static from the same input frame, the loop attains perfect spatiotemporal consistency. To penalize this trivial solution we seek to encourage pixels that are dynamic in the input video to also be dynamic in the loop. We therefore introduce a regularization term E_{static} which adjusts the energy objective based on whether the neighborhood N of each pixel has significant temporal variance in the input video. If a pixel is assigned a static label, it incurs a cost penalty c_{static} but this penalty is reduced according to the temporal variance of the pixel's neighborhood. Thus we define $E_{\text{static}} = \sum_{x|p_x=1} E_{\text{static}}(x)$ with

$$E_{\text{static}}(x) = c_{\text{static}} \min\Big(1, \lambda_{\text{static}} \operatorname*{MAD}_t \big\|N(x,t) - N(x,t+1)\big\|\Big),$$

where $\lambda_{\text{static}} = 100$ and N is a Gaussian-weighted spatiotemporal neighborhood with $\sigma_x = 0.9$ and $\sigma_t = 1.2$.

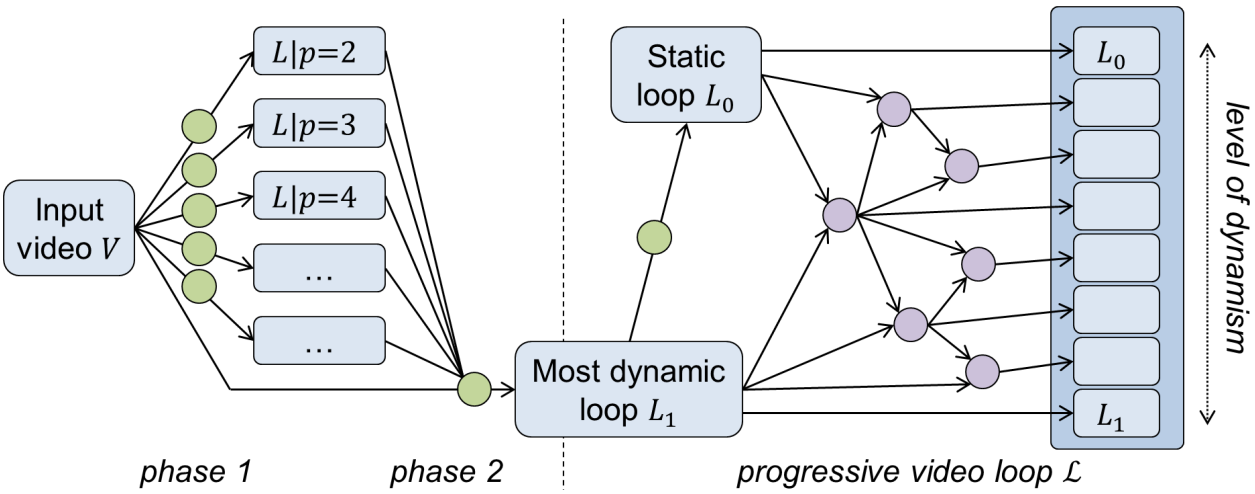


Figure 4.7: We construct an optimized video loop in two phases: finding optimized loops $L|p$ for each period p, then spatially merging these loops. Green circles denote multilabel graph cuts. Next, creating a progressive video loop involves a recursive partition using fast binary graph cuts, shown as purple circles.

4.3.4 Optimization algorithm

Poor results with traditional graph cut Our initial approach was to solve the MRF optimization using a standard multilabel graph cut algorithm, where the set of pixel labels is the outer product of candidate start times $\{s\}$ and periods $\{p\}$. However, this approach converges to poor local minima. These minima contain overly large regions with a common loop period and staggered start times. The problem is that a graph-cut alpha expansion (or alpha-beta swap) only considers a single new candidate label. As the algorithm tries to change the solution to a possibly better looping period p, it must consider a label that pairs this period p with a single start time s. This restriction prevents the optimization from jumping to another valley in the energy landscape where the new period is allowed different start times at different pixels. Jumping out of the local minimum would require considering multiple target labels simultaneously.

Two-phase approach Instead, we introduce an optimization procedure that works in two phases (left portion of Figure 4.7):

(1) For each candidate looping period $p > 1$, we find the per-pixel start times $s_{x|p}$ that

create the best video loop $L|p$ with just that period, by solving a multilabel graph cut.

(2) Next we solve for per-pixel periods $p_x \geq 1$ that define the best video loop $(p_x, s_{x|p_x})$ using the pixel start times obtained in phase 1, again by solving a multilabel graph cut. Here the set of labels includes all the periods $p > 1$ considered in the first phase, together with all possible frames s'_x for the static case $p = 1$. Essentially, the optimization merges together regions of $|\{p\}| + |\{s\}|$ different candidate loops: the optimized loops found in phase 1 for periods $\{p\}$ plus the static loops corresponding to the frames $\{s\}$ of the input video.

The graph cuts in both phases are solved using the iterative alpha-expansion algorithm of Kolmogorov and Zabih [107]. This algorithm assumes a regularity condition on the energy function, namely that for each pair of adjacent nodes and any three labels α, β, γ, the spatial cost should satisfy $c(\alpha, \alpha) + c(\beta, \gamma) \leq c(\alpha, \beta) + c(\alpha, \gamma)$. However, this constraint is not guaranteed in phase 2. One reason is that when two adjacent pixels are assigned the same period, the fact that they may have different start times means that their spatial cost $c(\alpha, \alpha)$ may be nonzero. Fortunately, the start times are solved in phase 1 to minimize this cost, so it is likely small.

Because the regularity condition does not hold, the nice theoretical bounded-approximation guarantees of the alpha expansion algorithm no longer hold. Nonetheless, several researchers have reported good results in this case e.g., [9, 52]. The workaround is to adjust some edge costs when setting up each alpha expansion pass. Specifically, we add small negative costs to the edges (β, γ) such that the regularity condition is satisfied.

Another reason that the energy function is irregular is that we omit taking a square root of $E_{\text{spatial}}(x, z)$ to make it a Euclidean distance rather than a squared distance. We find that introducing the square root yields inferior results.

Because the iterative multilabel graph cut algorithm may find only a local minimum of the objective function, the initial state is important. For phase 1, we initialize s_x to minimize temporal cost, and for phase 2, we select p_x whose loop $L|p_x$ has lowest spatiotemporal cost at pixel x.

4.4 Progressive video loops

We now generalize from a single video loop L to a spectrum of loops $\mathcal{L} = \{L_d \mid 0 \leq d \leq 1\}$ where d refers to *level of dynamism* – a normalized measure of the temporal variance in the video loop (Section 4.4.2). At one end of the spectrum, the least-dynamic loop L_0 corresponds to a static image (where each pixel may be copied from a different frame of the input video). At the other end of the spectrum, the most dynamic loop L_1 has the property that nearly all its pixels are looping.[1]

To define the spectrum of loops, we require that each pixel have exactly two possible states:

- *static*, with a color value taken from a single frame s'_x of the input video, or
- *looping*, with a looping interval $[s_x, s_x + p_x)$ that includes the static frame s'_x.

We then establish a nesting structure on the set of looping pixels by defining the *activation threshold* $a_x \in [0, 1)$ as the level of dynamism at which pixel x transitions between static and looping.

Thus, a progressively dynamic video loop has the time-mapping

$$\phi_d(x,t) = \begin{cases} s'_x & \text{if } d \leq a_x, \\ \phi(x,t) & \text{otherwise.} \end{cases}$$

The goal is to determine the activation threshold a_x, static frame s'_x, loop start frame s_x, and period p_x at each pixel such that all video loops in $\mathcal{L}$ have good spatiotemporal consistency.

4.4.1 Construction overview

As shown in Figure 4.7, our approach consists of three steps:

[1] Forcing all pixels to loop may introduce bad artifacts for scenes with non-loopable content – artifacts undesirable enough that it is not worthwhile presenting these to the user.

(1) We solve for a **most dynamic** loop L_1 using the two-phase algorithm of Section 4.3 by setting c_{static} to a large value, $c_{\max} = 10$.

(2) We create a **static** loop L_0 (i.e., a "reference image") by leaving as-is any pixels already static in L_1 and solving for the best static frame s'_x for each remaining pixel (Section 4.4.3).

(3) Having obtained the parameters (s'_x, s_x, p_x) defining the two loops $\{L_0, L_1\} \subset \mathcal{L}$, we assign an **activation threshold** a_x at each pixel to establish the loop spectrum $\mathcal{L}$. This step uses a recursive binary partition over c_{static} between L_0 and L_1 (Section 4.4.4).

4.4.2 Parameterization of progressive video loop

The most straightforward way to parameterize the progressive loop spectrum $\mathcal{L}$ is using the static-cost parameter c_{static} that is varied during construction. However, the level of activity in the loop often changes quite non-uniformly (Figure 4.9) and differs significantly across videos. We find that a more intuitive parameterization is to use a normalized measure of temporal variance within the loop. Let

$$\text{Var}(L) = \sum_x \operatorname*{Var}_{s_x \le t < s_x + p_x} \big(V(x,t)\big)$$

measure the temporal variance of all pixels in a video loop L. We define the *level of dynamism* as the temporal variance normalized relative to the most dynamic loop L_1:

$$\text{LOD}(L) = \text{Var}(L) / \text{Var}(L_1).$$

Thus by definition, the most dynamic loop has $\text{LOD}(L_1) = 1$ and the static loop has $\text{LOD}(L_0) = 0$.

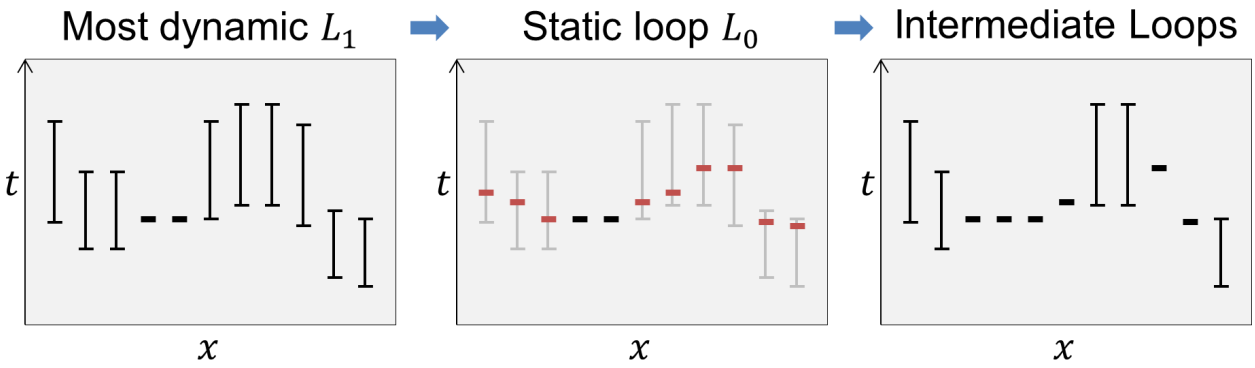


Figure 4.8: The looping parameters for the most static loop and the subsequent intermediate loops are constrained to be compatible with previously computed loops.

4.4.3 Construction of static loop

To obtain the static loop L_0 in step 2, we use the second-phase optimization of Section 4.3, setting $c_{\text{static}} = 0$, and enforcing the constraint $s_x \leq s'_x < s_x + p_x$ as shown in Figure 4.8. For this step we also introduce an extra data term that penalizes color differences from each static pixel to its corresponding median value in the input video. Encouraging median values helps create a static image that represents a "still" moment, free of transient objects or motions [108].

4.4.4 Optimization of per-pixel activation thresholds

The assignment of activation thresholds in step 3 operates as follows. For each pixel x looping in the most dynamic loop L_1, we know that its transition from static to looping must occur between loops L_0 and L_1, and therefore its activation threshold satisfies $0 \leq a_x < 1$.

We form an intermediate loop by setting $c_{\text{static}} = (0 + c_{\max})/2$ (halfway between the settings for L_0 and L_1) and constraining each pixel x to be either static as in L_0 or looping as in L_1. Minimizing E is thus a *binary* graph cut problem. Let d be the resulting loop's level of dynamism, so the loop is denoted L_d. The assignment of each pixel as static or looping in loop L_d introduces a further inequality constraint on its activation threshold a_x, i.e., either $a_x < d$ or $a_x \geq d$. Then, we further partition the intervals $[L_0, L_d]$ and $[L_d, L_1]$ recursively to

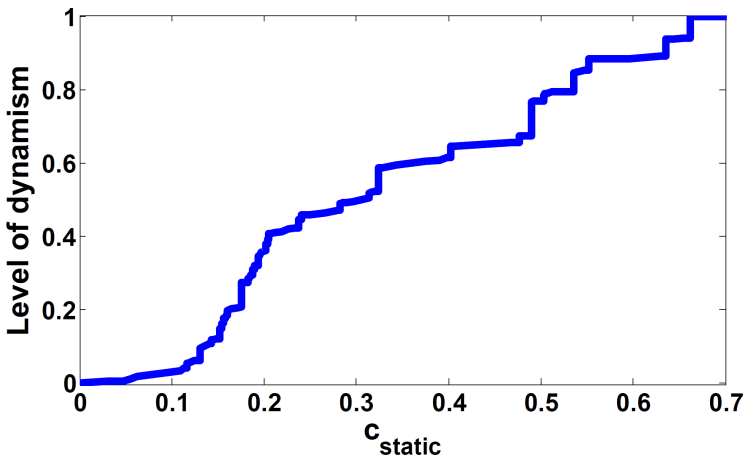


Figure 4.9: A video loop's level of dynamism is often non-uniform as a function of the optimization parameter c_{static}.

precisely define a_x at all pixels.

In the limit of recursive subdivision, the activation threshold of each pixel converges to a unique value. Recursion terminates when the change in the static-cost parameter c_{static} becomes sufficiently small ($< 1.0e-6$) or when the difference between the levels of dynamism of the two loops is sufficiently small (< 0.01). As a postprocess, each activation level is adjusted to lie at the midpoint of its vertical step in Figure 4.9 (rather than at the maximum of the step).

In the spectrum $\mathcal{L}$, there are intervals of dynamism over which the loop does not change, i.e., the vertical steps in Figure 4.9. Such discontinuities must exist, because the dynamism level is continuous whereas the set of possible loops is finite. The reason that some intervals are large is that maintaining spatiotemporal consistency requires some spatial regions to transition coherently. For some videos this leads to significant jumps in dynamism. To reduce these jumps, we lower the spatial cost parameter β from 10 to 5 during step 3; however, the tradeoff is that some of the new transitions are more noticeable.

Ordering of progressive dynamism Another issue is that in a progressive video loop, we would prefer that the activation threshold for subtle loops (with less activity) be smaller than that for highly dynamic loops. Unfortunately, with E_{static} as defined in Section 4.3.3,

varying c_{static} has the opposite effect: when c_{static} is low (near the static video L_0), pixels with high temporal variance benefit from the greatest drop in E_{static} when they transition to looping, and therefore the most active loops are introduced first. (Conversely, when c_{static} is high near the most dynamic video L_1, only pixels with low temporal variance have sufficiently small E_{static} penalty to become static, and therefore the least active loops are introduced late.)

To address this, only for the duration of step 3 we redefine $E_{\text{static}}(x) = c_{\text{static}}\Big(1.05 - \min\big(1, \lambda_{\text{static}}\,\text{MAD}_t\,\big\|N(x,t) - N(x,t+1)\big\|\big)\Big)$. Because the loops L_0, L_1 bounding the recursive partition process are fixed, the only effect is to modify the activation thresholds and thereby improve the ordering in which the loops appear.

4.4.5 Implementation details

In most of our results, the input video is a 5-second segment recorded at 30 frames/sec. To reduce computational cost we quantize loop start times and periods to be multiples of 4 frames. Also, we set a minimum period length of 32 frames.

As another speedup, to compute the spatial cost between two pixels looping with the same period (case (3) in Section 4.3.2), we aggregate groups of 4 consecutive pixels into 12-dimensional vectors, and perform principal component analysis (PCA) as a preprocess to project these vectors into an 8 dimensional space. This PCA projection retains over 99% of the data's variance in our tests.

As a further speed-up, we judiciously use OpenMP parallelization in our graph cut optimizations and only use a few iterations through all candidate alpha-expansion labels. We found that only two iterations are necessary for the second phase of our optimization and one is sufficient for all other phases.

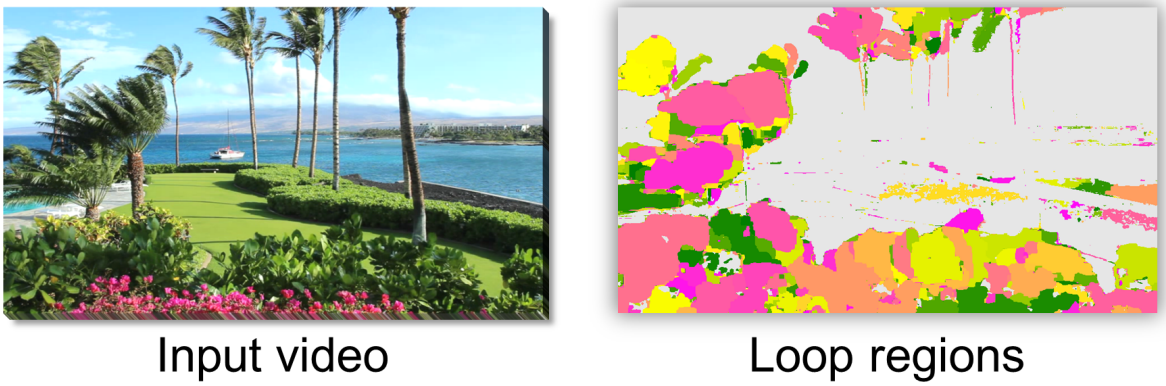


Figure 4.10: Segmentation of scene into independent looping regions, visualized with random colors and static pixels in light gray.

4.5 Interactive spatial control over dynamism

The per-pixel periods and activation levels induce a segmentation of the scene into independent looping regions. Whereas the progressive video loop defines a single path through this space, we can let the user adapt dynamism spatially by selectively overriding the looping state per region.

For fine-grain control, the selectable regions should be small. Yet, they must be sufficiently large to avoid spatial seams when adjacent regions have different states. We place two adjacent pixels in the same region if (1) they share the same looping period, (2) their temporal extents overlap, and (3) they have the same activation level. A flood-fill algorithm finds the equivalence classes for the transitive closure of this relation, as shown in Figure 4.10.

Our prototype system offers a simple interface to manipulate dynamism over the regions. As the cursor hovers over the video, the local underlying region is highlighted in yellow, and the other regions are shaded with a color code to delineate each region and its current state (shades of red for static and shades of green for looping). Clicking the mouse on the current highlighted region toggles its looping state. Alternatively, dragging the cursor starts the drawing of a stroke. All regions that overlap the stroke are activated or deactivated depending on whether the shift key is pressed. The action is instantaneous, so prior strokes are not retained. Please refer to the accompanying video for examples.

4.6 Local alignment

In some cases, scene motion or parallax can make it difficult to create high-quality looping videos. For these cases, shown in our supplemental video, we perform local alignment of the input video content to enable better loop creation. This is related to the scenario and approach explored by Bai et al., [103], with a few significant differences. Because their focus is creative control, their approach uses several types of user-drawn strokes to indicate regions to stabilize, keep static, and loop. In contrast, our approach is automatic and does not require any user input, and the creative control (see Section 4.5) is independent of the alignment process.

Our approach is to treat strong, low-frequency edges as "structural" edges that must be aligned directly and high spatiotemporal frequency areas as "textural" regions whose flow is smoothly interpolated. The visual result is that aligned structural edges appear static leaving the textural regions dynamic and able to be looped. We achieve this by using a pyramidal optical flow algorithm [109], with a high degree of smoothing, to align each frame of the video to a reference video frame (t_{ref}). The reference frame is chosen as the frame which is most similar to all other frames before local alignment.

Within the optimization algorithm (Section 4.3), we introduce two additional data terms. The first term

$$E_{\text{aligned}}(x) = \begin{cases} \infty & \text{if } p_x = 1 \text{ and } s_x \neq t_{ref}, \\ 0 & \text{otherwise,} \end{cases}$$

forces all static pixels (i.e., not looping) to be taken from the reference frame. The other term

$$E_{\text{flow}}(x) = \begin{cases} 0 & \text{if } p_x = 1, \\ \lambda_f \max\limits_{s_x \leq t < s_x + p_x} F(x,t) & \text{otherwise,} \end{cases}$$

penalizes looping pixels in areas of low confidence for optical flow, where $F(x,t)$ is the flow reprojection error (computed at the next-to-finest pyramid level) for a pixel x at time t aligned to the reference frame t_{ref}. We set $F(x,t) = \infty$ for any pixels where the reprojection

error is larger than the error before warping with the flow field or where the warped image is undefined (due to out-of-bounds flow vectors). (We set $\lambda_f = 0.3$ in all results.)

The result of these two terms is that the optimization will avoid loops aligned with poor flow and force regions that cannot be aligned at all to take on values from the static reference frame, which has no alignment error by construction. The looping areas then come from pixels where the flow error at a coarse level of the pyramid is low.

4.7 Rendering

Crossfading We apply crossfading to help mask spatial and temporal discontinuities in a video loop, but only where spatial and temporal inconsistencies are significant, so as to avoid excessive blurring elsewhere. We perform temporal crossfading during loop creation, using a linear blend with an adaptive window size that increases linearly with the loop's temporal cost (up to ± 5 frames). Spatial crossfading is performed at runtime using a spatial Gaussian filter G at a subset of pixels S. The set S consists of the spatiotemporal pixels with a large spatial cost (≥ 0.003), as well as the pixels within an adaptive window size that increases with the spatial cost (up to a 5×5 neighborhood). For each pixel $x \in S$ we compute:

$$L(x,t) = \sum_{x'} G(x' - x)\, V\big(x, \phi(x',t)\big).$$

Smooth progressive transitions As the level of dynamism is varied, some fraction of pixels transition from static to looping or vice versa. In either case, it is desirable to maintain spatiotemporal consistency to avoid visible artifacts.

Our initial strategy sought to exploit the fact that the static frame at each pixel lies within the temporal range of its loop. A natural idea is to wait until the (active or potential) loop reaches this frame before transitioning (from or to) the loop. Although this eliminates temporal pops, it has two significant drawbacks. First, it introduce a significant lag in responsiveness. Second, the transition frame often differs at adjacent pixels, resulting in

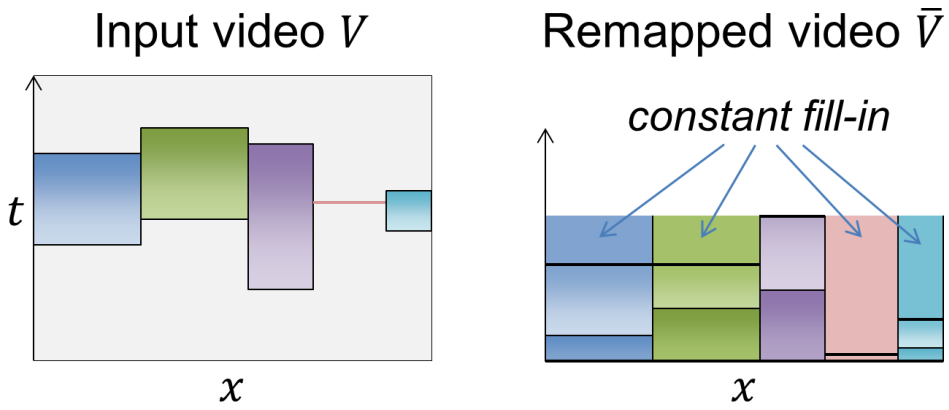


Figure 4.11: The accessed input is remapped to a shorter video $\bar{V}$, whereby the last value of each pixel's loop is held constant.

Name	Figure	Resolution	Time (min)	Compressed (KB)		Compressed remapped (KB)	
				Video V	(s', s, p, a)	Remapped $\bar{V}$	$(\bar{s}', p, a)$
Pool	4.12a	960x540	10	4,415	189	1,930	69
Drummers	4.12b	960x540	9	1,952	202	532	90
Waterwheel	4.12c	960x540	9	4,822	183	3,182	72
Aquarium	4.12d	960x540	8	2,065	238	887	98
Flowers	4.12e	960x540	9	4,386	198	1,498	95
Pinatas	4.12f	960x540	10	5,165	153	2,473	66
Flags	4.12g	960x540	10	3,780	164	1,585	59
Citylights	4.12h	960x540	6	1,767	192	736	75

Table 4.2: Timing and compression results. The optimized parameters (static frame s', loop start frame s, period p, and activation threshold a) are encoded into a PNG file. Remapping the video significantly reduces size while preserving the ability to adapt dynamism.

temporary but noticeable spatial inconsistencies. Instead, we simply crossfade between the static and looping states over an interval of 20 frames.

4.8 Compression

Besides the input video V, the progressive loop representation $\mathcal{L}$ needs only four per-pixel parameters (s'_x, s_x, p_x, a_x). These parameters have strong spatial coherence as visualized in Figure 4.12. We store the parameters into a four-channel PNG images, with activation thresholds a_x quantized to 8 bits. As shown in Table 4.2, the compressed representation is

about 200 KB per video, a small fraction of the video itself.

Because the progressive loop spectrum $\mathcal{L}$ accesses only a subset of the input video, we repack this content into a shorter video $\bar{V}$ of $\max_x p_x$ frames, by evaluating the initial frames of the most-dynamic loop L_1:

$$\bar{V}(x,t) = V\big(x, \phi_1(x,t)\big), \quad 0 \le t < \max_x p_x.$$

The time-mapping function is then extremely simple:

$$\bar{\phi}(x,t) = t \bmod p_x.$$

Note that it becomes unnecessary to store the loop start times s_x which have high entropy. The static frames are adjusted as $\bar{s}'_x = \bar{\phi}(x, s'_x)$. We reduce the entropy of unused portions of the shortened video by freezing the last pixel value of each loop (Figure 4.11). As shown by the compression results in the right columns of Table 4.2, this remapped video representation ($\bar{V}$ together with $\bar{s}'_x, p_x, a_x$) achieves significant space savings.

4.9 Results and discussion

Our procedure for constructing progressive video loops is fully automatic, and we have tested it on over a hundred videos. This includes a variety of casually shot videos from the authors (filmed using smartphones, handheld, and tripod-mounted cameras) and videos from five related works [14, 9, 15, 103, 16]. In all cases, we use the same default parameter settings. We enable local alignment for a few of the videos (those of Bai et al., [103]).

Our tests limit input videos to 5 seconds or less at up to 30 frames/second. Most of our results are computed at a resolution of 960×540 pixels, downsampling the input video if necessary. As shown in Table 4.2, the processing time for videos of this size is about 8–10 minutes. All our examples are computed on a 2.5GHz Intel Xeon L5420 (2 processor) PC with 16 GB of memory.

In supplemental material we include video results at HD (1920×1080) resolution. These are obtained by upsampling the looping parameters computed at the lower resolution (with nearest-pixel filtering), and using the resulting time-mapping on the HD input.

Figure 4.9 shows a selection of eight results using a diverse set of input videos. We show the static image corresponding to the least-dynamic loop L_0 and the per-pixel looping parameters: the period p_x, start frame s_x, and activation parameter a_x. Note how these parameter maps capture and encode a spatiotemporal segmentation of the video. Please see our supplementary video to appreciate these results and to see more examples.

The results show that there are generally a few dominant periods, which are a function of the scene's motion; however, within one period there are often many staggered per-pixel start times. We have found that having both per-pixel periods and start times to greatly increase the quality of the results, as demonstrated in our supplementary video.

Our supplementary video also shows how a user can explore a spectrum of dynamic imagery by interactively varying the level of dynamism of the looping video. The extremes of the spectrum produce a static image or a fully looping video, respectively, while values in the middle produce cinemagraph-type imagery. We also show how a user can override the prioritization encoded in the activation parameter, by changing the looping state of regions in an ad hoc way using mouse clicks and strokes.

Lastly, our supplementary video shows results with data from five previous works [14, 9, 15, 103, 16] along with side-by-side comparisons. The first two are automatic schemes like ours, and on their datasets our results are similar in quality or have slightly fewer artifacts. The next three schemes require user interaction, and our results are comparable in quality while requiring no interaction. Of course, without a user in the loop, we do not achieve the same semantics. Some of the videos of Bai et al., [103] are challenging due to large motions; where our automated alignment fails we still produce results with good quality but different semantics. Possibly our results could be improved with a more sophisticated optical flow method.

Compared to the approaches of Schödl et al., [14] and Kwatra et al., [9], our framework

sacrifices flexibility for performance by encoding only a single, contiguous loop per pixel. Nonetheless, our results are quite good. Notably, having fewer temporal transitions leads to larger spatiotemporal regions that are coherent with the input video and are therefore automatically free of inconsistencies. Achieving high-quality transitions comes largely from the fact that we optimize both loop periods and start times at each pixel, which no prior work has done. What we lose in temporal variety by having only a single loop per-pixel, we gain by having different loop periods across pixels. In other words, having multiple loops as in video textures [14] may be more important for a perception of variety when the loop is global across the image.

4.10 Conclusions and future work

In this work, we presented a method for creating a spectrum of looping videos from short input videos. As a result of per-pixel optimization of the loop period and start time, our method can create a high-quality, fully looping video without any user input. Our optimization also models scene liveliness and creates a segmentation that encodes spatial regions that loop contiguously. Along with this, we compute a default prioritization for transitioning these regions from static to dynamic with minimal visual artifacts.

A user may interactively override this prioritization while still exploiting a segmentation structure that conveniently identifies good independently looping regions. Using simple interactions, a user can create looping videos with a desired amount of scene liveliness. Our optimization is fairly efficient and produces results that are highly compressible. We have addressed many challenges in creating progressively dynamic videos; however, our results also suggest several areas for future work.

Limitations There are a few common failure cases and limitations we have encountered. Primarily, it is difficult to encode the semantic relationships between regions in a video. In some cases our optimization will produce results that have no perceivable artifacts, yet the

semantic relationship between objects is disturbed. Post-hoc user interaction can compensate for many of these semantic errors; however, an interesting opportunity for future work is to provide lightweight interaction for steering the optimization directly.

A second limitation is capturing motion with long periods, such as moving smoke or steam, or when a fine selection of periods is needed (i.e., where our sparse sampling is insufficient). These are not fundamental limitations of our approach, but limitations of performance. There are several graph cut optimization approaches that could address this, such as using a hierarchical solver [10].

Another limitation of our current implementation is that residual spatial seams are sometime visible for low-frequency content like clouds or steam. In these cases, the spatial support of our current crossfading approach is not enough to diffuse the edges perceptually. A fast Poisson or Laplacian blend [110, 111] could be used instead.

Lastly, while we have designed our method to be adaptive to diverse video content and thus minimized the need for "parameter tweaking", there are still cases where adjusting the spatial cost parameter β improves results. The fundamental reason for this is that "texture-like" and "object-like" content have different gradient distributions. In future work, we believe it is possible to learn a data-dependent function for β, by leveraging ideas from content-adaptive image priors [112].

New Features There are several new features and areas of research that are interesting for future work. In terms of improving result quality, to overcome the limits of our discrete sampling of periods, we could fine-tune loop periods and start times using a fast postprocess. Another improvement is for our optimization to anticipate subsequent spatiotemporal feathering when estimating temporal cost near misaligned features. Currently, we know that spatiotemporal feathering reduces visible artifacts, but our optimization does not solve for labels that will give the optimal feathered results. Lastly, we could incorporate optical flow measures in our spatiotemporal consistency term to ensure more consistent motion between neighboring pixels.

For new user-facing features, we are interested in automatic creation of mirror loops [16] in addition to ordinary loops. We would also like to use a design gallery [113] type of approach to let a user quickly select from an interesting set of automatically created results with different levels of dynamism or different static and looping regions.

Static image (L_0) Loop period p_x Start frame s_x Activation thres.

Figure 4.12: Progressive video loop results, showing the static image corresponding to the least-dynamic loop L_0, and the per-pixel looping parameters. All parameters are visualized using a colormap ranging from green through yellow to red as values increase, with light gray indicating static pixels. The activation parameter a_x in the rightmost column encodes the level of dynamism at which each pixel transitions from static to looping. Please refer to the accompanying material to see the progression of video loops.

CHAPTER 5

CONCLUSIONS

Despite of my initial dream of building a grand framework of theories and applications for my PhD thesis, it eventually turned out to be no more than a trivial concatenation of three of my papers. The only thing that I had done as the effort of making it look like a thesis was to try to summarize from these projects and provide a few ideas, realistic or not, in the domain of visual modeling and computing.

The first idea is to advocate *interactability* of visual modeling. First, Interactive visual modeling is extremely important and especially well suited for the widespread of touch screens. Second, the human nature has long evolved to be able to empathize with things we interact with. Interaction generates fun, inspires imagination, and most important, empowers users to drive and derive infinite possibilities from an interactable visual scene. Visual media computing is not to deliver the final result purely by a machine algorithm, but driven by the end users where computation serves merely as a way of transforming user intent into the output.

I then try to relate this user-driven content creation paradigm to the idea of *visual expression* – a topic that is least expected to appear in a thesis of scientific research. The term visual expression has two levels of meaning: (1) a designer, or model creator, to express his/her thoughts or view of the world through a certain form of visual format; and (2) such visual format allows a user to express one's internal states (mood, taste, preference, etc.) through his/her interaction with the scene. These ideas are not unfamiliar to an artist but may make an engineering or science major feel like "what?!". Nonetheless, after wandering between the realm of modern science and the art of the Chinese calligraphy, the preciseness of scientific logic and reasoning, and the beauty of ancient poems and essays by ancient

philosophers like Confucius and Zhuang Zhou, I am enticed to add a flavor of artisticness to the rigidness of science. In science we dive into the external universe and lose ourselves in the boundless pursuing of the infinite unknowns. In art, it is the inner being that is fulfilled and taken just as important as the outward. Therefore, visual computing should not be confined to how do we model the visual world but ask what purpose on earth does it serve.

REFERENCES

[1] J. T. Barron and J. Malik, "Color constancy, intrinsic images, and shape estimation," *ECCV*, 2012.

[2] P. Selinger, "Potrace: a polygon-based tracing algorithm," in *In http://potrace.sourceforge.net*, 2003.

[3] G. Lecot and B. Levy, "Ardeco: Automatic region detection and conversion," in *Eurographics Symposium on Rendering*, 2006.

[4] "Gradient art: Creation and vectorization," in *Image and Video-Based Artistic Stylisation*, ser. Computational Imaging and Vision, P. Rosin and J. Collomosse, Eds., 2013, vol. 42.

[5] J. Sun, L. Liang, F. Wen, and H.-Y. Shum, "Image vectorization using optimized gradient meshes," *ACM Trans. Graph.*, vol. 26, no. 3, July 2007. [Online]. Available: http://doi.acm.org/10.1145/1276377.1276391

[6] A. Orzan, A. Bousseau, H. Winnemöller, P. Barla, J. Thollot, and D. Salesin, "Diffusion curves: A vector representation for smooth-shaded images," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 92:1–92:8, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1360612.1360691

[7] T. Xia, B. Liao, and Y. Yu, "Patch-based image vectorization with automatic curvilinear feature alignment," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 115:1–115:10, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1618452.1618461

[8] Y.-K. Lai, S.-M. Hu, and R. R. Martin, "Automatic and topology-preserving gradient mesh generation for image vectorization," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 85:1–85:8, July 2009. [Online]. Available: http://doi.acm.org/10.1145/1531326.1531391

[9] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: image and video synthesis using graph cuts," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 277–286, July 2003.

[10] A. Agarwala, K. C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski, "Panoramic video textures," *ACM Trans. Graph.*, vol. 24, no. 3, July 2005.

[11] P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar, "Acquiring the reflectance field of a human face," in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. [Online]. Available: http://dx.doi.org/10.1145/344779.344855 pp. 145–156.

[12] P. N. Belhumeur and D. J. Kriegman, "What is the set of images of an object under all possible illumination conditions?" *IJCV*, 1998.

[13] R. Basri and D. Jacobs, "Lambertian reflectance and linear subspaces," *PAMI*, 2003.

[14] A. Schödl, R. Szeliski, D. H. Salesin, and I. Essa, "Video textures," in *SIGGRAPH Proceedings*, 2000, pp. 489–498.

[15] J. Tompkin, F. Pece, K. Subr, and J. Kautz, "Towards moment images: Automatic cinemagraphs," in *Proc. of the 8th European Conference on Visual Media Production (CVMP 2011)*, November 2011.

[16] N. Joshi, S. Mehta, S. Drucker, E. Stollnitz, H. Hoppe, M. Uyttendaele, and M. Cohen, "Cliplets: Juxtaposing still and dynamic imagery," *Proceedings of UIST*, 2012.

[17] V. Couture, M. Langer, and S. Roy, "Panoramic stereo video textures," *ICCV*, pp. 1251–1258, 2011.

[18] J. Beck and K. Burg, "Cinemagraphs," http://cinemagraphs.com/, 2012.

[19] C. T. Loop, "Smooth subdivision surfaces based on triangles," University of Utah, Department of Mathematics, 1987.

[20] P. Cavanagh, "Artists on science: scientists on art," *Nature*, pp. 301–307, 2005.

[21] H. Dee and P. Santos, "The perception and content of cast shadows: An interdisciplinary review," *Spatial Cognition and Computation*, vol. 11, pp. 226–253, 2011.

[22] M. F. Barnsley and L. P. Hurd, *Fractal image compression.* AK Peters, 1993.

[23] Z. Liao, H. Hoppe, D. Forsyth, and Y. Yu, "A subdivision-based representation for vector image editing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 11, pp. 1858–1867, 2012.

[24] M. Finch, J. Snyder, and H. Hoppe, "Freeform vector graphics with controlled thin-plate splines," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 166:1–166:10, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2070781.2024200

[25] R. D. Janssen and A. M. Vossepoel, "Adaptive vectorization of line drawing images," *Computer Vision and Image Understanding*, vol. 65, no. 1, pp. 38 – 56, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314296904841

[26] J. J. Zou and H. Yan, "Cartoon image vectorization based on shape subdivision," in *Computer Graphics International 2001. Proceedings*, 2001, pp. 225–231.

[27] X. Hilaire and K. Tombre, "Robust and accurate vectorization of line drawings," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 6, pp. 890–904, 2006.

[28] S.-H. Zhang, T. Chen, Y.-F. Zhang, S.-M. Hu, and R. Martin, "Vectorizing cartoon animations," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 4, pp. 618–629, 2009.

[29] D. DeCarlo and A. Santella, "Stylization and abstraction of photographs," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 769–776, July 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566650

[30] L. Demaret, N. Dyn, and A. Iske, "Image compression by linear splines over adaptive triangulations," *Signal Processing*, vol. 86, no. 7, pp. 1604 – 1616, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165168405002859

[31] S. Swaminarayan and L. Prasad, "Rapid automated polygonal image decomposition," in *Applied Imagery and Pattern Recognition Workshop, 2006. AIPR 2006. 35th IEEE*, 2006, pp. 28–28.

[32] P. Wesseling, *An Introduction to Multigrid Methods*. R.T. Edwards, Inc., 2004.

[33] K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi, "Volumetric modeling with diffusion surfaces," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 180:1–180:8, Dec. 2010. [Online]. Available: http://doi.acm.org/10.1145/1882261.1866202

[34] S. Jeschke, D. Cline, and P. Wonka, "Rendering surface details with diffusion curves," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 117:1–117:8, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1618452.1618463

[35] H. Hnaidi, E. Gurin, S. Akkouche, A. Peytavie, and E. Galin, "Feature based terrain generation using diffusion equation," *Computer Graphics Forum (Proceedings of Pacific Graphics)*, vol. 29, no. 7, pp. 2179–2186, 2010.

[36] H. Bezerra, E. Eisemann, D. DeCarlo, and J. Thollot, "Diffusion constraints for vector graphics," in *NPAR 2010: Proceedings of the 8th International Symposium on Non-photorealistic Animation and Rendering*. ACM Press, 2010. [Online]. Available: http://maverick.inria.fr/Publications/2010/BEDT10

[37] J. Powell, *A Thin Plate Spline Method for Mapping Curves Into Curves in Two Dimensions*, ser. Cambridge DAMTP. Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 1995. [Online]. Available: http://books.google.com/books?id=56x3QwAACAAJ

[38] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle, "Piecewise smooth surface reconstruction," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '94. New York, NY, USA: ACM, 1994. [Online]. Available: http://doi.acm.org/10.1145/192161.192233 pp. 295–302.

[39] P. Kovesi, "MATLAB and Octave functions for computer vision and image processing," http://www.csse.uwa.edu.au/∼pk/research/matlabfns/, 2010.

[40] C. Rother, V. Kolmogorov, and A. Blake, ""grabcut": Interactive foreground extraction using iterated graph cuts," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 309–314, Aug. 2004. [Online]. Available: http://doi.acm.org/10.1145/1015706.1015720

[41] C. Steger, "Subpixel-precise extraction of lines and edges," in *International Archives of Photogrammetry and Remote Sensing*, vol. XXXIII, part B3, 2000, pp. 141–156.

[42] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997. [Online]. Available: http://dx.doi.org/10.1145/258734.258849 pp. 209–216.

[43] w. D. C. Sivan Toledo and V. Rotkin, "Taucs:a library of sparse linear solvers. version 2.2," Tel-Aviv University, 2003.

[44] T. Igarashi, T. Moscovich, and J. F. Hughes, "As-rigid-as-possible shape manipulation," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1134–1141, July 2005. [Online]. Available: http://doi.acm.org/10.1145/1073204.1073323

[45] I. Guskov, W. Sweldens, and P. Schröder, "Multiresolution signal processing for meshes," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999. [Online]. Available: http://dx.doi.org/10.1145/311535.311577 pp. 325–334.

[46] P. J. Burt and E. H. Adelson, "The laplacian pyramid as a compact image code," *Communications, IEEE Transactions on*, vol. 31, no. 4, pp. 532–540, 1983.

[47] K. Fujiwara, "Eigenvalues of laplacians on a closed riemannian manifold and its nets," in *Proceedings of the American Mathematical Society*, 1995, pp. 2585–2594.

[48] A. Patney, M. S. Ebeida, and J. D. Owens, "Parallel view-dependent tessellation of catmull-clark subdivision surfaces," in *Proceedings of the Conference on High Performance Graphics 2009*, ser. HPG '09. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1572769.1572785 pp. 99–108.

[49] C. Eisenacher, Q. Meyer, and C. Loop, "Real-time view-dependent rendering of parametric surfaces," in *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ser. I3D '09. New York, NY, USA: ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1507149.1507172 pp. 137–143.

[50] NVidia, "Nvidia CUDA programming guide 2.0," http://developer.nvidia.com/object/cuda.html, 2008.

[51] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 313–318, July 2003. [Online]. Available: http://doi.acm.org/10.1145/882262.882269

[52] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen, "Interactive digital photomontage," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 294–302, 2004.

[53] Y. Ostrovsky, P. Cavanagh, and P. Sinha, "Perceiving illumination inconsistencies," *Perception*, vol. 34, pp. 1301–1314, 2005.

[54] P. Sinha, B. Balas, Y. Ostrovsky, and R. Russell, "Face recognition by humans: Nineteen results all computer vision researchers should know about," *Proceedings of the IEEE*, vol. 94, no. 11, pp. 1948–1962, 2006.

[55] B. Conway and M. Livingstone, "Perspectives on science and art," *Current Opinion in Neurobiology*, vol. 17, pp. 476–482, 2007.

[56] E. A. Khan, E. Reinhard, R. W. Fleming, and H. H. Bülthoff, "Image-based material editing," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 654–663, July 2006. [Online]. Available: http://doi.acm.org/10.1145/1141911.1141937

[57] K. Karsch, V. Hedau, D. Forsyth, and D. Hoiem, "Rendering synthetic objects into legacy photographs," in *ACM Trans. Graph (SIGGRAPH Asia)*, 2011, pp. 157:1–157:12.

[58] S. Nayar, K. Ikeuchi, and T. Kanade, "Determining shape and reflectance of hybrid surfaces by photometric sampling," *Robotics and Automation, IEEE Transactions on*, vol. 6, no. 4, pp. 418–431, 1990.

[59] H. P. A. Lensch, J. Kautz, M. Goesele, W. Heidrich, and H.-P. Seidel, "Image-based reconstruction of spatial appearance and geometric detail," *ACM Trans. Graph.*, vol. 22, no. 2, pp. 234–257, Apr. 2003. [Online]. Available: http://doi.acm.org/10.1145/636886.636891

[60] A. S. Georghiades, "Recovering 3-d shape and reflectance from a small number of photographs," in *Proceedings of the 14th Eurographics Workshop on Rendering*, ser. EGRW '03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=882404.882438 pp. 230–240.

[61] S. Mallick, T. Zickler, D. Kriegman, and P. Belhumeur, "Beyond lambert: reconstructing specular surfaces using color," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, 2005, pp. 619–626 vol. 2.

[62] A. Hertzmann and S. Seitz, "Example-based photometric stereo: shape reconstruction with general, varying brdfs," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 8, pp. 1254–1264, 2005.

[63] D. Goldman, B. Curless, A. Hertzmann, and S. Seitz, "Shape and spatially-varying brdfs from photometric stereo," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 6, pp. 1060–1071, 2010.

[64] H. Barrow and J. Tenenbaum, "Recovering intrinsic scene characteristics from images," in *Comp. Vision Sys.*, 1978, pp. 3–26.

[65] E. Land and J. McCann, "Lightness and retinex theory," in *J. Opt. Soc. Am.*, vol. 61, 1971, pp. 1–11.

[66] B. K. P. Horn, "Determining lightness from an image," *Computer Vision, Graphics and Image Processing*, vol. 3, pp. 277–299, 1974.

[67] A. Blake, "Boundary conditions for lightness computation in mondrian world," *Computer Vision, Graphics and Image Processing*, vol. 32, pp. 314–327, 1985.

[68] G. Brelstaff and A. Blake, "Computing lightness," *Pattern Recognition Letters*, vol. 5, no. 2, pp. 129–38, 1987.

[69] B. Funt, M. Drew, and M. Brockington, "Recovering shading from color images," in *ECCV*, 1992.

[70] Y. Weiss, "Deriving intrinsic images from image sequences," in *International Conference on Computer Vision*, 2001, pp. 68–75.

[71] M. Tappen, W. Freeman, and E. Adelson, "Recovering intrinsic images from a single image," *PAMI*, 2006.

[72] A. Bousseau, S. Paris, and F. Durand, "User-assisted intrinsic images," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 130:1–130:10, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1618452.1618476

[73] L. Shen, P. Tan, and S. Lin, "Intrinsic image decomposition with non-local texture cues," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008, pp. 1–7.

[74] L. Shen and C. Yeo, "Intrinsic images decomposition using a local and global sparse representation of reflectance," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 2011, pp. 697–704.

[75] Z. Liao, J. Rock, Y. Wang, and D. Forsyth, "Non-parametric filtering for geometric detail extraction and material representation," in *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '13. Washington, DC, USA: IEEE Computer Society, 2013. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2013.129 pp. 963–970.

[76] R. Ramamoorthi and P. Hanrahan, "An efficient representation for irradiance environment maps," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001. [Online]. Available: http://doi.acm.org/10.1145/383259.383317 pp. 497–500.

[77] B. K. Horn, "Shape from shading: A method for obtaining the shape of a smooth opaque object from one view," Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep., 1970.

[78] K. Ikeuchi and B. K. P. Horn, "Numerical shape from shading and occluding boundaries." *Artificial Intelligence*, vol. 17, no. 1-3, pp. 141–184, 1981.

[79] M. J. Brooks and B. K. P. Horn, "Shape and source from shading," in *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'85. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985. [Online]. Available: http://dl.acm.org/citation.cfm?id=1623611.1623664 pp. 932–936.

[80] R. Frankot and R. Chellappa, "A method for enforcing integrability in shape from shading algorithms," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 10, no. 4, pp. 439–451, 1988.

[81] Y. G. Leclerc and A. Bobick, "The direct computation of height from shading," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, 1991, pp. 552–558.

[82] D. Forsyth, "Variable-source shading analysis," *IJCV'11*, vol. 91, 2011.

[83] T.-P. Wu, J. Sun, C.-K. Tang, and H.-Y. Shum, "Interactive normal reconstruction from a single image," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 119:1–119:9, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1409060.1409072

[84] L. Zhang, G. Dugas-Phocion, J.-S. Samson, and S. Seitz, "Single view modeling of free-form scenes," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, 2001, pp. I–990–I–997 vol.1.

[85] T.-P. Wu, C.-K. Tang, M. S. Brown, and H.-Y. Shum, "Shapepalettes: Interactive normal transfer via sketching," *ACM Trans. Graph.*, vol. 26, no. 3, July 2007. [Online]. Available: http://doi.acm.org/10.1145/1276377.1276432

[86] R. Zhang, P.-S. Tsai, J. Cryer, and M. Shah, "Shape-from-shading: a survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 8, pp. 690–706, 1999.

[87] J.-D. Durou, M. Falcone, and M. Sagona, "Numerical methods for shape-from-shading: A new survey with benchmarks," *Comput. Vis. Image Underst.*, vol. 109, no. 1, pp. 22–43, 2008.

[88] S. F. Johnston, "Lumo: illumination for cel animation," in *NPAR '02*, 2002.

[89] M. Prasad and A. Fitzgibbon, "Single view reconstruction of curved surfaces," in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, ser. CVPR '06, 2006, pp. 1345–1354.

[90] R. Grosse, M. K. Johnson, E. H. Adelson, and W. T. Freeman, "Ground-truth dataset and baseline evaluations for intrinsic image algorithms," in *International Conference on Computer Vision*, 2009, pp. 2335–2342.

[91] J.-F. Lalonde, D. Hoiem, A. Efros, C. Rother, J. Winn, and A. Criminisi, "Photo clip art," *ACM Trans. Graph*, 2007.

[92] T. Chen, M.-M. Cheng, P. Tan, A. Shamir, and S.-M. Hu, "Sketch2photo: internet image montage," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 124:1–124:10, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1618452.1618470

[93] Y. Yu, P. Debevec, J. Malik, and T. Hawkins, "Inverse global illumination: Recovering reflectance models of real scenes from photographs," in *Siggraph99, Annual Conference Series*, A. Rockwood, Ed. Los Angeles: Addison Wesley Longman, 1999, pp. 215–224.

[94] J. Beck and S. Prazdny, "Highlights and the perception of glossiness," *Perception and Psychophysics*, 1981.

[95] J. Berzhanskaya, G. Swaminathan, J. Beck, and E. Mingolla, "Remote effects of highlights on gloss perception," *Perception*, 2005.

[96] D. Akers, F. Losasso, J. Klingner, M. Agrawala, J. Rick, and P. Hanrahan, "Conveying shape and features with image-based relighting," in *VIS '03*, 2003.

[97] R. Fattal, M. Agrawala, and S. Rusinkiewicz, "Multiscale shape and detail enhancement from multi-light image collections," in *SIGGRAPH '07*, 2007.

[98] V. Ramachandran and W. Hirstein, "The science of art: A neurological theory of aesthetic experience," *Journal of Consciousness Studies*, vol. 6, no. 6-7, pp. 15–51, 1999.

[99] A. Georghiades, P. Belhumeur, and D. Kriegman, "From few to many: illumination cone models for face recognition under variable lighting and pose," *PAMI*, 2001.

[100] V. Hedau, D. Hoiem, and D. A. Forsyth, "Recovering the spatial layout of cluttered rooms." in *ICCV*. IEEE, 2009, pp. 1849–1856.

[101] P. Debevec, "Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '98. New York, NY, USA: ACM, 1998. [Online]. Available: http://doi.acm.org/10.1145/280814.280864 pp. 189–198.

[102] M. Cohen and R. Szeliski, "The moment camera," *IEEE Computer*, vol. 39, no. 8, Aug 2006.

[103] J. Bai, A. Agarwala, M. Agrawala, and R. Ramamoorthi, "Selectively de-animating video," *ACM Trans. Graph.*, vol. 31, no. 4, 2012.

[104] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 11, pp. 1222–1239, 2001.

[105] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '01. New York, NY, USA: ACM, 2001. [Online]. Available: http://doi.acm.org/10.1145/383259.383296 pp. 341–346.

[106] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. on Pattern Anal. Mach. Intell.*, vol. 23, no. 11, 2001.

[107] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph cuts?" *IEEE Trans. on Pattern Anal. Mach. Intell.*, vol. 26, no. 2, 2004.

[108] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen, "Interactive video cutout," *ACM Trans. Graph.*, vol. 24, no. 3, 2005.

[109] B. K. P. Horn and B. G. Schunk, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185–203, 1981.

[110] P. J. Burt and E. H. Adelson, "A multiresolution spline with application to image mosaics," *ACM Trans. Graph.*, vol. 2, no. 4, October 1983.

[111] D. Mahajan, F.-C. Huang, W. Matusik, R. Ramamoorthi, and P. Belhumeur, "Moving gradients: A path-based method for plausible image interpolation," *ACM Trans. Graph.*, vol. 28, no. 3, p. 42, July 2009.

[112] T. S. Cho, N. Joshi, C. L. Zitnick, S. B. Kang, R. Szeliski, and W. T. Freeman, "A content-aware image prior," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[113] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber, "Design galleries: A general approach to setting parameters for computer graphics and animation," *ACM SIGGRAPH Proceedings*, 1997.