Foreground Segmentation in Images and Video: Methods, Systems and Applications

Jue Wang

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

University of Washington

2007

Program Authorized to Offer Degree: Electrical Engineering

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Jue Wang

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Chair of the Supervisory Committee:

Eve A. Riskin

Reading Committee:

Eve A. Riskin

Michael F. Cohen

Ming-Ting Sun

Date:

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with “fair use” as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to Proquest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted “the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform.”

Signature_____

Date_____

University of Washington

Abstract

Foreground Segmentation in Images and Video:
Methods, Systems and Applications

Jue Wang

Chair of the Supervisory Committee:
Professor Eve A. Riskin
Electrical Engineering

Separating foreground objects from natural images and video plays an important role in image and video editing tasks. Despite extensive study in the last two decades, this problem still remains challenging. In particular, extracting a foreground object from the background in a static image involves determining both full and partial pixel coverage, also known as extracting a matte, which is a severely under-constrained problem. Segmenting spatio-temporal video objects from a video sequence is even harder since extracted foregrounds on adjacent frames must be both spatially and temporally coherent. Previous approaches for foreground extraction usually require a large amount of user input and still suffer from inaccurate results and low computational efficiency.

This thesis demonstrates efficient foreground extraction methods and systems by combining advanced computational algorithms with novel user interfaces. Our systems are capable of extracting high quality foreground objects from images and video with a limited amount of user input such as a few paint strokes of the mouse. We also demonstrate a variety of applications with the extracted foreground objects.

Specifically, for still images, we develop a novel *Robust Matting* algorithm, which is capable of generating high quality alpha mattes for complex images in a robust way. Centered around this algorithm we build *Soft Scissors*, the first interactive tool for extracting high quality mattes in real-time. We also propose a *compositional matting* algorithm which combines matting and compositing into a single optimization process. Quantitative and objective evaluations demonstrate that these

systems outperform previous approaches in both accuracy and efficiency.

For motion pictures, we propose an interactive *Video Cutout* system which extracts spatio-temporal coherent foreground objects from video sequences through a novel 3D painting user interface. As an application we develop a *Video Tooning* system which can stylize the extracted video objects with a variety of cartoon styles. We also propose a *cartoon animation filter* which automatically exaggerates and stylizes the motion of the extracted video objects.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Table Number Page

ACKNOWLEDGMENTS

My sincerest gratitude goes first to my research advisor Michael Cohen. Collaborating with him in the past few years has been an invaluable experience. He discovered my research potential even before we met in person. He brought me into University of Washington, mentored me, and gave me freedom to do my own research. He not only constantly gives me insightful comments and guidance on my projects, but also is willing to spend a significant amount of time to dig into technical and programming details. He is always kind, encouraging and supportive. He is the type of advisor that every graduate student wants.

My sincerest gratitude to my school advisors Eve Riskin and Richard Ladner. It is a great pleasure to be a member in their Data Compression group. Serving on my advisory committee, they have given me invaluable comments and suggestions both on my study and my research. Special thanks to Eve for her timely help on my thesis and presentations, school-related issues, and her nominations and recommendations for my Microsoft Research fellowship and Yang Research Award.

Many thanks to my collaborators at the University of Washington and Microsoft Research. They are Provin Bhat, Alex Colburn, Maneesh Agrawala, Steven Drucker, Bo Thiesson, Yingqing Xu and Harry Shum. Special thanks to Alex Colburn for his tremendous but un-credited efforts on my early SIGGRAPH submissions. Thanks to Colin Zheng and Yung-Yu Chuang for sharing their thoughts, code and programs. Thanks to Keith Grochow and Mira Dontcheva for their help on my projects and paper submissions. Thanks to Professor Steven Seitz and Ming-Ting Sun for serving on my thesis committee. Thanks to Professor Brian Curless, David Salesin, Linda Shapiro who have given me thoughtful comments on my research. Thanks to all of my friends in Grail, Paul Allen Center and UW, I enjoyed the time that I spent with everyone of you.

Last in this page but first in my heart, thanks to Mom and Dad for their perpetual love and support. Thanks to my wife Qi Miao for wonderful memories we share together.

DEDICATION

To my wife Qi Miao for her love and support over the years.

Chapter 1

INTRODUCTION

1.1 Computational Photography and Video

The very first forms of photography date back to the early 19th century when America's history was captured on film during the Civil War. Since then many advancements have been made in the world of photography. One of the newest advancements in the field was the birth of digital photography. In 1963 Theodore Maiman at Stanford University invented a videodisk camera that could take a photograph and store the image on a disk for several minutes. This would be the precursor to digital photography. With this new form of taking photographs digitally and storing the image to a disk, photography would become less time consuming and a whole new chapter would open to future photographers. In the mid 70s, Kodak began to work on filmless technologies. In the mid 80s with the release of the compact disc, digital technology continued to increase. By 1990 the first digital camera hit shelves for commercial sales. These would be the first steps into a new digital world, and pave the way for many things to come.

One of the most obvious advantage of recording photographs as digital signals is that they can be easily manipulated by applying computational algorithms. Although cameras have been vastly improved over the last few decades, they are still incomparable to human eyes, which have much higher resolution and dynamic range than the most advanced cameras. People often postprocess digital photographs, for example, to adjust hue, saturations, brightness, contrast, sharpness and blur, to create more vivid representations of what the human eyes have perceived. Most of these low-level postprocessing tasks have already been implemented on the camera and can be automatically applied on-the-fly when a photograph is shot, resulting in a vivid, high quality picture.

Although the quality of photographs coming out of digital cameras has been largely improved, people are still constantly disappointed by photographs, as they don't seen to record what we believe

Figure 1.1: A digital camera often captures an undesired image as the one shown on the left. Human perception, on the contrary, can selectively perceive the right image as shown on the right.

we have seen. “The camera never lies”. It objectively records the light coming from the scene. In contrast, our higher cognitive functions constantly mediate our perceptions so that what we get is decidedly not what we perceive. For instance, almost everyone has the experience of shooting undesirable portrait photographs where the person’s eyes are closed or half-closed, although we almost never consciously perceive an eye blink (see Figure 1.1). The desired photographs should be able to more accurately convey our subjective impressions-or go beyond them, providing visualizations or a greater degree of artistic expression.

Computational photography and video are the means to achieve more perfect photography. The convergence of computer vision, graphics, and photography, has been facilitated by research breakthroughs in 2D image analysis/synthesis coupled with the growth of digital photography as a practical and artistic medium. In computational photography and video we apply computational, statistical and optimization methods to images and video for a variety of consumer digital photography and video applications, to overcome the limitations of the traditional camera and to produce a richer, more vivid, perhaps more perceptually meaningful representation of our visual world.

1.2 Foreground Extraction

Computational photography and video are a broad research field, and research topics range from advanced camera systems to intelligent image manipulation software. This thesis work focuses on one important problem in computational photography and video: *foreground extraction*.

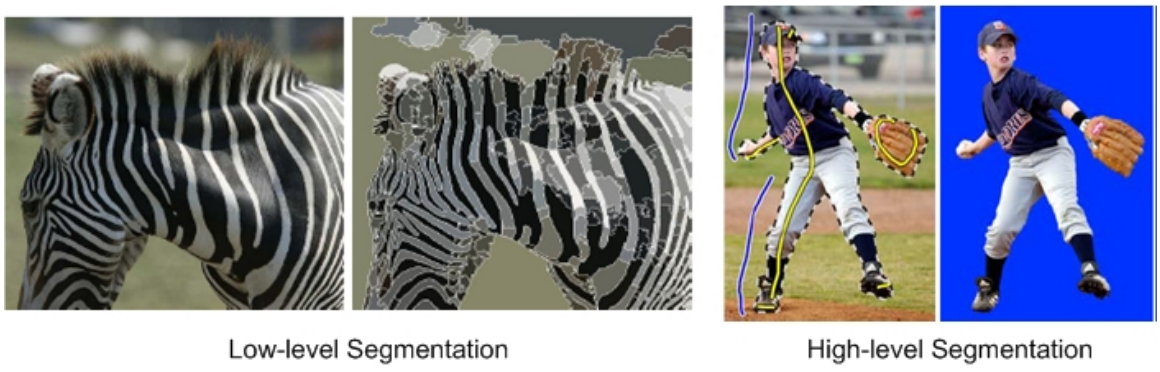
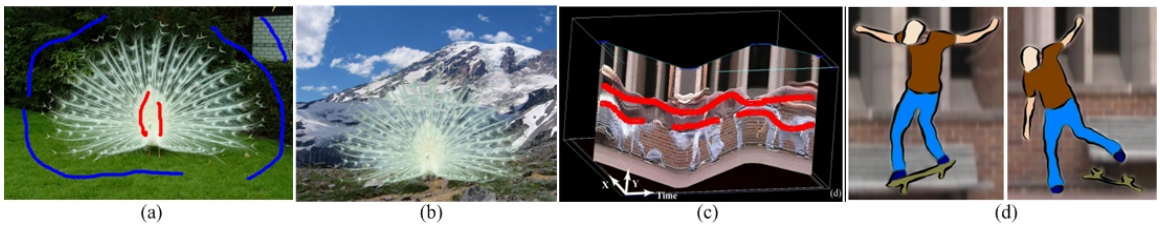


Figure 1.2: Low-level segmentation merges pixels into homogeneous regions, while high-level segmentation extracts semantically meaningful objects.

Figure 1.3: In the image matting problem the observed image I is modelled as a convex combination of a foreground image F and background image B. The interpolation coefficient α is called the *matte*.

Image segmentation is a fundamental problem in digital image processing and computer vision, and has been extensively studied for many decades. Previous approaches in this field can be classified into two categories according to their goals. As shown in Figure 1.2, low-level segmentation groups pixels into small homogenous regions which can be used for further analysis, while high-level segmentation aims at identifying semantically meaningful regions, allowing the user to query or edit images on the object level. This dissertation researches the problem of accurate foreground extraction for rendering purposes, thus it belongs to the high-level segmentation category.

Separating a foreground object from the background in a static image involves determining both full and partial pixel coverage, also known as extracting a matte. Mathematically, the observed image I_z ($z = (x, y)$) is modelled as a linear combination of foreground image F_z and background



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

Figure 1.4: (a) and (b): by accurately segmenting the peacock from the image we can seamlessly compose it onto a new background image. (c) and (d): by segmenting the skateboarder from the input video sequence we are able to coherently stylize the foreground object.

image B_z by an alpha map (see Figure 1.3):

$$I_z = \alpha_z F_z + (1 - \alpha_z) B_z \tag{1.1}$$

where α_z can be any value in [0,1]. If $\alpha_z = 1$ or $\alpha_z = 0$, we call pixel z *definite foreground* or *definite background*. Otherwise if $0 < \alpha_z < 1$, we call pixel z *mixed*. Although the majority of pixels belong to either definite foreground or definite background, accurately estimating alpha values for real mixed pixels is essential for fully separating the foreground from the background.

For natural images, all three values α, F and B need to be estimated for every pixel. The observation we have for a pixel are the three dimensional color vector $\underline{I}_z$, and the unknown variables are three dimensional color vector F_z and B_z, and one dimension alpha value α_z. Matting is thus inherently an under-constrained problem, since we need to solve for 7 unknown variables from 3 known values. Previous approaches and the methods proposed in this dissertation rely on user guidance combined with prior assumptions on image statistics to obtain good estimations of αs(the matte) and Fs. Once these values are estimated, we can seamlessly compose the foreground onto a new background B', as shown in Figure 1.4.

Almost all digital video processing applications can benefit from methods to identify coherent objects in the spatio-temporal volume of pixel data. For example, which pixels are part of a person walking across the street? An important technology in video processing is thus segmenting spatio-temporal objects. However, video object segmentation is intrinsically a harder problem than image segmentation due to its nature of spatio-temporal scene coverage. First, it requires accurate foreground pixel identification on each individual frame; second, it requires the segmentation on

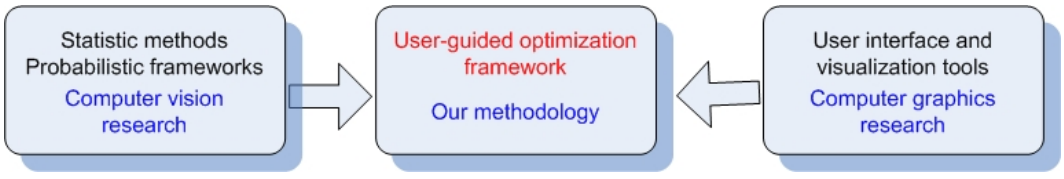


Figure 1.5: The general methodology of the systems we developed.

adjacent frames to be temporally coherent to avoid jittering artifacts.

Once a video object is successfully extracted, the user can apply a variety of applications on it. The most obvious one is to insert the object into a new background footage to create a novel composite as many Hollywood movies do. For video stylization, the extracted spatio-temporal objects provide basic rendering primitives, which can be rendered in various cartoon styles, as shown in Figure 1.4. For communication, the segmented objects make object-based compression and transmission possible.

1.3 The Methodology

The problem of extracting foreground objects from images and video belongs to the more general problem of image segmentation, which has been extensively studied as a classic problem in both the computer vision and computer graphics communities. However, the general methodology employed in this dissertation is significantly different from those employed in traditional approaches.

Most of the efforts made in traditional computer vision research for image segmentation have been focused on fully automatic methods. Since the definition of “foreground objects” is subjective, these approaches usually employ machine learning techniques to gain prior knowledge about the target object. To achieve this, a certain amount of training data must be collected in advance and used for the training of specific models of the target objects, which significantly limits the application range of these approaches. Furthermore, even the most advanced machine learning techniques may make errors and cannot perfectly identify the target object in a test image, thus the segmentation results usually contain noticeable errors, which is unacceptable in many applications such as re-compositing the foreground onto a new background.

On the other hand, segmentation approaches developed in computer graphics research heavily

rely on the user's assistant to achieve accurate segmentation. The representative techniques are "Magic Wand" [66] and "Intelligent Scissor" [101] in Adobe's Photoshop. These approaches leverage the user's efforts to roughly specify the foreground object boundary and then employ relatively simple optimization techniques to refine it. Thus it is a tedious process to get an accurate segmentation for complex objects. Furthermore, these approaches cannot deal with transparent or fuzzy objects.

In this dissertation we develop a new set of algorithms to efficiently extract foreground objects from arbitrary images and video sequences, which combines advantages of both previous computer vision and computer graphics approaches. Specifically, as shown in Figure 1.5, our research adopts statistical methods and probabilistic frameworks developed in computer vision research for accurate segmentation, and user interface and visualization tools from computer graphics research for efficient user guidance. By combining these techniques together, we formulate the problem of foreground extraction in a user-guided optimization framework, which can be solved efficiently and generate high quality results.

Chapter 2

ROBUST MATTING FOR STILL IMAGES

2.1 Interactive Matting

As we described in the previous chapter, image matting refers to the problem of estimating an opacity (alpha value), α_z, and foreground and background colors, F_z and B_z, for each pixel z in the input image I. This allows the foreground object to be fully extracted from the input image and to be seamlessly composed onto a new background using the estimated Fs and αs. As shown in Equation 1.1, for natural images, all three values α_z, F_z and B_z need to be estimated for every pixel z given only one observation I_z; thus it is inherently an underconstrained problem.

Without any user input, it is obvious that valid solutions to the equation are infinite. For instance, we can set all α_zs to be 1 and all F_zs to be identical to I_zs, which means that the whole image is fully occupied by the foreground. This of course is not always consistent with what a human being perceives from the image. To properly extract semantically meaningful foreground objects, almost all matting approaches start by having the user segment the input image into three regions: definitely foreground R_f, definitely background R_b and unknown R_u. This pixel map is often referred to as a *trimap*. The problem is thus reduced to estimating F, B and α for pixels in the unknown region based on information in the known foreground and background regions. Some examples of a trimap are shown in Figure 2.1. Some recently proposed approaches allow the user to specify a few foreground and background scribbles to extract a matte. This intrinsically defines a very coarse trimap by marking a large number of pixels as unknowns.

One of the important factors affecting the performance of a matting algorithm is how accurate the trimap is. Ideally, the unknown region in the trimap should only cover pixels whose actual alpha values are not 0 or 1. In other words, the unknown region in the trimap should be as thin as possible to achieve the best possible matting results. This is somewhat straightforward since the more accurate the trimap is, the less unknown variables we need to estimate and the more known foreground and background samples we can use. Intrinsically, a more accurate trimap defines a

Figure 2.1: (a). Input image. (b). An accurate trimap. (c). Estimated matte given trimap (b) using our robust matting algorithm. (d). A less accurate trimap. (e) Estimated matte given trimap (b) using robust matting. Note that matte (e) is less accurate than matte (c) due to different trimaps.

Figure 2.2: Specifying an accurate trimap (right) for the input spider web image (left) is a tedious process for the user.

narrower solution space where the optimized solution is easier to achieve.

However, accurately specifying a trimap requires a significant amount of user efforts and is often undesirable in practice, especially for objects with large semi-transparent regions or holes. An example is shown in Figure 2.2, where accurately specifying a trimap for the spider web requires extensive user efforts. To achieve a good trade-off between the accuracy of the matte and the amount of the user efforts required, newly developed algorithms such as the Robust Matting approach proposed in this chapter employ advanced statistical and optimization methods to achieve robust matte estimation.

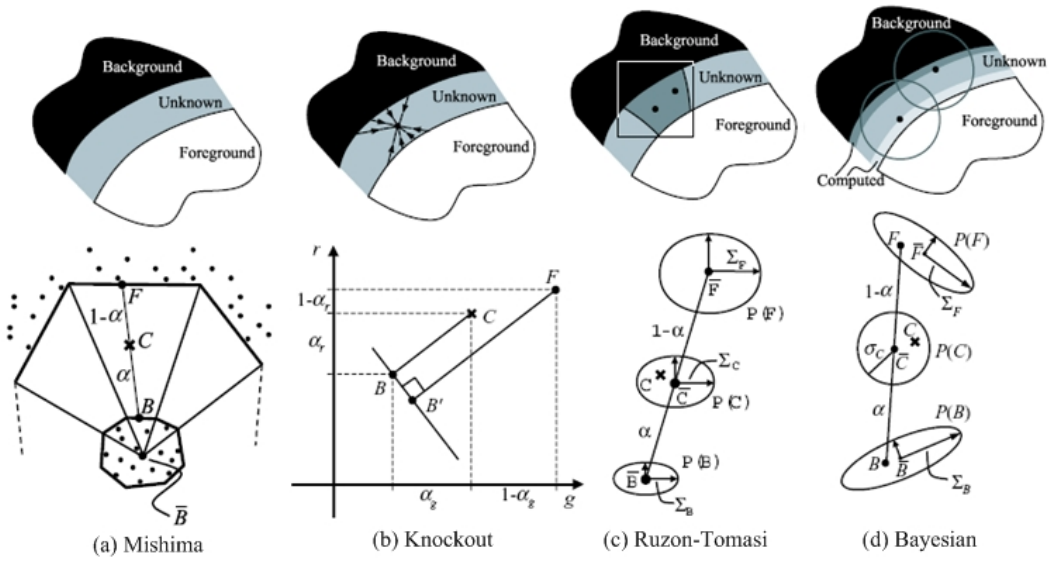


Figure 2.3: Summary of a few previous sampling-based matting algorithms. Diagrams are from [23].

2.2 Previous Approaches

Although the matting problem is ill-posed, the strong correlations between nearby image pixels can be leveraged to alleviate the difficulties. Intuitively, neighboring pixels that have similar colors often have similar alpha values. Roughly speaking, previous matting approaches can be classified into two categories based on their ways of using natural image statistics: sampling-based approaches and propagation-based ones.

2.2.1 Sampling-based Approaches

Sampling-based methods assume that the foreground and background colors of an unknown pixel can be explicitly estimated by looking at nearby user-specified ones. Once the foreground and background colors are estimated, the alpha value is left as the single unknown and thus is trivial to solve. This method was first used in early *blue screen matting* approaches, which is a technique commonly used in the film industry. In the blue screen matting setup, the subject is photographed against a constant-colored background, meaning that the background color is known and only the foreground colors and the alphas need to be estimated. Mishima [99] developed a blue screen matting tech-

nique based on representative foreground and background samples. As shown in Figure 2.3(a) It starts with two identical polyhedral (triangular mesh) approximations of a sphere in RGB color space centered at the average value B of the background samples. The vertices of one of the polyhedra (the background polyhedron) are then repositioned by moving them along lines radiating from the center until the polyhedron is as small as possible while still containing all the background samples. The vertices of the other polyhedron (the foreground polyhedron) are similarly adjusted to give the largest possible polyhedron that contains no foreground pixels from the sample provided. Given a new composite color C, then, Mishima casts a ray from B through C and defines the intersections with the background and foreground polyhedra to be B and F, respectively. The fractional position of C along the line segment BF is α.

Following this idea, the KnockOut system [33] extrapolates known foreground and background colors into the unknown region and estimates αs according to them. In particular, given a point in the unknown region, the foreground F is calculated as a weighted sum of the pixels on the perimeter of the known foreground region. The weight for the nearest known pixel is set to 1, and this weight tapers linearly with distance, reaching 0 for pixels that are twice as distant as the nearest pixel. The same procedure is used for initially estimating the background B based on nearby known background pixels, which is refined later by examining the geometrical relationships of F, B and the observed color C in color space. The alpha value is then computed according to the estimated F and B, as shown in Figure 2.3(b).

Ruzon and Tomasi [111] were the first to take a probabilistic view of the problem. First, they partition the unknown boundary region into sub-regions. For each sub-region, they construct a box that encompasses the sub-region and includes some of the nearby known foreground and background regions (see Figure 2.3(c)). The encompassed foreground and background pixels are then treated as samples from distributions $P(F)$ and $P(B)$, respectively, in color space. The foreground pixels are split into coherent clusters, and un-oriented Gaussians (i.e., Gaussians that are axis-aligned in color space) are fit to each cluster, each with mean F and diagonal covariance matrix Σ_F. In the end, the foreground distribution is treated as a mixture of Gaussians. The same procedure is performed on the background pixels yielding Gaussians, each with mean B and covariance Σ_B, and then every foreground cluster is paired with every background cluster. After building this network of paired Gaussians, they treat the observed color C as coming from an intermediate distribution

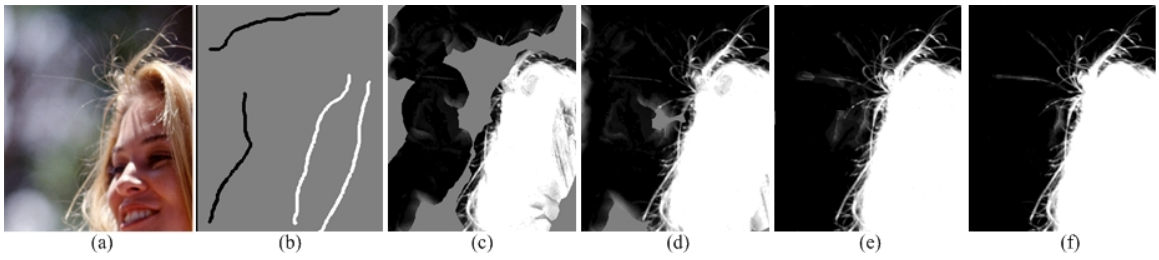


Figure 2.4: An example of iterative belief propagation matting. (a). Original image. (b). Initial user input. (c-f). Intermediate results after 3, 6, 9 and 15 iterations.

$P(C)$, somewhere between the foreground and background distributions. The intermediate distribution is also defined to be a sum of Gaussians, where each Gaussian is centered at a distinct mean value C located fractionally (according to a given alpha) along a line between the mean of each foreground and background cluster pair with fractionally interpolated covariance Σ_C, as depicted in Figure 2.3(d). The optimal alpha is the one that yields an intermediate distribution for which the observed color has maximum probability.

This approach has been further improved by the Bayesian matting system [23]. Similar to Ruzon and Tomasi's method, Bayesian matting also solves the problem by building foreground and background probability distributions from a given neighborhood. However, it uses a continuously sliding window for neighborhood definitions, marches inward from the foreground and background regions, and utilizes nearby computed F, B, and α values (in addition to these values from "known" regions) in constructing oriented Gaussian distributions, as illustrated in Figure 2.3(e). Furthermore, this approach formulates the problem of computing matte parameters in a well-defined Bayesian framework and solves it using the maximum a posteriori (MAP) technique, where the optimal α, F and B are estimated simultaneously.

Given the fact that an accurate trimap is tedious and sometimes impossible to specify, we have proposed an iterative matting system [134] which employs more complicated sampling and optimization methods to try to solve a matte directly from a few user specified scribbles instead of a carefully specified trimap. An example is shown in Figure 2.4. In this approach pixels are divided into two groups: group U_c includes all pixels whose alpha values have been specified by the user or estimated in previous iterations; and group U_n includes all other pixels left unconsidered. Initially

pixels the user marked are in U_c and all others are in U_n. The approach proceeds iteratively. In each iteration, the pixels in U_n which are nearby (within 15 pixels) to ones in U_c are added to U_c, and (F, B, α, u) are estimated or re-estimated for each pixel in the expanded set (u is the uncertainty of the pixel). The algorithm stops when U_n is null and the uncertainty for the whole image (sum of uncertainties of pixels) cannot be reduced any further. Thus, the algorithm works in a front-propagation fashion: the estimated alpha map is propagated out from user marked pixels to rest of the image. The U_c region is modelled as a Markov Random Field (MRF), and a belief propagation algorithm is used to estimate the matte. In the MRF construction, the system uses foreground and background color samples to set the data cost for an unknown pixel, and uses color difference betweeen neighboring pixels to set the link cost. The whole matte thus is estimated in an iterative fashion. This approach has been further improved by the easy matting system [53]. In these approaches since the user input is very sparse, global sampling methods are proposed to assist the local sampling procedure to generate enough color samples for unknown pixels.

We will demonstrate later that all these approaches suffer from inaccurate color estimation in complicated cases.

2.2.2 Propagation-based Approaches

Propagation-based methods do not explicitly estimate foreground and background colors; instead, they assume foreground and background colors are locally smooth, i.e., are constant or can be fit by a linear model. In this way foreground and background colors can be systematically eliminated from the optimization process and the matte can be solved in a closed form.

The Poisson matting algorithm [120] assumes the foreground and background colors are smooth in a narrow band of unknown pixels; thus the gradient of the matte matches with the gradient of the image, and can be calculated by solving Poisson equations. Specifically, Poisson matting consists of two steps. First, an approximate gradient field of matte is computed from the input image as

$$\nabla\alpha \approx \frac{1}{F-B}\nabla I \tag{2.1}$$

Second, to recover the matte in the unknown region Ω given an approximate $(F-B)$ and image gradient ∇I, a set of Poisson equations are defined as

$$\triangle\alpha = div(\frac{\triangle I}{F-B}) \tag{2.2}$$

with boundary conditions

$$\widehat{\alpha}_z|_{\partial\Omega} = \begin{cases} 1 & : \quad z \in \Omega_F \\ 0 & : \quad z \in \Omega_B \end{cases} \tag{2.3}$$

As we can see, estimating an accurate $F - B$ is critical to obtain the desired alpha matte by solving these equations. However, it is difficult to achieve for complex foreground and background patterns. When global Poisson matting fails to produce high quality mattes due to a complex background, local Poisson matting is employed in the system to manipulate a continuous gradient field in a local region. Local Poisson matting brings human interaction into the loop. In most cases, image gradients caused by foreground and background colors are visually distinguishable locally. Knowledge from the user can be effectively integrated into Poisson matting by using a set of tools that operate on the gradient field of the matte. However, local manual adjustments are very time-consuming for the user.

A similar method based on Random Walk is proposed in [52]. Given a user-supplied trimap of the pixels, this approach sets a value at each pixel in the unknown region as the probability that a random walker starting from this location will reach a pixel in the foreground before striking a pixel in the background, when biased to avoid crossing the foreground boundary. Despite the fact that these probabilities may seem prohibitively expensive to compute, this approach shows that they may be calculated exactly by solving a single system of linear equations. To set the edge weights between neighboring pixels, it applies a manifold learning technique to project a pixel's RGB color into a more appropriate feature space. It also shows that although a sparse set of linear equations may be efficiently solved with a variety of conventional techniques, several aspects of the problem formulation allow for a particularly efficient solution via a graphics processor unit.

The recently proposed closed-form matting [81] approach assumes foreground and background colors are smooth and can be fitted with linear models in small (typically 3×3) local windows, which leads to a quadratic cost function in α and can be minimized globally. Figure 2.5 shows an example to justify this assumption. It has been approved that under this assumption, alpha values in a small window w can be expressed as

$$\alpha_i = \sum_c a^c I_i^c + b, \forall i \in w, \tag{2.4}$$

where c refers to color channels, and a^c and b are constants in the window. The matting cost function

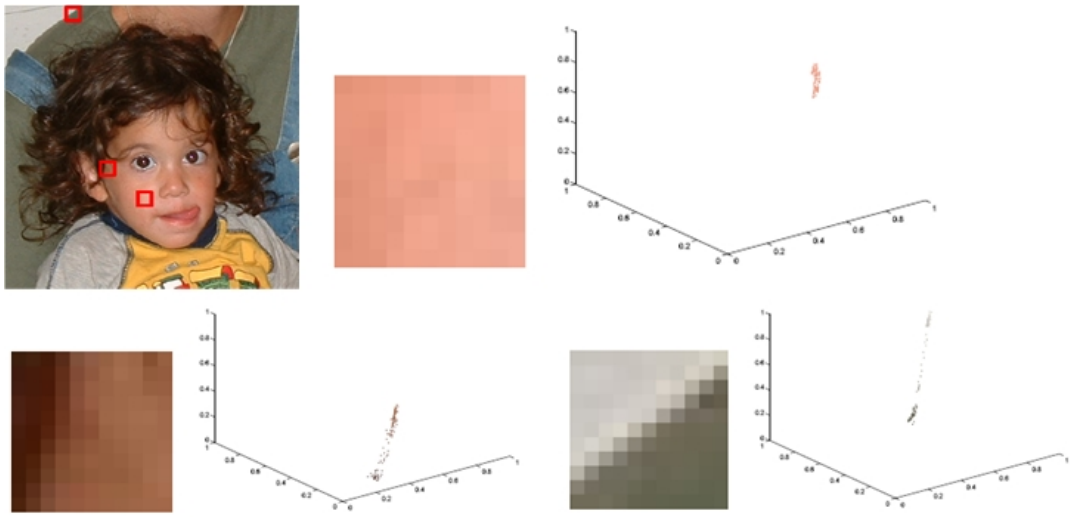


Figure 2.5: Local patches selected from a real image and the RGB plots of their color distributions, which can be approximated with linear models [81].

is then defined as

$$J(\alpha, a, b) = \sum_{j \in I} \left(\sum_{i \in w_j} \left(\alpha_i - \sum_c a_j^c I_i^c - b_j \right)^2 + \epsilon \sum_c a_j^{c2} \right). \tag{2.5}$$

It has been further shown that a^c and b can be eliminated from the cost function, yielding a quadratic cost in the α alone:

$$J(\alpha) = \alpha^T L \alpha, \tag{2.6}$$

where L is an $N \times N$ matrix, whose (i, j)-th element is:

$$\sum_{k|(i,j) \in w_k} \left(\delta_{ij} - \frac{1}{|w_k|} \left(1 + (I_i - \mu_k)(\Sigma_k + \frac{\varepsilon}{|w_k|} I_3)^{-1} (I_j - \mu_k) \right) \right), \tag{2.7}$$

where Σ_k is a 3×3 covariance matrix, μ_k is a 3×1 mean vector of the colors in a window w_k, and I_3 is the 3×3 identity matrix.

The matrix L, which is called *matting Laplacian*, is the most important analytic result in this approach. The optimal alpha values are then computed as

$$\alpha = \arg\min \alpha^T L \alpha, s.t. \alpha_i = 1 \; or \; 0, \forall i \in \partial\Omega, \tag{2.8}$$

which can be solved by a single ”backslash” operator in Matlab.

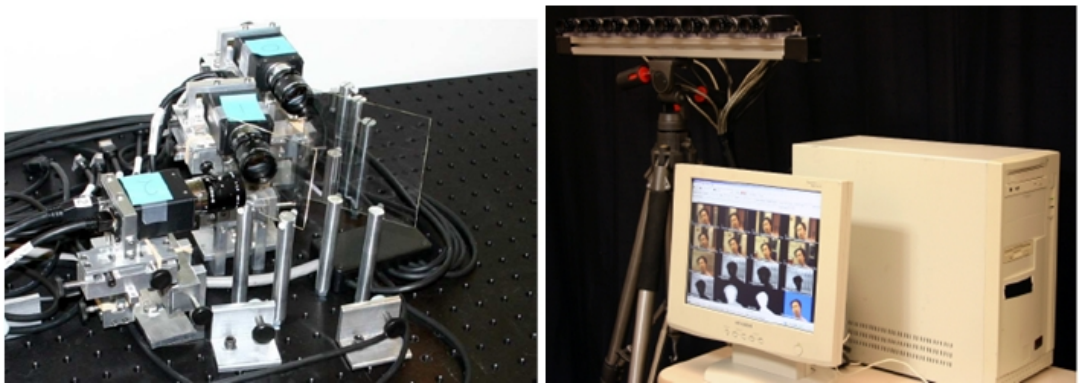


Figure 2.6: Left: the defocus matting system. Right: the multi-camera matting system.

This approach is capable of generating robust mattes for complex images, however as we will show later, since no F and B are estimated explicitly, it may not be able to generate good mattes for fine details in the image.

2.2.3 Extensions

Instead of a single input image, additional information can be used for reducing unknowns in matte estimation if available. The defocus matting system over-constrains the matting problem by capturing multiple synchronized video streams using a multi-sensor camera, as shown in Figure 2.6(a). Beam splitters allow all sensors to share a virtual optical center yet have varying parameters. The pinhole sensor has a small aperture that creates a large depth of field. It is nominally focused on the foreground. The foreground and background sensors have large apertures, creating narrower depths of field. The foreground sensor produces sharp images for objects within about 12 m of depth of the foreground object and defocuses objects farther away. The background sensor produces sharp images for objects from about 5m to infinity and defocuses the foreground object. Because the background camera's depth of field is very large and there is no parallax between our cameras, a background with widely varying depths can still be well approximated as a plane for the purpose of matting.

The flash matting system [121] recovers the matte from flash/no-flash image pairs, based on the simple observation that the most noticeable difference between the flash and no-flash images is the

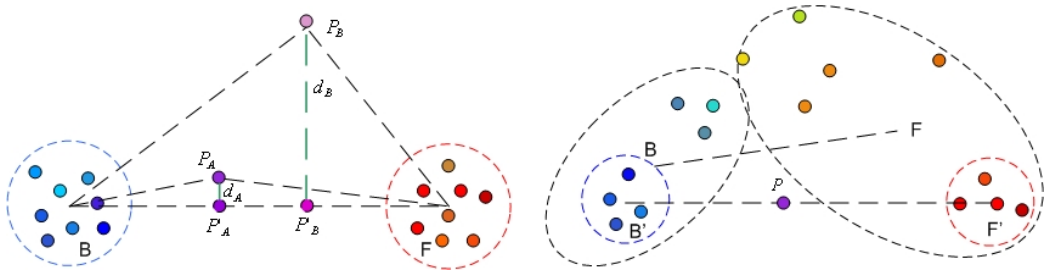


Figure 2.7: Two failure modes of previous matting approaches. Left: samples together with current pixel color do not fit well with the linear model. Right: only a subset of samples are valid for alpha value estimation.

foreground object, if the background scene is sufficiently distant. It makes two basic assumptions: 1) only the appearance of the foreground is dramatically changed by the flash; 2) the input image pair is pixel aligned. A joint Bayesian flash matting algorithm is developed to selectively fuse information from two images.

The multi-camera system [72] solves the matting problem by using a camera array, as shown in Figure 2.6(b). It uses high frequencies present in natural scenes to compute mattes by creating a synthetic aperture image that is focused on the foreground object, which reduces the variance of pixels re-projected from the foreground while increasing the variance of pixels re-projected from the background. A simple algorithm is proposed to extract the matte based on the aperture image, which has a per-pixel running time that is linear in the number of cameras.

In this dissertation we only consider the general problem of matte estimation from a single input image with limited user input. Other additional information, once available, can be easily incorporated into our algorithms for more accurate or automated matte estimation.

2.3 Failure Modes for Previous Approaches

Although various successful examples have been shown for previous approaches, their performance rapidly degrades when foreground and background patterns become complex. The intrinsic reason is that in complicated images the foreground and background regions contain significant discontinuities; thus direct color sampling may be erroneous, as well as fitting low-order models to them.

For an unknown pixel, sampling-based approaches collect a group of nearby foreground and

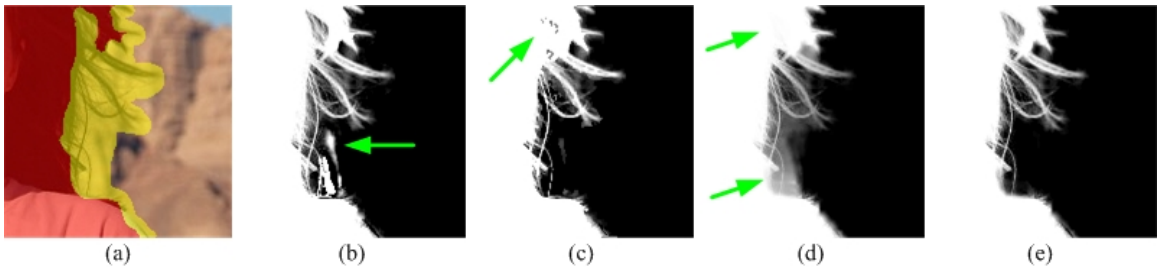


Figure 2.8: (a). Original image with user input. Following images show matte extracted by (b). Bayesian matting[23]. (c). Iterative matting[134]. (d). Closed-form matting[81]. (e). The robust matting proposed in this dissertation. Green arrows highlight artifacts.

background colors for alpha estimation. As illustrated in Figure 2.7a, these samples form clusters in color space, and the alpha value is estimated by projecting the pixel under consideration onto the line between foreground and background cluster centers to fit a linear model as in Equation 1.1. In the figure, we can see that pixel P_A fits the linear model very well, meaning it has a high probability of being a true mixed pixel between the foreground and background clusters. On the contrary, pixel P_B is far away from the interpolation line and it is very unlikely to be generated by a linear combination of the two clusters. Such a pixel is more likely to be an unmarked foreground or background pixel. Unfortunately, previous approaches ignore this fact and simply estimate an alpha value for P_B based on its projection to the line P'_B.

For complex foreground and background patterns, samples collected from local regions do not have a uniform color distribution, as shown in Figure 2.7b. In this case, propagation-based approaches will fail (at least partially) since the smoothness assumption is violated. For sampling-based methods which treat each sample equally, such as the Belief Propagation matting system [134], these samples will produce erroneous alpha values. Although Bayesian matting [23] will try to fit multiple Gaussians to color samples, it is still insufficient to determine the specific color samples (F' and B' in Figure 2.7b) that best explain the observed pixel, and thus should be used for alpha estimation.

Figure 2.8 demonstrates these limitations on a real example. Figure 2.8a shows the original image with the user specified foreground (red) and unknown region (yellow). Note that although the foreground color is relatively uniform, the background contains complex patterns, in which dark

regions match well with the foreground color. Figure 2.8 b, c, and d show that Bayesian matting, Belief Propagation matting and closed-form matting will produce noticeable artifacts due to color sampling errors or assumption violations. In contrast, the robust matting approach proposed in this thesis is able to generate a good matte.

2.4 Robust Matting

We propose a new matting algorithm [1] to counter the sampling problems faced by previous approaches. Our algorithm is based on an optimized color sampling scheme, as we believe (and shown by experimental results) that this non-parametric technique is more robust for natural images. A Random Walk optimizer is employed to solve for the matte.

2.4.1 Optimized Color Sampling

Given the input image and a roughly specified trimap, for a pixel z with unknown α, our algorithm first assembles a large number of foreground and background samples as candidates for estimating the true foreground and background colors at this location. We will discuss how these samples are drawn later. We make the assumption that for any mixed pixels, the true foreground and background colors F_z and B_z are close (in color space) to some samples in this large sample set. This assumption is relatively loose compared with those proposed in previous approaches, since we can always increase the size of the sample set to cover more known pixels.

The challenging task is to pick out "good" samples from this large candidate set. Good sample pairs should explain any mixed foreground/background pixels as linear combinations of the samples. For example, as shown in Figure 2.7a, two samples define a line in color space. If this line passes through (or near) the color of the pixel under consideration, it successfully explains the pixel's color as a convex combination. Specifically, for a pair of foreground and background colors F^i and B^j, the estimated alpha value is

$$\hat{\alpha} = \frac{(C - B^j)(F^i - B^j)}{\| F^i - B^j \|^2} \tag{2.9}$$

We define a *distance ratio* $R_d(F^i, B^j)$, which evaluates this sample pair by examining the ratio of the distances between (1) the pixel color, C, and the color it would have, $\hat{C}$, predicted by the linear

[1] This algorithm has been published in [135].

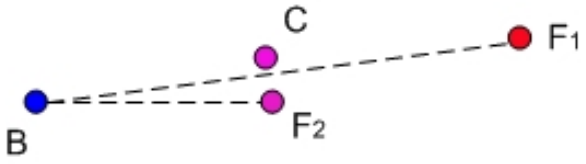


Figure 2.9: Using distance ratio defined in Equation 2.10 alone cannot guarantee good sampling. C is an unmarked foreground pixel, thus F_2 is the correct foreground sample and F_1 is not.

model in Equation 1.1, and (2) the distance between the foreground/background pair:

$$R_d(F^i, B^j) = \frac{\parallel C - (\hat{\alpha}F^i + (1-\hat{\alpha})B^j) \parallel}{\parallel F^i - B^j \parallel} \tag{2.10}$$

In the example shown in Figure 2.7a, the distance ratio will be much higher for P_B than P_A, indicating the samples are not as good for estimating alpha for P_B.

The distance ratio alone will favor sample pairs that are widely spread in color space since the denominator $\parallel F^i - B^j \parallel$ will be large. Since we expect most pixels to be fully foreground or background, pixels with colors that lie nearby in color space to foreground and background samples are more likely to be fully foreground or background themselves. Thus, for each individual sample we define two more weights $w(F^i)$ and $w(B^j)$ as

$$w(F^i) = 1.0 - exp\left\{-\parallel F^i - C \parallel^2 / D_F^2\right\} \tag{2.11}$$

and

$$w(B^j) = 1.0 - exp\left\{-\parallel B^j - C \parallel^2 / D_B^2\right\} \tag{2.12}$$

where D_F and D_B are the minimum distances between foreground/background sample and the current pixel, i.e., $\min_i(\parallel F^i - C \parallel)$ and $\min_j(\parallel B^j - C \parallel)$.

Combining these factors, we calculate a final *confidence* value $f(F^i, B^j)$ for a sample pair as

$$f(F^i, B^j) = exp\left\{-\frac{R_d(F^i, B^j)^2 \cdot w(F^i) \cdot w(B^j)}{\sigma^2}\right\} \tag{2.13}$$

where σ is fixed to be 0.1 in our system. This roughly says that if the distance ratio $R_d(F^i, B^j) > 0.5$, the confidence $f(F^i, B^j)$ for this sample pair will degrade to zero. Alternatively, if $F^i(B^j)$ is very close to C, then $w(F^i)(w(B^j))$ will be significantly smaller, resulting a larger confidence value $f(F^i, B^j)$.

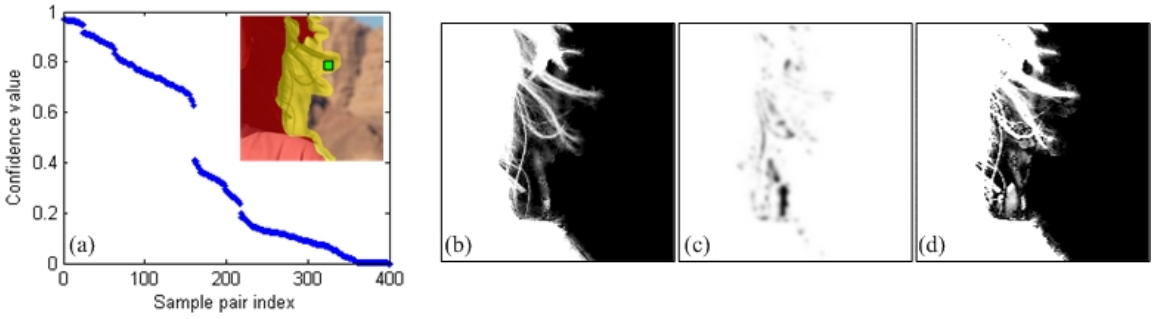


Figure 2.10: (a). Sorted confidence values for 400 foreground-background sample pairs for the green pixel. (b). Estimated alpha values by using sample pairs with highest confidences. (c). Confidence values. White pixels have higher confidence. (d). Estimated alpha values by using sample pairs with lowest confidence.

We examine the confidence of every pair of foreground and background samples from the large number of candidates. Finally, we select a small number of pairs (3 in our system) with the highest confidences. The average estimated alpha value and confidence of these three sample pairs are taken as the final values in the color sampling step. These values are used in the optimization process described later to generate the final matte for the image.

In Figure 2.10(a), we first selected 20 foreground and 20 background samples corresponding to the highlighted pixel. This results in 400 foreground/background pairs. Each pair provides an estimated alpha and confidence. The confidence values are sorted and plotted in the figure.

By selecting those pairs with highest confidence values, we generate the initial alpha matte shown in Figure 2.10(b), which will be further improved by the optimization process. Figure 2.10(c) shows the accompanying confidence map (i.e., the average of the three highest confidence values). Note the correlation between pixels whose alpha values are mis-estimated and the dark (low confidence) regions in the confidence map. For comparison, Figure 2.10(d) shows a much poorer matte generated by using sample pairs with lowest confidences.

2.4.2 Collecting Samples

Finally, one remaining question is how to construct the initial sample set. Previous approaches such as Bayesian matting and Belief Propagation matting collect pixels known to be fully foreground

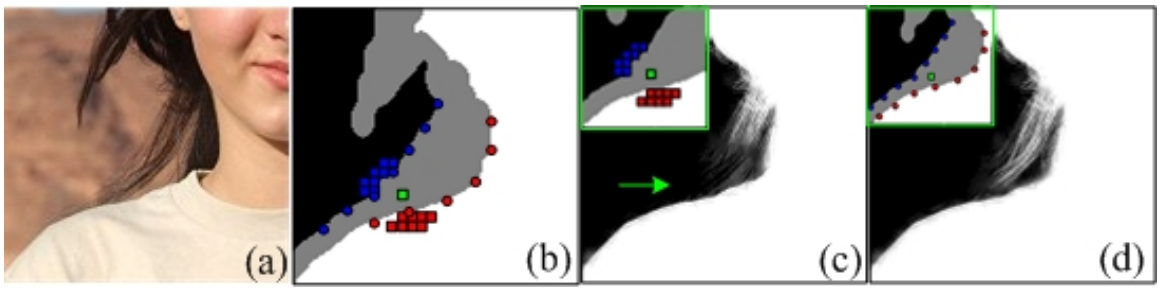


Figure 2.11: (a). Original image. (b). Two sampling method: nearest spatial neighbors (square) and sparse samples (round). (c). Matte generated with square samples. (d). Matte generated with round samples.

or background that have the shortest spatial distances to the target pixel as samples. As shown in Figure 2.11, in regions with complex structure, the spatially nearest pixels may not fully span the variation in foreground and background colors.

We thus spread the sampling of foreground and background samples along the boundaries of known foreground and background regions. In this way the sample set can better capture the variation of foreground and background colors. The higher variation benefits the robust sampling method outlined above but may be problematic for previous approaches that simply average over the full sample set. Figure 2.11 shows that a sample set we collect performs better at least for this example.

2.4.3 Matte Optimization

As we have seen, the sampling process leads to a good initial alpha estimate and a confidence value for each pixel. This initial estimate can be further improved by leveraging a priori expectations about more global aspects of the alpha matte. In particular, we expect the matte to exhibit local smoothness. We also expect alpha values of one or zero (fully foreground or background) to be much more common than mixed pixels. As we can see in Figure 2.10b and c, the pixels with low confidence values are those unmarked foreground and background pixels.

Our expectation for the matte is thus two fold: firstly, it should respect the alphas chosen for each individual pixel (*data constraint*) especially when the confidence value is high; secondly, the matte should be locally smooth and robust to image noise (*neighborhood constraint*). As shown in previous graph-based image labelling approaches [134, 83, 110], this expectation can be satisfied

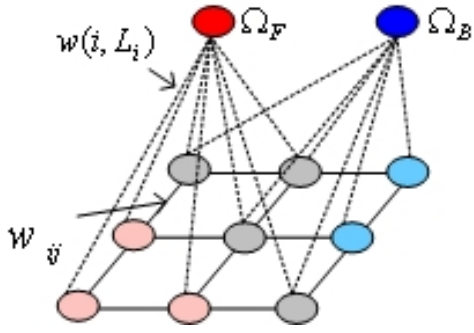


Figure 2.12: Our system estimates the matte by using Random Walk to solve a graph labelling problem.

by solving a graph labelling problem shown in Figure 2.12, where Ω_F and Ω_B are virtual nodes representing pure foreground and pure background; white nodes represent unknown pixels on the image lattice; and light red and light blue nodes are known pixels marked by the user. A *data weight* is defined between each pixel and a virtual node to enforce the data constraint, and an *edge weight* is defined between two neighboring pixels to enforce the neighborhood constraint.

The data weights correspond to relative probabilities of a node being foreground or background. For nodes for which we have high confidence values, $\hat{f}_i$, we rely on the alpha that fits the linear model from the selected samples. When the confidence is low, we have a higher expectation that the node is fully foreground or background (i.e., alpha is either 1 or 0). Which one to bias alpha towards is determined by the initial alpha estimate.

Specifically, as shown in Figure 2.12, for an unknown pixel i, two data weights $W(i, F)$ and $W(i, B)$ are assigned to the links between pixel i and each virtual node. They are defined as

$$W(i, F) = \gamma \cdot [\hat{f}_i \hat{\alpha}_i + (1 - \hat{f}_i)\delta(\hat{\alpha}_i > 0.5)] \tag{2.14}$$

and

$$W(i, B) = \gamma \cdot [\hat{f}_i (1 - \hat{\alpha}_i) + (1 - \hat{f}_i)\delta(\hat{\alpha}_i < 0.5)] \tag{2.15}$$

where $\hat{\alpha}_i$ and $\hat{f}_i$ are estimated alpha and confidence values, and δ is boolean function returning 0 or 1. γ is a free parameter in our system which balances the data weight and the edge weight.

We also need to specify the weights, $W_{i,j}$, between nodes to encourage alpha to have local smoothness. The closed-form matting system [81] sets the weights between neighboring pixels

based on their color difference computed from local color distributions. In this work, we also choose to use this formulation for edge costs W_{ij} due to its simplicity and efficiency.

Formally, the neighborhood term, W_{ij}, is defined by a sum over all 3×3 windows that contain pixels i and j.

$$W_{ij} = \sum_{k}^{(i,j)\in w_k} \frac{1}{9}(1 + (C_i - \mu_k)(\Sigma_k + \frac{\epsilon}{9}I)^{-1}(C_j - \mu_k)) \tag{2.16}$$

where w_k represents the set of 3×3 windows containing pixels i and j, and k iterates over those windows. μ_k and Σ_k are the color mean and variance in each window. ϵ is a regularization coefficient which is set to be 10^{-5} in our system. More details and justifications of this neighborhood term can be found in [81].

Solving for Optimal αs

Given the fact that alpha values are continuous, we avoid discrete labelling optimizations such as graph cut or belief propagation. Instead, we solve the graph labelling problem as a Random Walk [50, 51], which has been shown to minimize the total graph energy over real values. Although a detailed description of Random Walk theory is beyond the scope of this thesis, it essentially works as follows.

First, we construct a Laplacian matrix for the graph as

$$L_{ij} = \begin{cases} W_{ii} & : \text{ if } \quad i = j, \\ -W_{ij} & : \text{ if i and j are neighbors}, \\ 0 & : \text{ otherwise}, \end{cases} \tag{2.17}$$

where $W_{ii} = \sum_j W_{ij}$. L is thus a sparse, symmetric, positive-definite matrix with dimension $N \times N$, where N is the number of all nodes in the graph, including all pixels in the image plus two virtual nodes Ω_B and Ω_F.

We then decompose L into blocks corresponding to unknown nodes P_u, and known nodes P_k including user labelled pixels and virtual nodes, as

$$L = \begin{bmatrix} L_k & R \\ R^T & L_u \end{bmatrix} \tag{2.18}$$

It has been shown [51] that the probabilities of unknown pixels belonging to a certain label (for example, foreground) is the solution to

$$L_u A_u = -R^T A_k \tag{2.19}$$

where A_u is the vector of unknown alphas we wish to solve for, and A_k is the vector encoding the boundary conditions, (i.e., 1's and 0's for the known alpha values of the virtual and user specified nodes). L_u is guaranteed to be nonsingular for a connected graph, thus the solution A_u is guaranteed to exist and be unique with values guaranteed to lie between 0 and 1. We use Conjugate Gradient (CG) to solve the linear system.

Iterative Refinement

The full sampling and solution run iteratively. After each step, alpha values above 0.98 are declared to be known foreground and clamped to 1.0. Likewise, alpha values below 0.02 are set to known background. This provides new color samples for mixed pixels in the next iteration. We compute a new confidence map after each iteration, and stop the iterative process when the change in the confidence map is small enough. In our tests the process converged after 2 to 4 iterations.

The Free Parameter γ

As we mentioned there is one major free parameter in our system, the balancing weight γ in Equation 2.14. In general, setting the weight to be too low will result in an over-smoothed matte, while setting it to be too high will generate a noisy one. We set $\gamma^* = 0.1$ in our system for all examples shown in this dissertation, which generates generally good results.

2.4.4 The Algorithm

The robust matting algorithm is formally described as follows:

The Robust Matting Algorithm

1. Given the input image and a user specified trimap;

Figure 2.13: Up: test images. Middle: ground-truth mattes. Bottom: fine-to-coarse trimaps for one image.

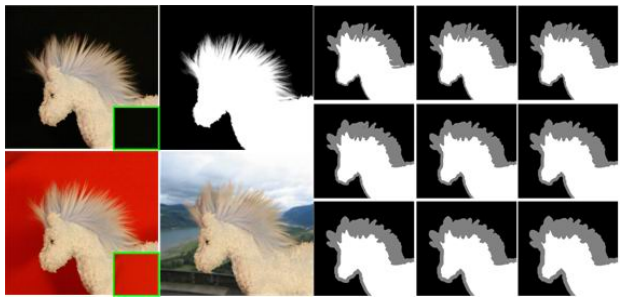


Figure 2.14: Left: same foreground against two known backgrounds. Middle: extracted matte and new composite. Right: fine-to-coarse trimaps.

2. Use the optimized color sampling method described in Section 2.4.1 to generate an initial alpha map $\hat{\alpha}(0)$ and a confidence map $\hat{f}(0)$;

   while(true)

   (a) Solve the matte as described in Section 2.4.3 using current $\hat{\alpha}$ and $\hat{f}$;

   (b) Apply color sampling again to update the alpha map $\hat{\alpha}(t)$ and the confidence map $\hat{f}(t)$;

   (c) If $\| \hat{f}(t) - \hat{f}(t-1) \| < \epsilon$, break;

3. End.

2.5 A Quantitative Evaluation

Previous matting papers only show a few successful mattes to verify their methods. Although limited quantitative comparisons have been shown in [81], there is no objective and quantitative evaluation on how robust these methods to various user inputs. In this section we present such an evaluation and comparison.

2.5.1 Materials and Methods

Our test set includes 8 test images along with ground-truth mattes, as shown in Figure 2.13. Image T1 to T3 are generated using the blue screen matting technique described in [118]. As shown in Figure 2.14, a fuzzy foreground is shot against two known backgrounds, which enables us to use triangulation matting technique to pull out a fairly accurate matte. The matte is then used to create a new composite which serves as the test image. Image T4 to T6 use real people and animals as foregrounds. They are generated in a similar way except only one solid-color background is used, and the Bayesian matting technique is used for extracting the initial matte for synthesizing the composites.

To evaluate how the algorithms perform on real natural images instead of synthetic ones, our data set includes two natural images T7 and T8. Since the real ground-truth mattes are unknown in this case, we generate the best possible approximations using a variety of methods along with extensive user assistance. We first manually create a very accurate trimap for each image. We then apply previous matting algorithms to get the best matte from each individual algorithm. We then manually combine these mattes by using the best matte regions of all candidates. Finally, a manual correction process which is similar to the one proposed in Poisson Matting system [120] is employed for further improvement.

To evaluate the accuracy and robustness of different algorithms, we first create a fairly accurate trimap T_0 for each test image, then dilate the unknown region gradually to create a series of less accurate trimaps T_1, T_2, ..T_8. We apply a number of previous algorithms on each test image and each trimap, and calculate the Mean Squared Error (MSE) of the estimated mattes against the ground-truth. In this way we can quantitatively compare different methods.

In this study we compare the performances of 7 algorithms: Bayesian matting [23], Believe

Propagation matting [134], Global Poisson matting [120], Random Walk matting [52], KnockOut 2 [33], closed-form matting [81], and the robust algorithm proposed in this thesis. Note that some approaches such as Belief Propagation matting and closed-form matting have the added ability to work with user defined scribbles. However, as shown in their examples, the scribbles must be carefully placed in order to get good mattes. A roughly specified trimap, which can be drawn with a single fat stroke, creates a similar burden for the user as creating scribbles and thus stands in for user scribbles. This allows us to test all algorithms in a uniform way.

2.5.2 Results and Discussion

Figure 2.15 shows the MSE curves of different algorithms for one test image. As expected, most algorithms achieve their best performance with the finest trimap. As the trimaps becomes coarser, their performances generally degrade, but at different rates. We also show partial mattes at two trimap levels (2 and 8) since MSE values do not always correlate exactly with visual quality. Our algorithm gives the best result at all trimap levels, both quantitatively and visually. Figure 2.16 shows the result on another test image.

We define two numerical indicators I_a and I_r to measure the accuracy and robustness of different algorithms, respectively. For an algorithm and a test image, we compute a series of MSE values using different trimaps, and define I_a as the minimal MSE value, and I_r as the difference between the maximal and minimal values. Table 2.5.2 shows the I_a and I_r values for all algorithms on all the test images along with their ranks. It demonstrates that our algorithm consistently ranks the top in terms of both accuracy and robustness.

This study reveals that pure sampling-based approaches, such as Bayesian matting, can generate fairly good mattes with tightly defined trimaps. However, when the trimap becomes coarser, the performance rapidly degrades since their assumptions on color samples are violated. Although our approach cannot fully avoid the performance degradation, the robust color sampling method we propose helps our system generate more accurate result with fine trimaps, and be more robust to the variance of user input.

On the other hand, propagation-based methods, such as the closed-form matting, can generate stable results across all trimap levels. It is not a surprise that for some examples, at coarser trimap

Table 2.1: I_a and I_r values for different algorithms on different test images and their ranks(Format: $I_a^{rank} : I_r^{rank}$). Bottom line shows the average ranks.

	Poisson	Rand. Walk	Knockout2	Bayesian	Iterative BP	Closed-form	Our
T1	717^7:1959^7	286^5:343^5	155^4:321^4	453^6:1198^6	61^2:182^3	79^3:97^2	46^1:85^1
T2	477^6:1577^6	482^7:748^5	326^4:678^4	415^5:2006^7	118^3:337^3	105^2:234^2	44^1:101^1
T3	843^7:1736^7	218^6 :278^5	113^3:180^3	137^5:1005^6	130^4:205^4	61^2:80^2	38^1:67^1
T4	340^7:1330^7	198^6:307^3	154^5:596^5	82^4:724^6	69^3:356^4	59^2:137^2	41^1:95^1
T5	451^7:2891^7	151^6:393^5	33^5:336^4	28^4:687^6	27^3:227^2	23^2:356^3	10^1:155^1
T6	879^7:3174^7	279^5:638^3	338^6:1387^6	194^3:938^5	207^4:903^4	157^2:237^1	69^1:381^2
T7	359^7:1830^7	274^6:401^4	150^5:516^6	69^2:406^5	78^4:362^3	77^3:143^1	31^1:165^2
T8	832^6:2442^6	1732^7 :1795^5	435^4:888^4	120^2:4994^7	214^3:553^2	503^5:582^3	114^1:394^1
Rank	6.9:6.8	6.0:4.4	4.5:4.5	3.9:6.0	3.3:3.1	2.6:2.0	1.0:1.3

levels, the closed-form approach estimates mattes with lower MSEs than our algorithm does, which means the color samples are no longer reliable when known pixels are far away from unknown ones. However, the major limitation of this approach is that it cannot generate very accurate mattes with fine trimaps. As shown in Figure 2.15, some fine details of the foreground are always missing in its estimation. This fact demonstrates the contribution of the color sampling method in our approach to the accuracy of the extracted mattes. In practice we can potentially lower the data weight γ in Equation 2.14 for coarse trimaps where samples are not reliable.

Figure 2.17 shows the best mattes we extracted for a few other test images.

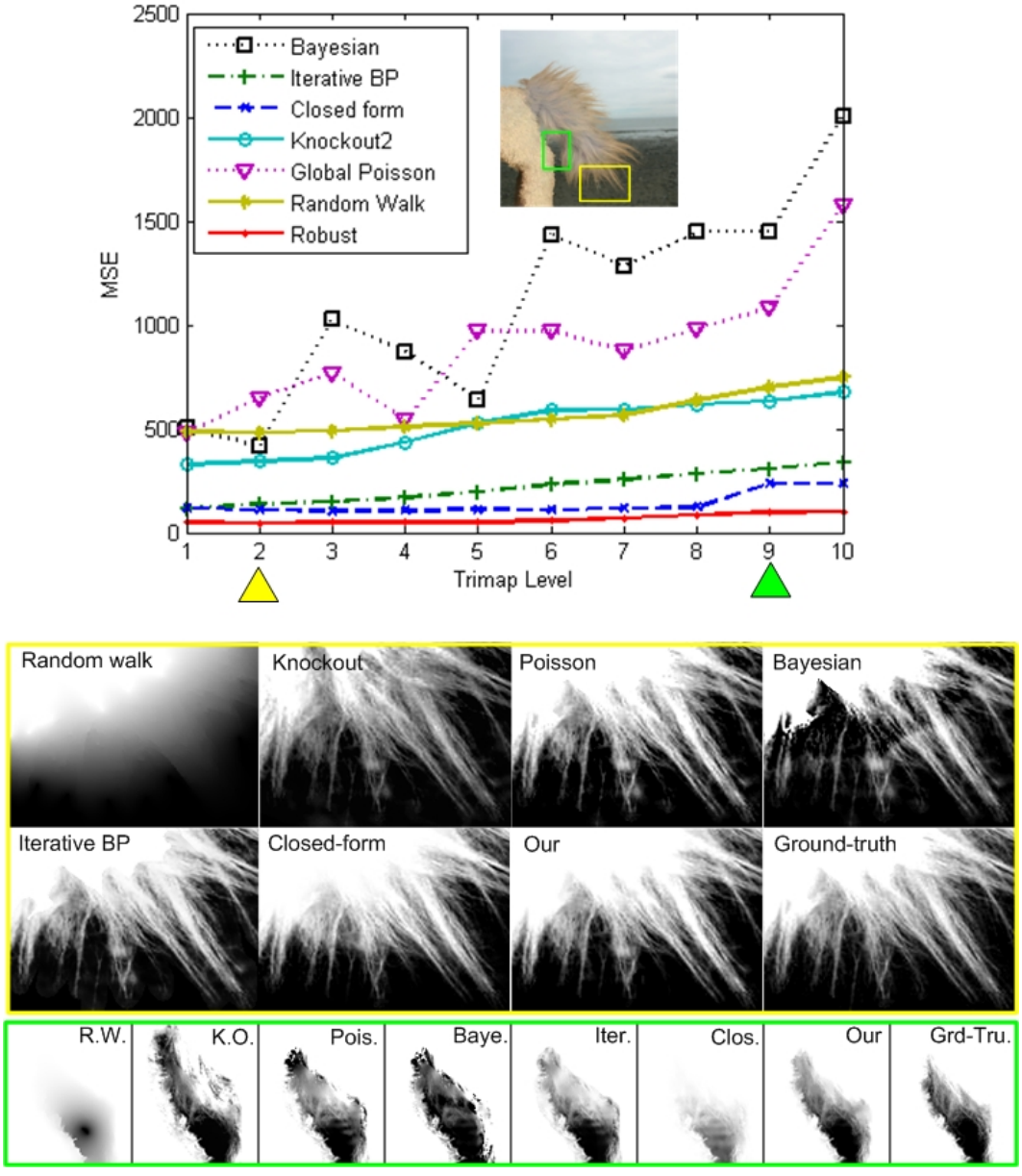


Figure 2.15: Top: MSE curves for test image T2. Bottom: Partial mattes computed at two trimap levels (indicated by colored triangular marks).

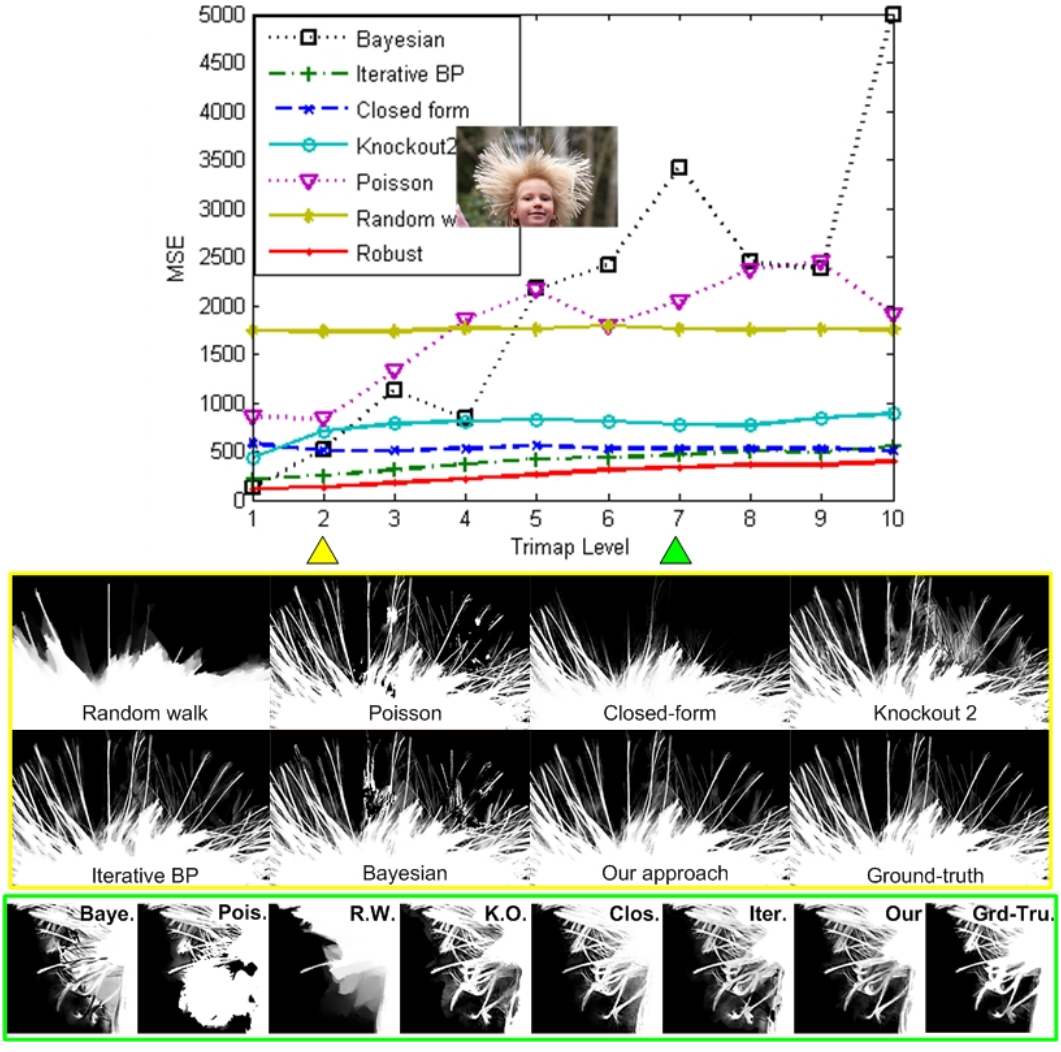


Figure 2.16: Top: MSE curves for test image “kid”(700×438). Bottom: Mattes computed at two trimap levels (indicated by colored triangular marks). Only two small regions of the mattes are shown due to space limitation.

Figure 2.17: Best mattes extracted by our algorithm for other examples.

Chapter 3

SOFT SCISSORS FOR REALTIME MATTING

3.1 Offline V.S. Realtime Matting

As described in the previous chapter, most previous matting algorithms as well as the Robust Matting algorithm we presented work in an offline fashion. They require the user to first roughly segment the image into a trimap in which pixels are marked as definitely belonging to the background, definitely belonging to the foreground, or unknown. These algorithms then use information from the known background and foreground regions to compute a matte. If the initial results are not satisfactory (which is often the case), the user must then refine the trimap and run the algorithm again until the process converges. In practice this process is usually very inefficient for the user.

Recent matting algorithms focus mainly on improving the quality of the matte by introducing more sophisticated analysis and optimization methods. However they are generally slow. As a result, the wait time between each iteration of the interactive loop described above can be very long. For instance, Bayesian matting [23] takes 141 seconds of computation time to generate a result for the example shown in Figure 3.1. Also, these techniques re-compute the whole matte on each iteration and there is no good strategy to update the matte incrementally. On the other hand, earlier approaches such as the Knockout 2 system [33] are extremely simple and fast, but are not capable of generating high quality mattes for complex images.

We create a system which we dub *Soft Scissors* [1], an interactive tool for extracting high quality alpha mattes of foreground objects in realtime. The system is built upon the offline Robust Matting algorithm we proposed in the previous chapter, and we improve it to give it the ability to incrementally update the matte in an online interactive setting. In our system the user roughly specifies the foreground boundary using an intelligent paint stroke (or *soft scissor*). As shown in Figure 3.1(a,b), the system automatically updates the matte and foreground colors according to the newly-added information along the stroke to instantly reveal a local region of the final composite. The high quality

[1] This system has been published in [132].

Figure 3.1: In the *soft scissors* system the high quality matte (a) and a novel composite (b) is computed in realtime when the user is specifying the rough foreground boundary. Using the system a successful composite (c) is created in a few seconds with one scissor stroke.

composite shown in Figure 3.1(c) took about 40 seconds of total interleaved user and computation time.

In adapting the Robust Matting algorithm to the realtime setting, we make three contributions in the Soft Scissors system:

Incremental matte estimation. Based on newly-add user strokes, the system first determines the minimal number of pixels that need to be updated, and then computes their new alpha values.

Incremental foreground color estimation. In addition to alpha values, our system also incrementally computes the foreground colors for truly-mixed pixels so the final composite can be updated immediately.

Intelligent user interface. The soft scissor width and the boundary conditions are automatically adjusted to approximately capture the boundary that lies ahead on the scissor's path.

By combining all of these elements together, we develop the first system to generate high quality mattes and composites in realtime.

3.2 The Soft Scissors Algorithms

3.2.1 Overview

The system updates the matte in realtime while the user roughly paints a scissor stroke along the boundary of the foreground object. A flowchart of each internal iteration of the system is shown in Figure 3.2. We assume that the scissor stroke implicitly defines a trimap, usually with the left edge of the stroke assumed to lie in the background (blue pixels); the right edge assumed to lie in

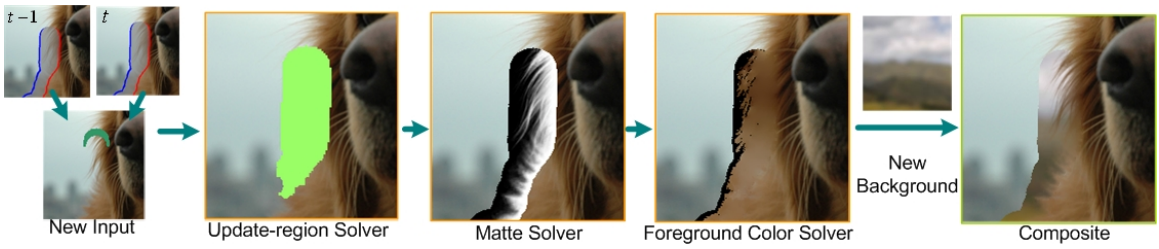


Figure 3.2: A flowchart of our system.

the foreground (red pixels); and the middle of the stroke unknown (gray pixels). The Soft Scissors interface is described in detail in Section 3.3. On each iteration the system determines which pixels were painted since the previous iteration. Because this new input region, M_t (shown in dark green), provides more trimap information about the image, it will affect the alpha values for surrounding pixels. Thus, we use the input region to seed an update-region solver (see Section 3.2.4) which computes a small region of pixels for which the alpha values will be computed. This matting region Ω_t (shown in light green) is generally much smaller than the whole unknown region in the trimap, and therefore solving the matte is significantly more efficient than re-calculating the whole unknown region in each iteration.

We update the alpha values for pixels in Ω_t using a robust matte solver (see Section 3.2.2). By treating pixels outside Ω_t as boundary conditions, the solution is guaranteed to be smooth across the boundary of Ω_t. Finally, the foreground colors of pixels in Ω_t are updated by a foreground color solver that uses the newly computed alpha values (see Section 3.2.3). We then display the updated composite in Ω_t.

3.2.2 Solving for the Matte

The central component of our soft scissors system is the Robust Matting algorithm presented in the previous chapter. For completeness we briefly summarize the algorithm here.

As illustrated in Figure 3.3, assume that we have already computed the update region Ω_t (shown in light and dark green). We treat the problem of solving for α in this region as a soft graph-labeling problem. We use the graph structure shown in Figure 3.4(a), where Ω_F and Ω_B are virtual nodes representing pure foreground and pure background; white nodes represent unknown pixels in the

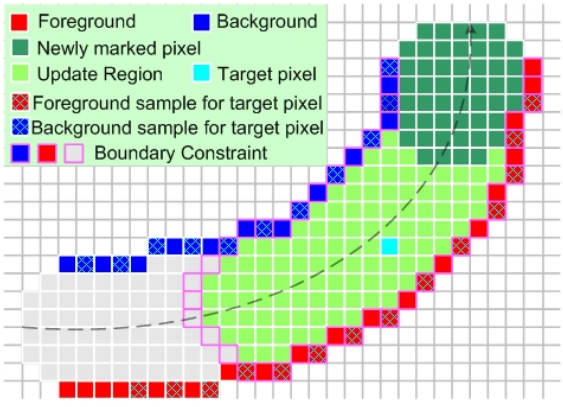


Figure 3.3: Our system quickly solves the matte under the leading edge of the soft scissors, constrained by boundary pixels.

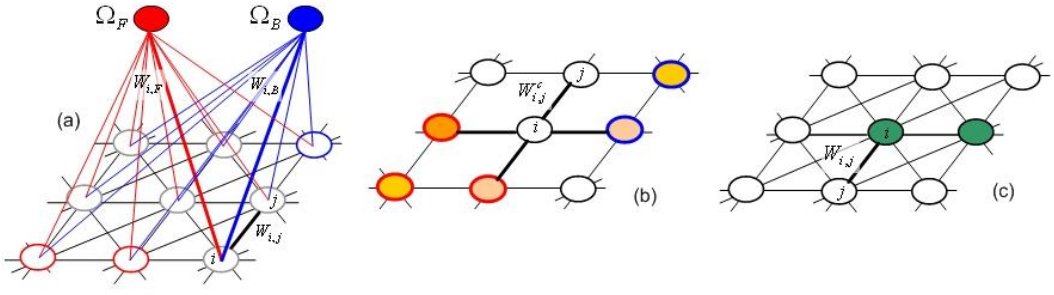


Figure 3.4: The matte (a), foreground colors (b) and the update region (c) are solved as soft graph-labelling problems.

image; and light red and light blue nodes are boundary nodes whose alpha values are fixed in this iteration. The boundary nodes for this graph include not only user marked foreground and background pixels, but also unknown pixels on the boundary of Ω_t whose alpha values have been estimated in previous iterations. In this way we ensure the matte is smooth across the entire boundary of the update region.

We selectively sample a group of known foreground and background pixels from the boundary of the trimap to compute non-parametric models of the foreground and background color distributions. We then assign *data weights* $W_{i,F}, W_{i,B}$ between pixel i and the virtual nodes based on these distributions. The data weights constrain pixels that are similar in color to the foreground

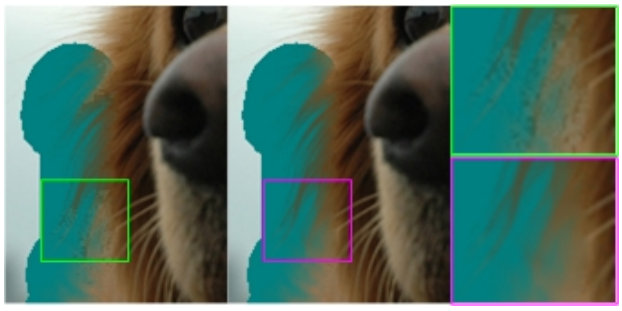


Figure 3.5: Left: Initial estimates of foreground colors after the matte estimation step; Right: Final foreground colors after optimization.

(background) to have a stronger $W_{i,F}$($W_{i,B}$) and therefore make them more likely to have a higher (lower) alpha values. We use the formulation proposed in the closed-form matting paper [81] to set the *edge weights* $W_{i,j}$ between each pair of neighboring pixels i and j. Note that each pixel is connected to its 25 spatial neighbors in this formulation. The edge weights constrain nearby pixels to have similar alpha values. Once the graph is constructed, we solve the graph-labelling problem as a Random Walk [51], which minimizes the total graph energy over real values.

Intuitively the Random Walk solver determines the alpha values by placing a random walker at pixel i that can walk to any neighboring node j (i.e. any node connected to i including the two virtual nodes) with probability $W_{i,j}/\sum_j W_{i,j}$. The walker then moves from j to another neighbor k in the same manner, and so on. This process iterates until the walker reaches one of the boundary nodes. The probability that the walker ends up at the foreground virtual node determines the alpha value of pixel i. This probability can be naively estimated by simulating the random walk process a large number of times, and counting how many times it arrives at the foreground node. In practice, however, we calculate the unknown alphas in closed-form by solving a large linear system using the random walk algorithm outlined in [51].

3.2.3 Solving for the Foreground Colors

In addition to computing alpha values we also estimate the foreground color F for each pixel in the unknown region. Although we select a few foreground samples for each pixel in the matte estimation step, these samples are chosen individually without enforcing smoothness constraints.

As a result, after the matte estimation step, the composite may contain visual artifacts, as shown in Figure 3.5.

To achieve higher quality composites, we refine the estimated foreground colors by solving a second graph-labeling problem using Random Walk, as shown in Figure 3.4(b). Only those pixels in Ω_t whose alpha values are strictly between 0 and 1 are treated as unknown pixels in this step, and each unknown pixel is connected to its 4 spatial neighbors. We define a color edge weight $W^c_{i,j}$ between two neighbors as $W^c_{i,j} = |\alpha_i - \alpha_j| + \varepsilon$, where ε is a small value ensuring the weight is greater than zero. This edge weight encodes explicit smoothness priors on F, which are stronger in the presence of matte edges (where α_i and α_j have a larger difference).

The boundary pixels in this step are either foreground pixels ($\alpha = 1$) or background pixels ($\alpha = 0$). For foreground pixels (red outline nodes in Figure 3.4(b)), we use their true colors as boundary conditions, while for background pixels (blue outline nodes in Figure 3.4(b)), we use their initially estimated foreground colors in the matte estimation step as boundary conditions. The initial estimates are shown as the node colors in Figure 3.4(b). We then solve for the three foreground color channels individually using the Random Walk solver.

3.2.4 Solving the Update Region

A key feature of the Soft Scissors system is that it is incremental – we only update the alpha and foreground colors for a small portion of the image on each iteration. Given a new input region we compute the set of pixels that might be affected by the new information as the update region Ω_t. Once again we solve a graph-labeling problem as shown in Figure 3.4(c). All pixels that have been newly marked by the user in the current iteration are treated as boundary pixels with an assigned label of 1 (the dark green nodes in Figure 3.4(c)). All other pixels that have been marked in previous iterations are treated as unknown pixels in this step (white nodes in Figure 3.4(c)). Similar to the alpha estimation graph in Figure 3.4(a), each pixel is connected to its 25 spatial neighbors with the same edge weights $W_{i,j}$ as we defined in the matte estimation step. We again solve the graph using the Random Walk solver. Those pixels that are assigned non-zero labels form the new update region Ω_t.

Intuitively, the Random Walk solver determines how far potential changes of alpha values due

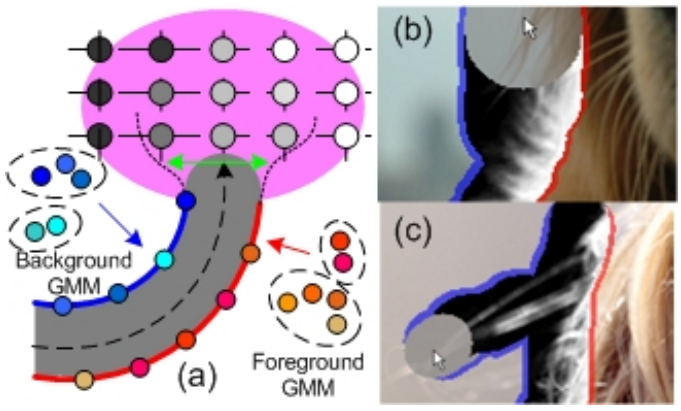


Figure 3.6: (a). Our system can automatically determine the soft scissor width and boundary conditions. (b). An example of enlarging the width to cover the mixed foreground/background region. (c). An example of changing the boundary condition.

to the newly marked pixels should be propagated towards the boundary of the image. A smoother local image region will result in a larger Ω_t since the weights between neighboring pixels are high, and vice versa. This solver is similar in spirit to the region solvers employed in the interactive tone mapping system [86], but with different graph topologies and edge weights.

3.3 The Soft Scissor Interface

As the user paints along the boundary of the foreground object our system dynamically adjusts two properties of the Soft Scissors brush; 1) brush width and 2) boundary conditions for the trimap that are implicitly defined by the brush strokes. The adjustments are based on local statistics near the current brush stroke. In addition, users can manually adjust these parameters if necessary.

Choosing the scissor brush width

Wider scissors are appropriate for object edges that are very fuzzy while narrower scissors are better for sharper edges as they provide tighter bounds on the solution and greatly improve computation efficiency. Our approach for automatically determining scissor brush width is shown in Figure 3.6(a). At each time t, we create a wide "look-ahead" region (shown in purple) by extending the current path of the scissor. We treat all pixels in this region as unknowns and include them in Ω_t for alpha

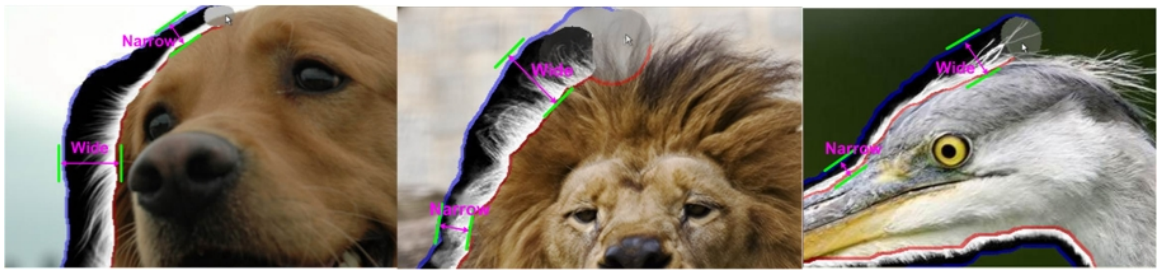


Figure 3.7: Examples of automatic brush width adjustment.

estimation. Then, to estimate the matte profile we sample a group of pixels sparsely distributed along lines perpendicular to the current scissor path direction. The scissor width is set so that it covers all of the sample pixels with fractional alpha estimates.

Figure 3.7 shows a few other examples on automatically changing the brush width.

Determining the boundary conditions

Initially we assume that the user orients the scissor brush strokes so that the left edge of the brush is in the background region and the right side of the brush is within the foreground region. However, at times users need to follow thin structures (e.g. single hairs). In this case they require a brush in which both sides are marked as background as in Figure 3.6(c). In other situations users will paint back and forth over the foreground object and the brush must be able to reverse the background/foreground edges as the user reverses the brush direction.

We dynamically adjust the scissor brush boundary conditions by building color models of the foreground and background colors. Once the user has created a short stroke and marked enough foreground/background pixels under the initial assumptions we use Gaussian Mixture Models (GMM) to build foreground and background color models. Then, for each new brush position we classify the brush edges based on whether their average color is closer to the foreground or background GMM. In addition, the GMMs are updated periodically using recently marked foreground and background pixels. An example is shown in Figure 3.6(c).

3.4 Results and Evaluation

The images in Figure 3.1 show one example of the Soft Scissors in use. Figure 3.8 shows three more challenging examples of our system running on complex images and the resulting high quality mattes.

The system not only runs in realtime, but also generates higher quality results than previous approaches. To quantitatively and objectively evaluate our system, similar to the approach we used for evaluating the Robust Matting algorithm as we described in the previous section, we constructed a test data set of 5 examples as shown in Figure 3.9. Each foreground object was originally shot against a solid colored background and we extracted a high quality matte using Bayesian matting. We used the resulting matte as the ground-truth and composited the foreground object onto a more complex background to synthesize a "natural" image as a test image. Finally we applied various matting approaches on these test images.

We compare mattes extracted using Soft Scissors with five previous matting approaches: Bayesian matting [23], iterative BP matting [134], closed-form matting [81], knockout 2[33] and global Poisson matting [120]. We also compare the mattes with our offline Robust Matting approach. All of these techniques were run using the same trimap created using our interactive system. Figure 3.10 shows the Mean Squared Error (MSE) of the extracted mattes against the ground-truth. Two visual examples are shown in Figure 3.12. These results clearly suggest that our system extracts the mattes with highest quality. Note that Soft Scissors generates slightly better results than the offline Robust Matting approach due to implementation improvements. The total processing time of different approaches are also shown in Figure 3.11. For fair timing comparisons we fixed the total number of iterations in the offline approaches to 4. Not surprisingly, Soft Scissors takes the least amount of time for extracting good mattes.

Our system also works well with more solid foreground objects. In Figure 3.13 we compare our algorithm with binary cutout tools on extracting the rabbit. Note that the results generated by Intelligent Scissors and GrabCut suffer from inaccuracies along the boundary and more severely, the "color bleeding" problem, where the boundary pixels represent a mixed foreground-background color due to partial pixel coverage (see the greenish boundary pixels in the binary results). In contrast, our system is able to fully extract the rabbit without such artifacts.

3.5 Discussion

We have demonstrated the first realtime tool for generating high quality mattes and composites as the user roughly paints along the foreground boundary. The scissor brush width and boundary conditions adjust intelligently as the user draws the scissor stroke. Our evaluation demonstrates that Soft Scissors outperform previous matting techniques both in quality and efficiency.

Currently we rely on the user to trace the foreground edge. One could imagine a hybrid system that first performs a quick binary segmentation to further guide the user and the underlying algorithms. However, for still images the speed and simplicity of the current approach may not warrant the complexity such an approach would add to the system. For video matting however, some hybrid of a fully automated and a user guided system will be needed. We feel the Soft Scissors approach can provide the basis for the user guided aspect of such a system.

Figure 3.8: Three examples. From left to right: original image, snapshots of extracting the matte in realtime, and the new composite.

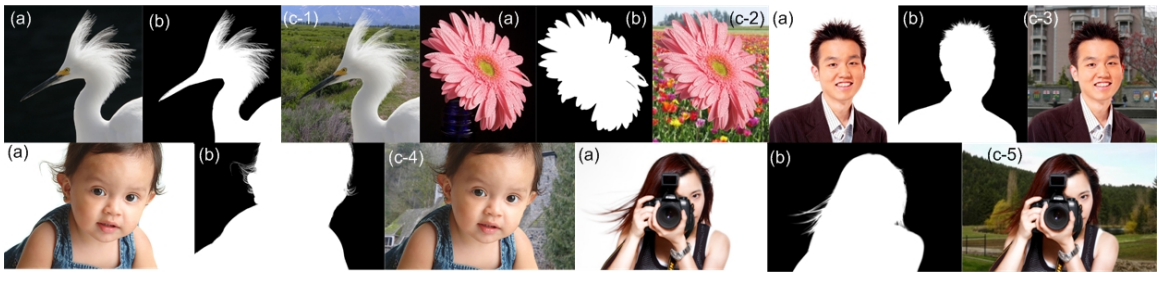


Figure 3.9: Test data set. (a). Original image. (b). Ground-truth matte. (c). Target image on which we apply matting algorithms.

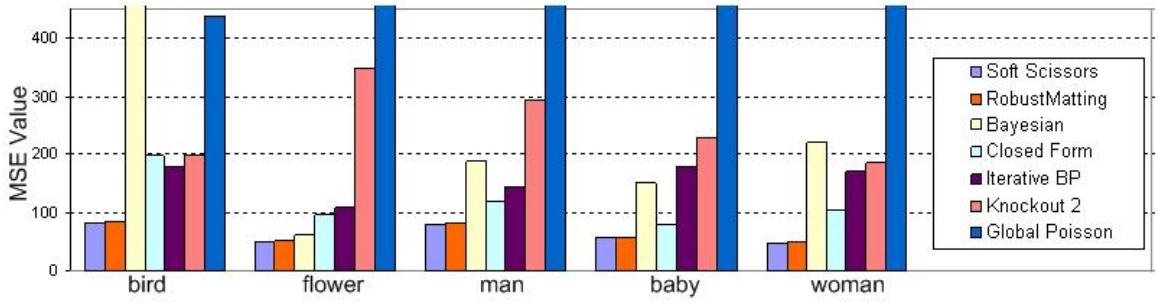


Figure 3.10: Comparing different algorithms on the data set in terms of matte errors.

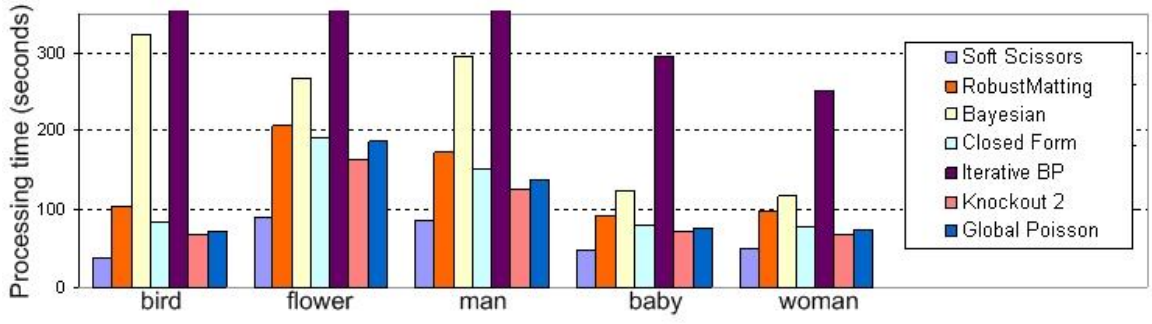


Figure 3.11: Comparing different algorithms on the data set in terms of processing time.

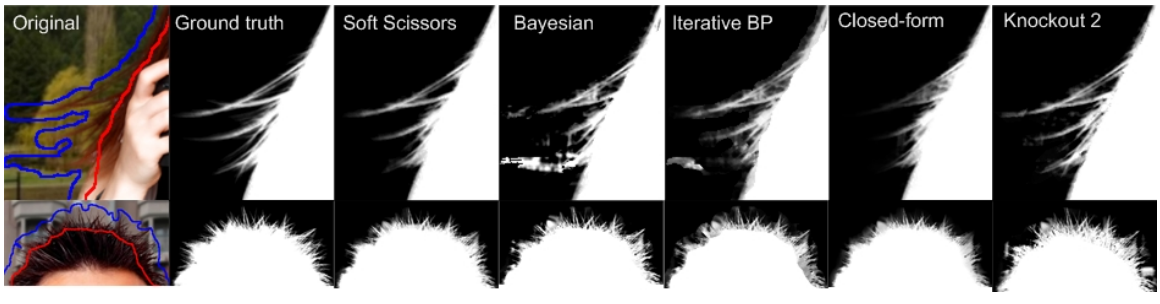


Figure 3.12: Partial results on test image “man” and “woman” in Figure 3.9.

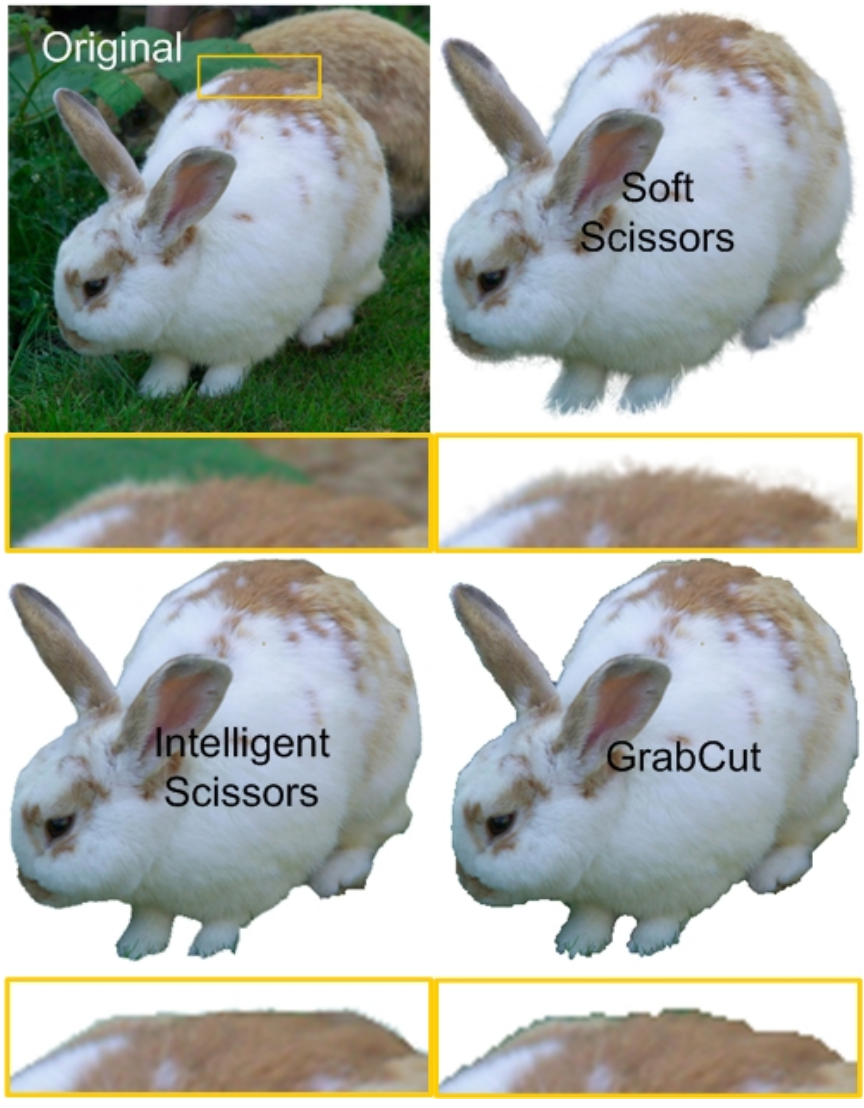


Figure 3.13: Comparing our system with Intelligent Scissors and GrabCut on extracting the foreground rabbit.

Chapter 4

COMPOSITIONAL MATTING

4.1 Introduction

We have demonstrated a Robust Matting algorithm in Chapter 2, and the Soft Scissors system which is built upon it for realtime matting in Chapter 3. Although the main purpose of these systems is to re-compose the foreground onto a new background, these approaches, as well as other previous matting systems, treat matting and compositing as separate tasks by assuming the new background is unknown. In this chapter we show that, by combining matting and compositing into a single optimization process, the matting algorithm can be more robust and efficient to create a successful composite. We dub this matting algorithm *compositional matting* [1]. An example is shown in Figure 4.5 where we use this algorithm to re-compose the girl onto a much nicer beach image. Compared with previous approaches, our algorithm creates the best composite with the least amount of visual artifacts.

Besides composing the foreground onto a new background, we also explore the advantages of the proposed algorithm for re-organizing and re-composing elements within a single image. A common photograph we have all seen contains a person standing in front a beautiful outdoor scene (for example, the image shown in Figure 4.4). Although the background is nicely framed, the foreground person often looks quite small since the focal length of the camera was set to capture the wide angle scene. This problem becomes more obvious as the image is resized to standard snapshot size or is displayed on a small mobile device, as the foreground character may shrink to the point of being unrecognizable. One could crop the foreground from the original image, however the composite becomes less interesting by losing the background scene.

Based on the compositional matting algorithm we allow the user to set different display ratios for foreground and background objects. As shown in Figure 4.4, the user roughly indicates the foreground by a few paint strokes, and our system is able to generate a novel version of the photograph

[1] This algorithm has been published in [136].

where the foreground is enlarged while the background remains as is.

4.2 Related Work

4.2.1 Foreground Segmentation

Many systems have been developed to accurately identify the foreground regions with minimal user guidance. Intelligent paint [107] and the object-based image editing system [6] first oversegment the image and then let the user select the regions that form the foreground object. LazySnapping [83] and GrabCut [110] systems provide interactive graph-cut-based segmentation solutions. The GrowCut system [130] employs cellular automation for interactive foreground extraction. In these systems users coarsely indicate foreground and background regions with a few paint strokes of the mouse and the system tries to determine the ideal boundary for segmenting the foreground.

4.2.2 Foreground Matting

Pixels on the edge of a foreground object usually contain some percentage of the background. These "mixed pixels" can create visual seams when composed onto a new background. Seamlessly composing a foreground object onto a new background requires estimating an opacity (alpha) value for each pixel as well as the foreground color. Ruzon and Tomasi [111] show how to estimate the alpha matte and foreground color using statistical methods. Chuang et al. [23] extend this approach by employing Bayesian framework for alpha estimation in both images and video. The Poisson matting approach [120] solves Poisson equations for matte estimation, under the assumption that the foreground and background colors are distinct and smooth.

Recently, the problem of foreground segmentation and matting is combined together in the Belief Propagation matting system[134], where an iterative optimization process is employed for estimating foreground matte from sparse user input such as a few scribbles. This quickly becomes a new trend in matting research and some new systems, such as the easy matting [53] and the closed-form matting system [81], have been proposed to achieve more robust and accurate matting results with sparse user inputs. These approaches have been described in detail in Section 2.2.

4.2.3 Image Compositing

Since our system composes the foreground object in a source image onto a new background image, it belongs to the general framework of "photomontage." Agarwala et al. have proposed an interactive framework [3] for this task by using graph-cut optimization. Their framework makes use of two techniques primarily: graph-cut optimization, to choose good seams within the constituent images so that they can be combined as seamlessly as possible; and gradient-domain fusion, a process based on Poisson equations, to further reduce any remaining visible artifacts in the composite. Also central to the framework is a suite of interactive tools that allow the user to specify a variety of high-level image objectives, either globally across the image, or locally through a painting-style interface. Image objectives are applied independently at each pixel location and generally involve a function of the pixel values (such as maximum contrast) drawn from that same location in the set of source images. Typically, a user applies a series of image objectives iteratively in order to create a finished composite. They show how this framework can be used for a wide variety of applications, including selective composites (for instance, group photos in which everyone looks their best), relighting, extended depth of field, panoramic stitching, clean-plate production, stroboscopic visualization of movement, and time-lapse mosaics. However, this system only generates hard segmentation of source images, and thus is not capable of handling partial foreground coverage.

The Poisson image editing system [106] uses generic interpolation machinery based on solving Poisson equations for composing a foreground region onto a destination region. This approach shows that given methods for crafting the Laplacian of an unknown function over some domain, and its boundary conditions, the Poisson equation can be solved numerically to achieve seamless filling of that domain. By using this method independently in each of the channels of a color image, seamless composite can be achieved. This approach works well when the destination region has a relatively simple gradient field, but is insufficient to handle highly textured regions which are common in many images and in our examples.

The recently proposed "Drag-and-Drop Pasting" system [69] improves Poisson image editing by finding optimal boundary conditions. A shortest closed-path algorithm is designed to search for the location of the boundary. However as shown in their examples, this system will always change the color of the entire foreground to blend it into the new background, which is often undesirable

when the foreground and background have a significant gap in depth.

In contrast, our system seamlessly composes foreground and background images by estimating a matte for the foreground region. The matte is optimized in a sense that it will minimize the visual artifacts on the result image, although it may not be the true matte for the foreground.

4.2.4 Image Retargeting

Image retargeting adapts images for display on devices different than originally intended. Liu and Gleicher have proposed a retargeting algorithm [90] by using non-linear fisheye-view warping to emphasize parts of an image while shrinking others. Like the previous cropping-based approaches, this method automatically identifies important aspects of the source image to emphasize. However, rather than completely discarding less important aspects of the image, it uses a non-linear image warping function that de-emphasizes them. The idea of using an idealized fisheye warp is a common method in information visualization to provide a "focus+context" display that both emphasizes the focus area, while retaining its context. Although this method can fit a large image into a small screen, both the foreground and background are very distorted. Instead, we set different display ratios for the foreground and background to achieve the retargeting goal, and only alter a small region around the foreground to eliminate the compositing seams without significant distortion.

4.3 The Compositional Matting Algorithm

As shown in the matting Equation 1.1, the observed image I_z ($z = (x, y)$) is modelled as a convex combination of foreground image F_z and background image B_z. In this work we assume the new background image B', onto which the extracted foreground will be composed onto, is known. The final composite thus can be calculated as

$$I_z^* = I_z + (1 - \alpha_z)(B'_z - B_z) \tag{4.1}$$

We can see that the final composite I_z^* is determined by two unknowns: α_z and the original background color B_z. As we will show later, our algorithm starts estimating the matte from the outside of the foreground where $B_z = I_z$, thus mostly the composite will only affected by the single unknown α_z, and the goal of our system is to estimate α_zs to minimize the visual artifacts of I_z^*s.

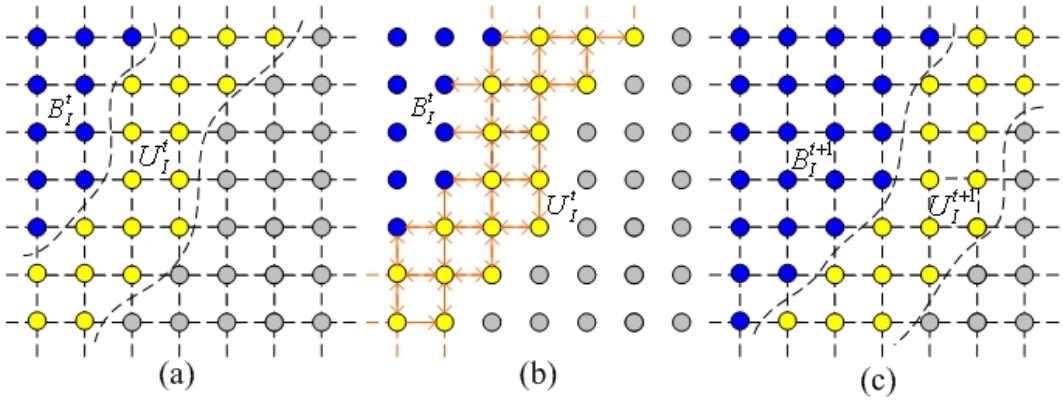


Figure 4.1: Our algorithm solves the composite in a front-propagation fashion. (a). Dilating background region B_I^t (blue) to create an unknown region U_I^t (yellow). (b). Finding the composite for U_I^t by optimization with boundary conditions. (c). Regions are evolved for the next iteration.

The most obvious advantage B' provides is that if the new background has very similar regions to the original one, instead of extracting the true foreground matte in these regions which can be erroneous, we can find a good transition between the old background and new background for a good composite. In other words, our matte can be conservative and thus some of the original background is carried into the composed image with the foreground.

4.3.1 Front Propagation for Matte Estimation

In our system the user first roughly indicates the foreground object on image I by specifying a bounding box R_I around it, and a few paint strokes on it, as shown in Figure 4.4a. We treat the region outside R_I as the initial background region B_I, the region under paint strokes as the initial foreground region F_I, and all other pixels uncovered as the initial unknown region U_I.

The matte M_I is estimated iteratively. In each iteration B_I is dilated to create a narrow band region U_I^t and update the matte in this region. In early iterations many pixels in U_I^t will have estimated alpha value of 0 since they are very likely to be background pixels. We then expand B_I based on the updated matte and dilate it again to create a new unknown region U_I^{t+1} for the next iteration. The algorithm stops when $U_I^{t+1} \approx U_I^t$. One can imagine that the unknown region is shrinking until becoming stable, as illustrated in Figure 4.1. The front propagation can stop

immediately whenever it finds a good composite, thus can avoid visual artifacts resulting from inaccurate alpha values and foreground colors for real foreground pixels.

4.3.2 Cost Functions

In this section we describe how we set up the optimization objective in each iteration. A pixel z in the unknown region U_I^t has two possible modes. The first is that the new background color B'_z is very different from the original color I_z, thus an accurate alpha value should be estimated for z in order to fully substitute B_z with B'_z. We call this mode *matting mode*, and design a *matting cost* for it. The second mode, which we call *compositing mode*, is that B'_z is very close to I_z, thus it is a good place for transiting from the original image to the new background image. A *compositing cost* is defined for this case.

Matting Cost

The matting cost is designed to accurately estimate alpha values. In this case, the new background is simply ignored thus the problem degrades to the traditional matting problem. Inspired by recent optimization approaches for image segmentation and matting [134, 110, 83], our matting cost contains two terms: a *data term* which measures the alpha value based on the pixel's own observations, and a *neighborhood term* which enforces the alpha estimations to be consistent in a local neighborhood area.

Specifically, we sample a group of known background colors from the neighborhood of z, and use them to estimate a Gaussian distribution $G(\overline{B_z}, \Sigma_z^B)$. Since the user marked foreground pixels are usually far away, we use the global sampling method proposed in [134] to gather a group of foreground colors for z, and estimate a Gaussian distribution on them as $G(\overline{F_z}, \Sigma_z^F)$. An estimated alpha value α_z^m is calculated as

$$\alpha_z^m = \frac{d(I_z, \overline{B_z}, \Sigma_z^B)}{d(I_z, \overline{F_z}, \Sigma_z^F) + d(I_z, \overline{B_z}, \Sigma_z^B)} \tag{4.2}$$

where $d(I, \overline{C}, \Sigma)$ is the Manhattan distance from color I to a Gaussian $G(\overline{C}, \Sigma)$. The data term is then defined to be

$$D_z^{mat} = (\alpha_z - \alpha_z^m)^2 \tag{4.3}$$

It has been approved in previous approaches that enforcing proper neighborhood constraints can help estimate accurate alpha mattes. The Poisson matting [120] approach enforces the matte gradient to be proportional to the image gradient. The closed-form matting system [81] set the weights between neighboring pixels based on their color differences computed from local color distributions. In this work we choose to use this method for setting the neighborhood term due to its simplicity and efficiency.

Formally, the neighborhood term is defined as

$$J_z^{mat} = \sum_{v \in w_z} (\alpha_v - a_z I_v - b_z)^2 + \epsilon a_z^2 \tag{4.4}$$

where w_z is a 3×3 window around z, and a_z and b_z are two coefficients which can be eliminated later in the optimization process. ϵ is a regularization coefficient which is set to be 10^{-5} in our system. More details and justifications of this neighborhood term can be found in [81].

The total matting cost is defined by combining the data term and neighborhood term together as

$$C_z^{mat} = w^{mat} \cdot D_z^{mat} + J_z^{mat}. \tag{4.5}$$

where w^{mat} is a free parameter in the system which balances the data term and the neighborhood term.

Compositing Cost

The compositing cost is defined for image regions where the new background matches well with the original image. Similar to the matting cost, the compositing cost also contains a data term and a neighborhood term. Roughly speaking, the data term determines where to make a transition between the original image and the new background image, and the neighborhood term enforces the transition to be smooth.

Since I_z(which equals to B_z most likely) and B'_z are very close, we can ignore the true alpha of z and make a direct transition from the new background to the foreground image. To do this we examine the color difference between I_z and B'_z, and calculate a transition probability as

$$p_z = exp\left(-\left(B'_z - I_z\right)^T \Sigma_z^{I-1} \left(B'_z - I_z\right)\right), \tag{4.6}$$

where Σ_z^I is the local color variance computed from a 3×3 window centered at z in the original image.

The data term in the compositing cost is then defined as

$$D_z^{cmp} = (\alpha_z - p_z)^2. \tag{4.7}$$

If the color difference between B'_z and I_z (normalized by local color variance) is small, the transition probability p_z will be large, which encourages a high alpha value for z to force the front propagation to stop here.

The neighborhood term is defined as

$$J_z^{cmp} = \sum_{v \in w_z} \left[1 - exp\left(-d_t(z,v)\right)\right](\alpha_v - \alpha_z)^2, \tag{4.8}$$

where $d_t(z,v)$ is the *transition distance* between z and v which is calculated as $[d(I_z, B'_z, \Sigma_z^I) + d(I_v, B'_v, \Sigma_v^I)]/2$. Minimizing J_z^{cmp} will force the transition happens only at places where the transition distance is small, or in other words, only at places where the new background pixels are very close to the original ones. This neighborhood term is similar to the one defined in the photomontage system[3] for seamless compositing, however in the photomontage system the alpha values can only be either 1 or 0.

By combining the data term and the neighborhood term together, the compositing cost is defined as

$$C_z^{cmp} = w^{cmp} \cdot D_z^{cmp} + J_z^{cmp}, \tag{4.9}$$

where w^{cmp} is another balancing parameter of the system.

Combined Cost

For an unknown pixel z, we classify it to be in either matting mode or compositing mode based on its local image characteristics. The classification is achieved by defining a function δ_z as

$$\delta_z = \begin{cases} 1 & : \quad \alpha_z^m > \alpha_z^c \\ 0 & : \quad \alpha_z^m < \alpha_z^c \end{cases} \tag{4.10}$$

The total cost for tth iteration is defined as as

$$\begin{aligned} C^t &= \sum_{z \in U_I^t} \left\{ \delta_z C_z^{mat} + (1 - \delta_z) C_z^{cmp} \right\} \\ &= \sum_{z \in U_I^t} \Big\{ \delta_z [w_z^{mat} \cdot D_z^{mat} + J_z^{mat}] \\ &\quad + (1 - \delta_z)[w_z^{cmp} \cdot D_z^{cmp} + J_z^{cmp}] \Big\} \end{aligned} \tag{4.11}$$

For better justification of the total cost we defined, we can examine some extreme situations. If the new background B' is totally different from the original image I, then δ_zs will be all ones since α_z^cs defined in equation 4.6 will be very small. Then only the matting cost will be minimized and the compositional matting degrades to a general matting algorithm, which can generate competing (and often times better) mattes compared with previous matting approaches. Similarly, if B' is very close to I' over the whole image, thus only the compositing cost will be minimized and our system works in a similar way as the photomontage system [3]. However, since our system allows a smooth alpha transition, it can generate more smooth composites than the photomontage system. For most of the cases, our system performs mixed matting and compositing.

4.3.3 Optimization

As we can see that since each term in equation 4.11 is defined as a quadratic function over alpha values, minimizing the total cost could be achieved by solving a sparse linear system.

Specifically, the matte is calculated as

$$\alpha = argmin\ \bar{\alpha}^T L \bar{\alpha}, \quad s.t.\ \alpha_{bi} = s_{bi} \tag{4.12}$$

where $\bar{\alpha} = [1, 0, \alpha_{z1}, ..., \alpha_{zn}, \alpha_{b1}, ..., \alpha_{bm}]^T$, and $z1, ..., zn$ are all pixels in U_I^t, and $b1, ..., bm$ are boundary pixels. The s_{bi} represent the boundary condition. In our system if the boundary pixel bi is on the outer boundary of the unknown region then $s_{bi} = 0$, otherwise it is set to be $\max(\alpha_{bi}^m, \alpha_{bi}^s)$.

L is a sparse, symmetric, positive definite Laplacian matrix with dimensions $(2+n+m) \times (2+n+m)$, given by

$$L(1, \alpha_z) = -\delta_z w_z^{mat} \alpha_z^m - (1 - \delta_z) w_z^{cmp} \alpha_z^c \tag{4.13}$$

$$L(0, \alpha_z) = -1 - L(1, \alpha_z) \tag{4.14}$$

$$L(\alpha_z, \alpha_v) = -\Big\{(1 - \delta_z\delta_v)[1 - exp(-d_t(z, v))] + \sum_k^{(z,v)\in w_k} [\frac{\delta_z\delta_v}{9}(1 + (I_z - u_k)^T(\Sigma_k + \frac{\epsilon \cdot I_3}{9})^{-1}(I_v - u_k)]\Big\} \tag{4.15}$$

$$L(\alpha_z, \alpha_z) = -\Big[L(1, \alpha_z) + L(0, \alpha_z) + \sum_{v \neq z} L(\alpha_z, \alpha_v)\Big] \tag{4.16}$$

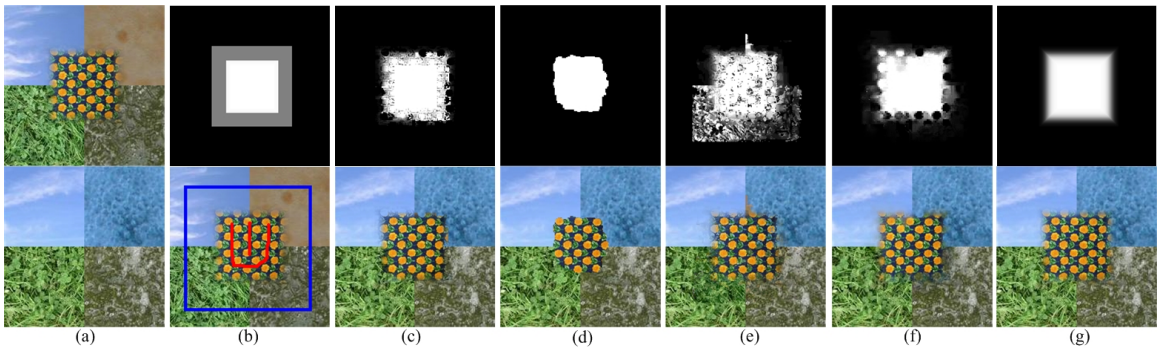


Figure 4.2: (a). The original image I(top) and the new background I'(bottom). (b). User input: a trimap for Bayesian matting (top) and paint strokes for scribble-based systems. (c)-(f): Matte and composite created by Bayesian matting(c), photomontage(d), iterative BP matting(e) and closed-form matting(f). (g). The ground truth matte and composite.

Specifically, $L(1, \alpha_z)$ and $L(0, \alpha_z)$ are derived from the data terms in Equation 4.11, and $L(\alpha_z, \alpha_v)$ is derived from the neighborhood terms in Equation 4.11. Note that the first term in $L(\alpha_z, \alpha_v)$ is derived from J_z^{cmp}, and the second term is derived from J_z^{mat}, which is provided in [81]. In this term $u_k(3 \times 1)$ and $\Sigma_k(3 \times 3)$ are color mean and variance in the local window, and I_3 is the 3×3 identity matrix. Note that the parameters a and b in Equation 4.4 have been eliminated in the final optimization, as demonstrated in [81].

The definition of $L(\alpha_z, \alpha_z)$ ensures that every row of L sums to zero. In contrast to the "matting Laplacian" defined in [81], we call the matrix L defined here as *compositional matting Laplacian*.

Once the matte converges, we use locally sampled background colors to estimate a background color B_z for a pixel z whose alpha value is between 0 and 1, and then use Equation 4.1 to generate the final composite.

4.4 Comparisons

We first compare the proposed algorithm with previous matting and compositing approaches on a synthetic example shown in Figure 4.2. More comparisons on real images will be shown later. Figure 4.2a shows the original image I where the foreground texture is composed onto a background texture using a pre-defined matte. Below this is the new background image B' where the bottom half is similar but the top has changed.

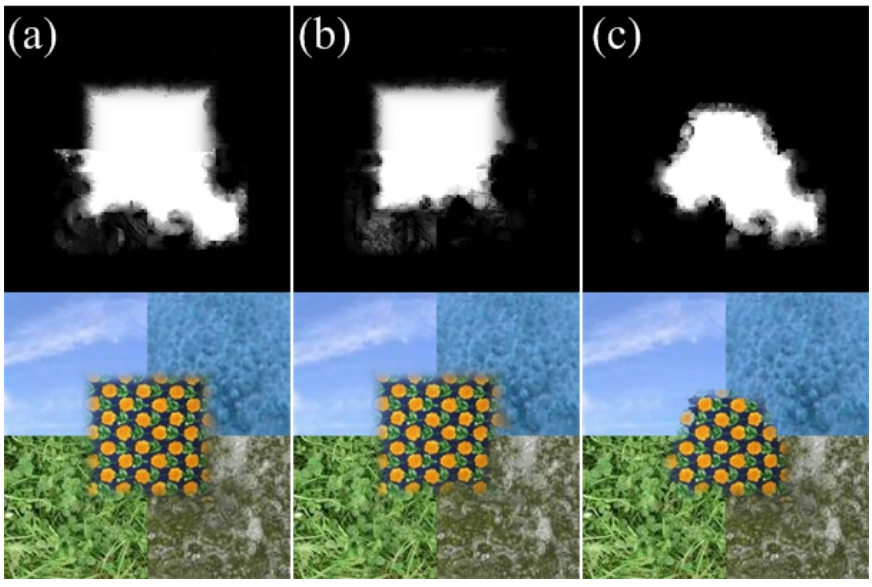


Figure 4.3: (a). Matte and composite created by our system. (b). The results when the compositing cost is disabled. (c). The results when the matting cost is disabled.

Figure 4.2b-e shows that previous approaches such as Bayesian matting [23], photomontage [3], iterative BP matting [134] and closed-form matting [81] all have difficulties dealing with this data set. Given the complex foreground and background patterns, these algorithms fail to extract an accurate matte thus the final composites are erroneous.

Figure 4.3a shows the composite generated by our system, which has higher visual quality than composites created by others, and is quite similar to the ground truth shown in Figure 4.3g. This is achieved by implicitly treating different regions in different ways. For the bottom half of the image where old and new backgrounds are similar, the front propagation stops earlier when it finds a good transition, thus the hard problem of finding an accurate matte in this region is avoided. For the upper half of the image where the old and new backgrounds are different, our algorithm works in a similar fashion as traditional matting algorithms to try to extract a good matte for the foreground.

Figure 4.3b shows the results if we disable the compositing cost in Equation 4.11 by letting all δ_zs to be 1. And Figure 4.3c shows the results of disabling the matting cost. These results demonstrate that minimizing a single cost is not sufficient for creating a successful composite.

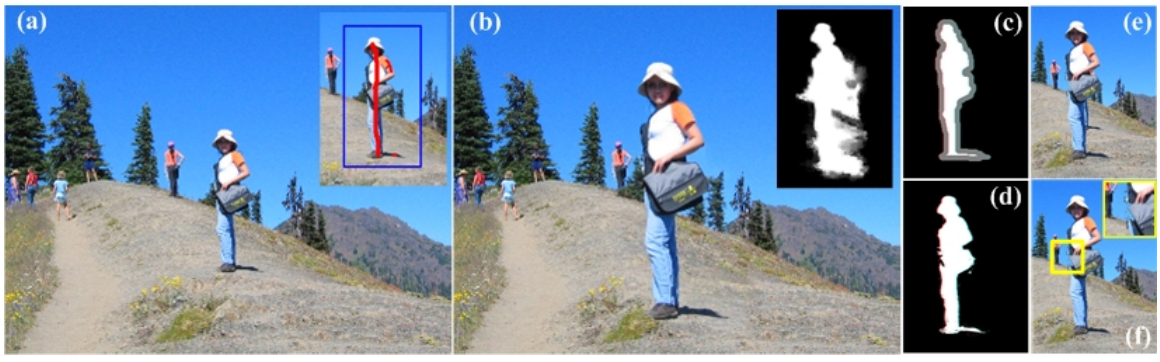


Figure 4.4: (a). Zoomed original image and user inputs. (b). Novel composite created by our system. (c). Trimap for Bayesian matting. (d). Matte computed by Bayesian matting. (e). Composite with matte in d. (f). Composite with matte in d after hole filling still contains visual artifacts.

4.5 Foreground Zooming

We apply the compositional matting algorithm to the task of recomposing a single image by varying the size ratio between the foreground and background within a single image. In general, we set a higher display ratio for the foreground relative to the background to emphasize the foreground. In this case, both the image I and the new background B' are differently scaled versions of the original photograph. This result is similar to virtually pulling the foreground towards the camera, as shown in Figure 4.4.

Using previous methods, one could achieve this by first extracting a high-quality matte for the foreground. Then hole filling methods would be needed to repair the background. One could then compose a scaled up version of the foreground matte onto the background. However, extracting a perfect matte for the foreground is difficult for general images, as is hole filling, and the composed image may contain visual artifacts.

For example, to attempt to use Bayesian matting to enlarge the foreground in Figure 4.4a, the user needs to specify a good trimap as shown in Figure 4.4c, which generates the matte in Figure 4.4d and results in a composite in Figure 4.4e. We can see “ghost” artifacts since the enlarged foreground does not fully cover the original foreground. We then use the image inpainting technique proposed in [142] to fill the holes on the background, resulting in a better composite in Figure 4.4f. However the errors in the matte estimation still cause noticeable visual artifacts.

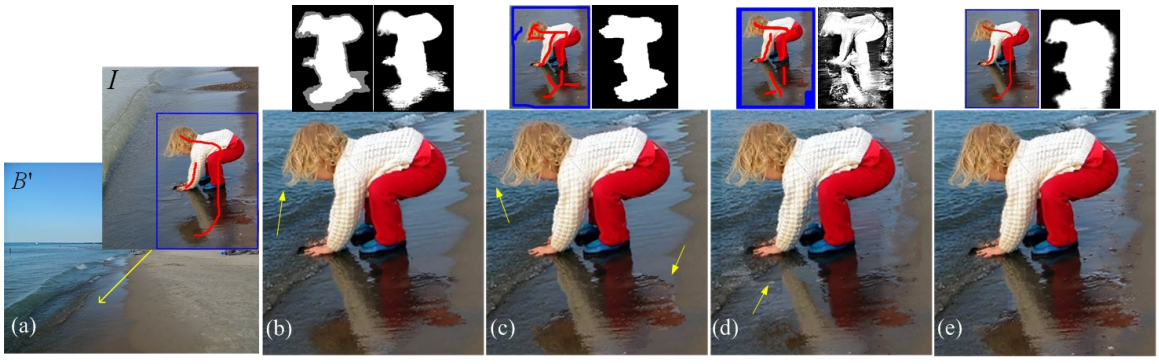


Figure 4.5: (a). I, B' and user input. (b)-(e). Input strokes, extracted matte and composite of Bayesian matting, photomontage, iterative matting and our system, respectively. Yellow arrows highlight artifacts.

In our system instead of using image inpainting techniques to fill-in the holes, we simply modify one step of our algorithm to avoid introducing holes as shown in Figure 4.4c. Once we calculate α_z^m in Equation 4.2 for pixel z, we find its corresponding location z' on the new background B'. If $\alpha_{z'}^m$ is smaller than α_z^m, we then let $\alpha_{z'}^m$ equals to α_z^m. In other words, we set high alpha values to pixels insides holes to encourage them to be occluded in the final composite. In this way our system achieves hole filling, foreground matting and compositing in a single optimization procedure.

4.6 More Results

Figure 4.5 compares different approaches on extracting the foreground from I and composing it onto B'. It shows that our system is able to create a more satisfying composite than previous approaches. Additionally, Figure 4.6 shows that if we use a totally different new background such as a solid blue, our system will try to extract an accurate matte since no useful new background information can be used in this case. This demonstrates that our system will work just as a normal matting algorithm when the new background is different from the original one instead of generating unpredictable results.

Figure 4.7 shows another example where we want to create a more impressive waterfall from the original one. We stretch the waterfall in the horizontal direction and recompose it onto the original image. Using previous matting approaches to achieve this is particularly hard since the foreground

Figure 4.6: The matte and composite generated by our system when the new background is provided as a solid blue.

object is semi-transparent thus creating a trimap for matting is erroneous. Instead, we compare our system with the photomontage system. Since photomontage cannot deal with partial coverage, the composition generated from it contains more visual artifacts than the one generated from our system.

Figure 4.8 compares our system with the image retargeting system [90]. In the zoomed out fish-eye image created by the image retargeting system the mountain behind the person is unacceptably distorted. As shown in Figure 4.8c, a true fish-eye image is even worse since the foreground character is also unacceptably distorted. In contrast, our system is able to enlarge the foreground while keeping both the foreground and the background in as original a state as possible.

Although our system works well on most of the examples we have tested, it does not always give satisfying composites. When the new background differs significantly from the original one, the compositional matting faces the same difficulties as traditional matting algorithms do. Difficult and successful examples are shown in Figures 4.10 and 4.9. In the original image, Figure 4.9a, the foreground is so similar to the background that extracting an accurate matte is almost impossible. However, if the new background is similar to the original one, our system is able to create a good composite as shown in 4.9d. If unfortunately, the new background is substantially different from the original one, our system along with previous approaches all fail to give good composites, as shown in Figure 4.10.

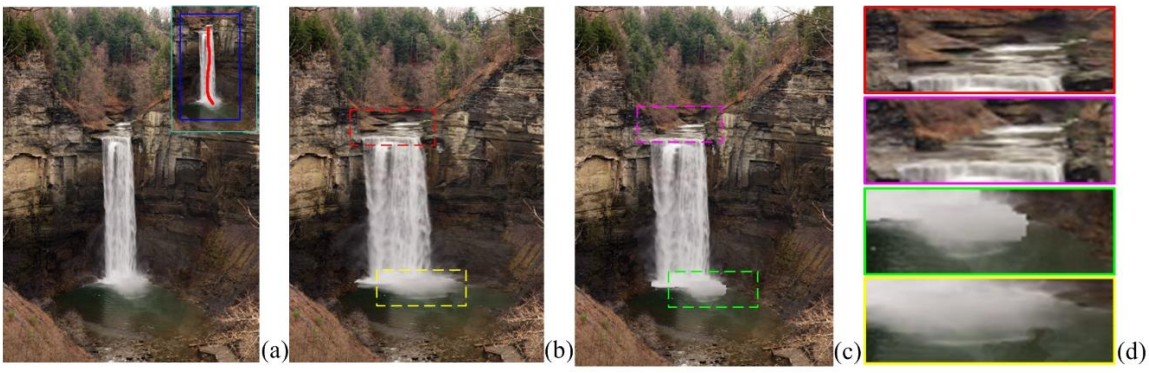


Figure 4.7: (a). Scaled original image with user inputs to expand the waterfall. (b). Composite generated by our system. (c). Composite generated by the photomontage system. (d). Details of composites.

4.7 Discussion

A key lesson to take from this work is that such image processing methods should take advantage of all information known in a real application. Matting in the absence of the knowledge of the new background may not describe the full task.

In this chapter we have demonstrated a compositional matting algorithm by taking the advantage of knowing the new background image onto which the foreground is to be composed. Experimental results show that our algorithm outperforms previous proposed matting and composition algorithms when the new background has similar regions with the old one. Based on the new matting algorithm we show how to recompose images by displaying foreground and background with different scales.

In the future we hope to consider how one might take temporal coherence into account for recomposing video objects. One could create a similar unified optimization framework, but computational considerations would certainly need to be addressed.

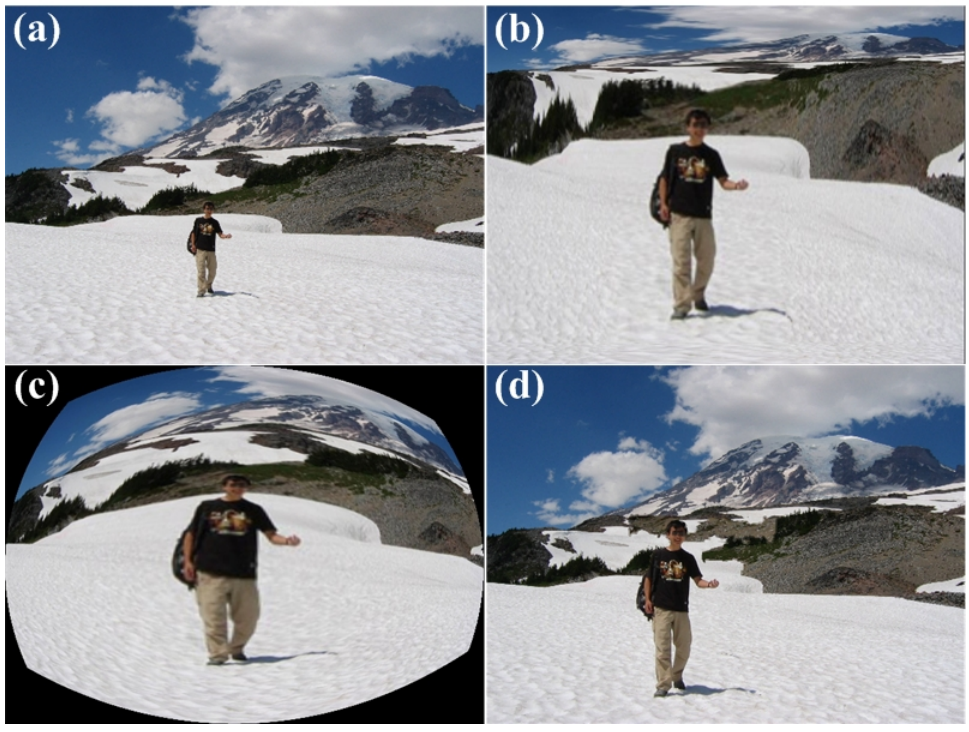


Figure 4.8: (a). Scaled original Image. (b). Simulated fish-eye image used by the image retargeting system. (c). A normal fish-eye image. (d). Enlarging foreground by 1.9 times using our system.

Figure 4.9: (a). Original image and matte extracted by Beyesian matting. (b). Composition created by Bayesian matting. (c). Composition created by our algorithm.

Figure 4.10: A failure example. From left to right: composing the foreground in Figure 4.9a onto a substantially different background using our system; matte extracted by our system; composition created by Bayesian matting.

Chapter 5

INTERACTIVE VIDEO CUTOUT

5.1 Introduction

In previous chapters, we have presented new innovations for interactively extracting foreground objects from still images. In this chapter we target the more challenging problem of video segmentation, and propose a novel system to extract foreground objects from video sequences.

One could imagine a simple solution that operates on each frame of a video individually using image-based techniques, but these would fail in two ways. The requirements for user interaction would become overly tedious. In addition, slight differences in the extraction from frame to frame will lead to a lack of temporal coherence, resulting in a jittery video.

We thus will operate on the video as a whole, which presents many new challenges ranging from the large number of pixels in a video to deal with, to difficulties in providing intuitive interfaces for a user to indicate regions of interest that move and deform significantly through time. On the other hand, within a single shot, a video provides some advantages since each frame can be expected to be highly correlated with other frames. We will take advantage of this fact when possible in constructing our solution.

We develop an interactive system for efficiently extracting foreground objects from a video[1]. The system we present for video foreground extraction offers a number of contributions over previous work:

- We present a novel user interface for allowing the user to indicate foreground and background regions within the video by painting of surfaces within the spatio-temporal volume.

- We introduce a hierarchical mean-shift over-segmentation preprocess to allow for solving global graph-cut optimizations in interactive time (a few seconds).

[1]The system has been published in [133].

- We define new spatially local color and edge models within a graph-cut framework combined with the global models in previous work.

- We extend the 2D alpha matting presented in [110] to the 3D spatio-temporal video object to preserve both spatial and temporal smoothness of the alpha matte.

We also present a number of applications for video editing once the foreground is extracted. The simplest of these is to place the foreground object on a new background. Once an object is extracted it can be broken into sub-objected by iterating on the process. It is then a simple matter to recolor each sub-object. We also have developed a Non-photorealistic rendering (NPR) system to stylize the extracted video objects using a variety of cartoon styles. Finally we apply free-form deformation to the 3D spatio-temporal object before reinserting it into a new (or the original) background to exaggerate the motion. These applications will be described in detail in following chapters.

5.2 Related Work

The problem of extracting foreground objects from images and video has been studied extensively over the last 20 years. We focus on several strands of previous work that are directly related to our interactive video cutout system.

5.2.1 Segmentation-Based Cutout

The earliest image cutout techniques allowed users to segment the image by choosing pixels that match a particular low-level image feature such as color. Photoshop's magic wand [66] takes this approach, letting users select pixels that match a range of colors. In these techniques the burden is on the user to choose the proper feature parameters in order to extract the foreground object.

Recently, we have seen some successful segmentation-based image cutout systems. Intelligent paint [107] first over-segments the image and then lets the user select the regions that form the foreground object. LazySnapping [83] and GrabCut [110] provide interactive graph-cut-based cutout solutions. The Lazy Snapping system is a novel coarse-to-fine UI design for image cutout which consists of two steps: a quick object marking step and a simple boundary editing step. The first step, object marking, works at a coarse scale, which specifies the object of interest by a few paint

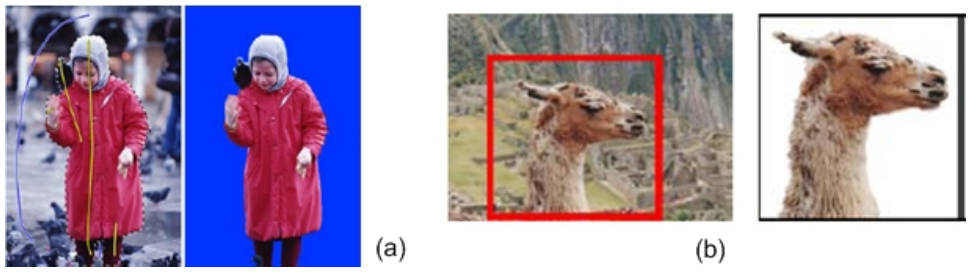


Figure 5.1: (a): The user interface of the LazySnapping system [83]. (b). The user interface ofthe GrabCut system [110].

strokes, as shown in Figure 5.1(a). The second step, boundary editing, works at a finer scale or on the zoomed-in image, which allows the user to edit the object boundary by simply clicking and dragging polygon vertices. This system inherits the advantages of region-based and boundary-based methods in two steps. The first step is intuitive and quick for object context specification, while the second step is easy and efficient for accurate boundary control. The GrabCut system first obtains a “hard” segmentation using iterative graph cut. This is followed by a border matting algorithm in which alpha values are computed in a narrow strip around the hard segmentation boundary. The novelty of this approach lies first in the handling of segmentation, where two enhancements have been made to the graph cuts mechanism: “iterative estimation” and “incomplete labelling” which together allow a considerably reduced degree of user interaction for a given quality of result. This allows GrabCut to put a light load on the user, whose interaction consists simply of dragging a rectangle around the desired object, as shown in Figure 5.1(b). Our work extends these techniques to the more challenging problem of video cutout.

Levin et al. [80] present a simple optimization technique for softly segmenting black and white video into regions of similar intensity for the purpose of colorization. Users set constraints on the segmentation using a painting interface.

Mean-shift clustering has been successfully used in the context of image segmentation [29]. The basic computational module of this technique is an old pattern recognition procedure, the mean shift. It proves for discrete data the convergence of a recursive mean shift procedure to the nearest stationary point of the underlying density function and thus, its utility in detecting the modes of the density. Image segmentation is described as an application of this technique. We extend this

technique from images to video and present a hierarchical mean shift approach to find relatively small video regions, and then use an interactive graph-cut optimization to extract the full foreground object. Our approach is much faster, and more importantly, leaves the user in control since the mean shift segmentation only provides efficiency, and does not constrain the final shape of the foreground object.

The Video Paintbox system [26] applies mean-shift to each 2D frame individually and then runs a heuristic matching algorithm based on color, area, overlap and shape to associate 2D regions across adjacent frames. While we borrow the idea of applying mean-shift to 2D frames individually, we go even further and apply mean-shift again to the 2D regions to produce 3D spatio-temporal regions. The Video Paintbox is designed for transforming video into NPR renderings and requires far less accuracy. We follow our initial segmentation with refinement and alpha matting to yield a more precise cutout.

5.2.2 Boundary-Based Cutout

Another way to extract a foreground object is to wrap a curve around its boundary. However, precisely drawing this boundary curve by hand can be extremely difficult. Boundary tracing techniques such as intelligent scissors [101], image snapping [83] jetstream [105] and Photoshop's magnetic lasso [66] use curve optimization methods to significantly reduce the accuracy required to trace a boundary. As long as the the user draws near the boundary these techniques will continuously optimize the curve and fit it precisely onto the boundary.

The process of tracking the boundary curve of an object through a video sequence is called *rotoscoping*. User-guided rotoscoping techniques [54, 93] allow users to trace curves every few frames and automatically interpolate between them. More recently Agarwala et al. [4] have combined an optimization technique [12] with user guidance to significantly reduce user effort. As shown in Figure 5.2, in order to track contours in video in Agarwala's system, the user specifies curves in two or more frames and these curves are used as keyframes by a computer-vision-based tracking algorithm. The user may interactively refine the curves and then restart the tracking algorithm. Combining computer vision with user interaction allows the system to track any sequence with significantly less effort than interpolation-based systems. Their tracking algorithm is cast as a

Figure 5.2: Agarwala's rotoscoping system [4] tracks outlines in a video sequence using a keyframe-based optimization algorithm. Images are from [4]

spacetime optimization problem that solves for time-varying curve shapes based on an input video sequence and user-specified constraints. However, this technique requires that the video contain strong edges between foreground and background and has difficulty with fast moving foreground objects.

Several groups have attacked the problem of automatically finding object boundaries using advanced camera hardware. Zitnick et al. [144] use multi-camera video sequences to reconstruct 3D information and then segment foreground objects aided by depth discontinuities. In this dissertation we focus on developing a software-only technique that can be applied to any video sequence regardless of the hardware used to acquire it.

5.2.3 Video as 3D Object

The user interface we describe relies on the idea of treating video at a 3D volume of data. Kwatra et al. [78] have applied graph-cut methods directly to a video volume for texture synthesis. Fels et al. [43] and Klein et al. [75] present approaches for interactively viewing video in the form of a 3D volume. The Proscenium [8] system uses the video cube paradigm for video editing. Users can manipulate the cube by slicing it with a cutting plane, and distort or warp the video to facilitate alignment. We leverage the idea of manipulating the video cube and allow users to indicate foreground objects by painting on arbitrary spatio-temporal surfaces in the video volume.

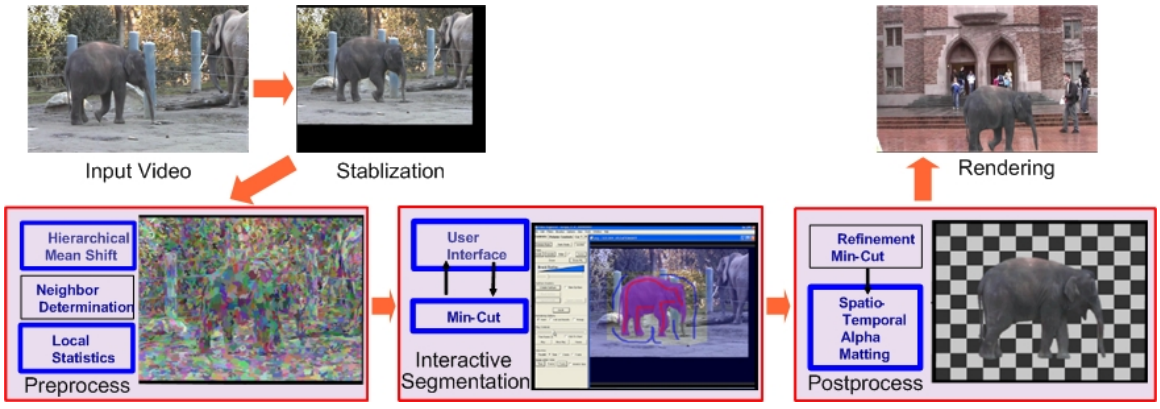


Figure 5.3: The framework of the video cutout system.

5.3 The System Overview

Figure 5.3 shows the framework of the proposed system. We assume that the input video has been stabilized either by using software stabilization tools [92], or by shooting the video with the aid of a tripod.[2] Our system processes the input video in three major stages: automatic preprocessing, interactive segmentation and automatic postprocessing[3].

Automatic Preprocessing. To ensure efficient updates during the interactive segmentation stage we pre-compute a hierarchical decomposition of the input video using a new version of the mean-shift clustering algorithm. We also compute neighborhood information between the clusters in the decomposition and local statistics over the entire video sequence. We have worked mostly with clips at 720x480 resolution and of 100 to 300 frames. The preprocess takes from 10 to 30 minutes.

Interactive Segmentation. In the interactive segmentation stage the user "paints" pixels to indicate that they are foreground or background. A key feature of our system is that it allows users to interactively rotate, slice and cutaway parts of the video volume to paint on pixels inside it. At any point in the interaction, the user can run a graph-cut optimization over the entire video to compute a segmentation based on the current set of painted hints. We have designed a new, efficient, version

[2]If there was considerable camera translation that induces parallax the video cannot be stabilized via software techniques. In such a case we invoke a less robust optical flow method to serve a similar purpose as discussed later.

[3]This system was built as a joint work of Pravin Bhat, Alex Colburn and myself. I contributed mostly to the automatic preprocessing and post-processing steps. Pravin Bhat contributed mostly to the interactive segmentation step, and Alex Colburn contributed mostly to the user interface.

of the graph-cut algorithm that runs on the hierarchical representation created in our preprocessing stage. After several seconds of computation the system presents a segmentation result and the user can add more paint strokes as necessary to refine the segmentation.

Automatic Post-processing. The segmented video then passes through an automatic foreground refinement and alpha matting step. The foreground boundary is first refined by a pixel-based graph-cut optimization constrained to a narrow region around the course segmentation. Then a spatio-temporal coherent foreground matte is extracted, which allows the extracted foreground to be seamlessly composed onto other backgrounds or used for a variety of other applications.

5.4 Preprocessing

Even short video sequences contain a large number of pixels (e.g., 100M pixels in a 10 second shot). The scale of the problem makes it computationally infeasible to run a segmentation algorithm at the pixel level. Instead we have developed a hierarchical algorithm that works with a hierarchical decomposition of the video volume. The goal of the preprocessing stage is to pre-segment the video into the hierarchical decomposition and compute local statistics on it so that the interactive segmentation stage runs efficiently.

5.4.1 Hierarchical Mean-Shift Segmentation

We use the mean-shift clustering algorithm [29] to build the hierarchical decomposition of the input video. As a general nonparametric density gradient estimator, mean shift is an old pattern recognition procedure proposed by Fukunage and Hostetler [47]. The underlying theory is kernel density estimation, also called the Parzen window technique, which is the most popular density estimation method. Recently, mean shift based image and video segmentation has gained much attention due to its promising performance.

The mean-shift algorithm clusters pixels that lie near one another in some *feature space*; typically the cross product of spatio-temporal position and color. However, the standard mean-shift procedure applied directly to pixels can be extremely costly both in computation time and memory, especially when the cluster regions are large.

To solve this problem, we have developed a new hierarchical mean-shift algorithm that first cre-

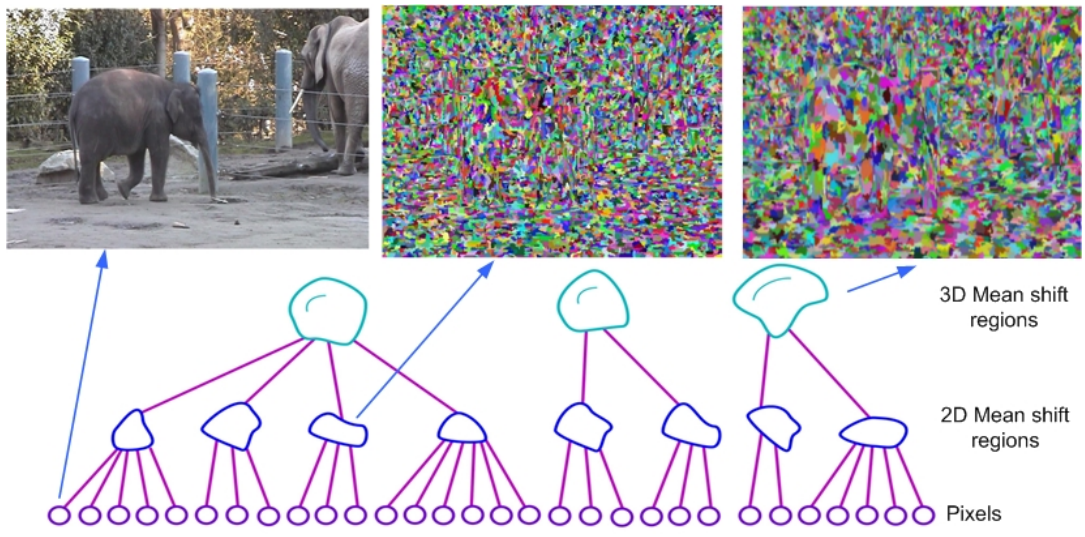


Figure 5.4: Illustration of the hierarchical mean shift segmentation.

ates small 2D regions (on average 100 pixels in size) on each frame of the input video. The complete collection of 2D regions, parameterized by their mean positions and colors, are then clustered into 3D spatio-temporal regions. On average, about 20 2D regions are merged into each 3D regions, thus a typical 3D region contains about 2000 pixels.

This procedure generates a strict hierarchy of regions so that each pixel belongs to a single 2D region and each 2D region belongs to a single 3D region. The exact coverage will ensure that each pixel is given exactly one label in the interactive graph-cut optimization. Figure 5.4 visualizes the structure of the hierarchy and also shows Pseudo-colored examples of 2D and 3D mean shift regions on a video frame (the 3D example shows a slice through the 3D mean shift region at the frame time).

By constructing the hierarchy in this two-step manner we avoid the prohibitive expense of running mean-shift directly on all pixels of the video. In our system, each 720x480 frame segments in approximately 2 seconds. The 3D clustering then operates on only about less than 1% of the number of nodes as pixels and thus completes in only 5 seconds per frame or about 10-15 minutes on a 150 frame sequence.

5.4.2 Neighbor Determination

In the preprocessing stage we also compute neighborhood relationships between all pairs of adjacent regions in the mean-shift hierarchy. At the pixel level, we consider each pixel to have 26 *links* to its neighbors, the 8 pixels surrounding it in its own frame, and the 9 adjacent neighbors in the frames immediately preceding and following it. The neighboring pixel links are implicitly encoded by the pixel ordering.

At higher levels in the hierarchy, we establish the neighborhood relationships between all 2D and 3D mean shift regions. For each region we record pointers to all neighboring regions. We also record number of pixels each region, and the number of pixel-to-pixel links between each pair of neighboring clusters.

5.4.3 Local Statistics

As we will discuss in section 5.5.2, we have developed new local cost models for the graph-cut process. To efficiently compute these local costs, we pre-compute statistics on *pixel-spans* and *link-spans* in the video (see Figure 5.5). A pixel-span is defined as the set of N pixels at some spatial location (x, y) through time (where N is the number of frames). Similarly, a link-span is the set of links that connect two adjacent pixel-spans. Note that the two pixel-spans may be the same, thus such a link-span contains only the direct temporal links between pixels in a single pixel-span. We collect the mean and variance of the gradients across each link in a link-span in each of the RGB color channels. We also compute and store local link costs as will be described in section 5.5.2.

5.5 Interactive Segmentation

The interactive segmentation loops between two stages. In the first stage, the user provides some indications of what is foreground and what is background. Then, a graph-cut optimization is invoked. The graph-cut takes about 10 seconds on a 720x480x200 pixel video. The user then examines the solution and adds some more hints if needed, and repeats the process until satisfied.

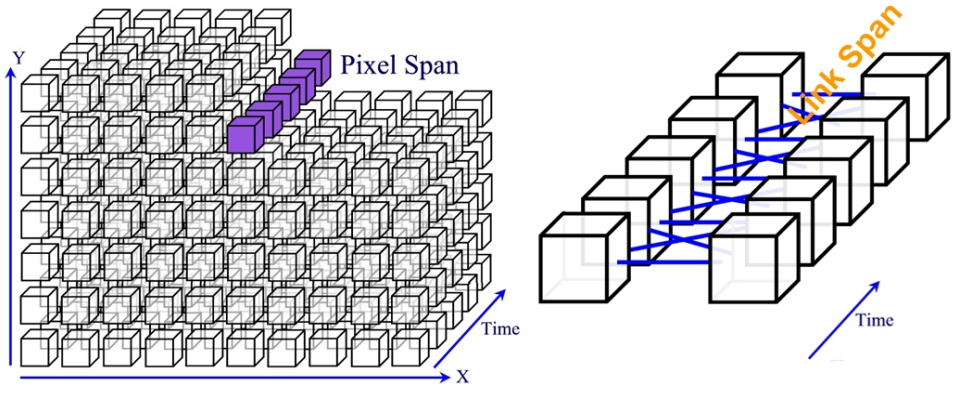


Figure 5.5: Left: all pixels in time at one spatial location are called a *pixel-span*. Right: all links in time across one spatial location are called a *link-span*.

5.5.1 The User Interface

With 2D images all the data are visible on the image plane. Users can directly access each pixel to indicate foreground and background. With 3D spatio-temporal video volumes, pixels in front occlude those in back and therefore we must provide tools for interactively accessing pixels that lie inside the volume.

As shown in Figure 5.6, our interface allows users to directly manipulate the video cube in a variety of ways to access pixels anywhere in it. Our system treats the video volume as a 3D cube of data where the X- and Y-axes represent the normal horizontal/vertical axes of a single frame, while the Z-axis represents time. Users can rotate the cube to view it from any angle, cut through the cube with cutting planes at any orientation and slice through parts of the cube by drawing arbitrary surfaces through it. These manipulations make it very easy to mark foreground (red paint) and background (blue paint) pixels within the volume using simple paint strokes as well as larger-scale volume fill operations.

Figure 5.6(a) is a screenshot of our user interface showing a video sequence of a skateboarder in the 3D video cube representation. The user has rotated the video cube representation with an arcball interface and the viewing plane is oriented parallel to the standard XY-plane. The user can switch to any orthogonal axis-aligned view, like the top-down view shown in Figure 5.6(b). In this case the user is looking at an XZ-plane passing through the volume. Our system provides interactive scrolling

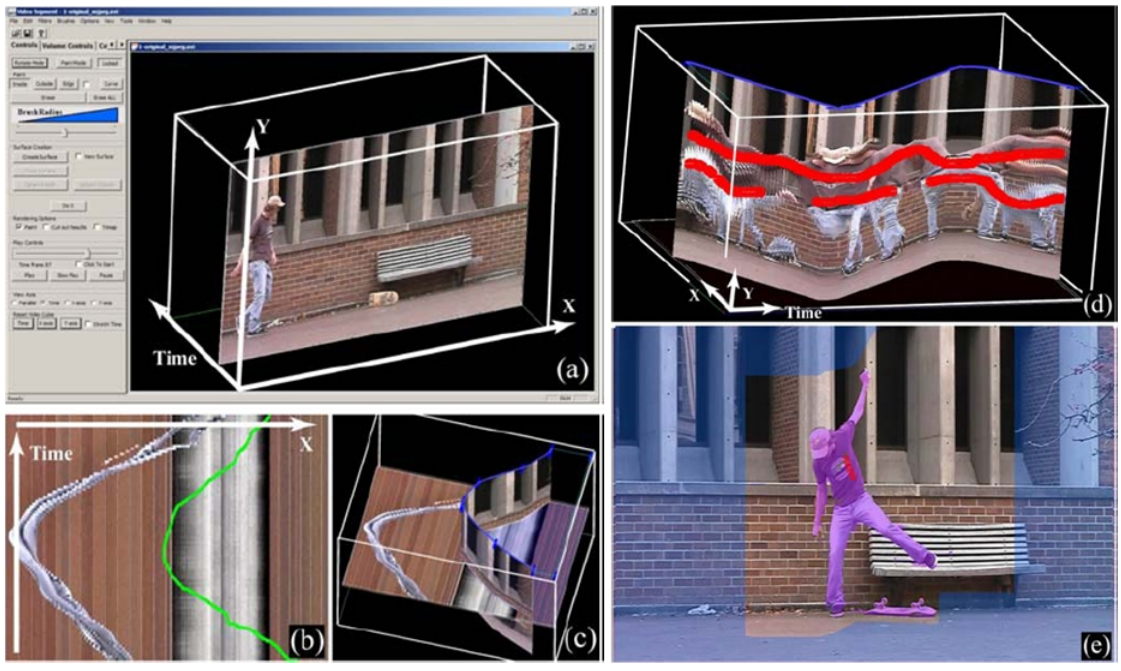


Figure 5.6: The user interface.

controls for orthogonally sweeping these through any such cutting plane through the volume. We use memory mapping to enable loading of very large videos that normally cannot fit in main memory. With memory mapping these large videos can be rotated, sliced, and viewed in realtime.

Spatio-temporal cutting planes are very useful for seeing the motion of objects over the entire video sequence in a single image. In Figure 5.6(b), the blue-gray C-shaped curve shows the motion of the skateboarder over time. The user can see that the skateboarder never appears on the right side of this view, and can confirm this hypothesis by scrolling the XZ-plane through the volume. To mark the entire right side of the video volume as background the user simply draws a curve (shown in green) across this XZ-plane. As shown in Figure 5.6(c), the system extrudes this curve through the volume. The user then fills the large portion of the volume to the right of the surface with background paint (the blue tint on XZ-plane indicates that it is marked as background). This operation is equivalent to setting a tight bound on the right side of the skateboarder in every frame. These large background regions are excluded from the graph-cut optimization to significantly speed up the optimization.

In Figure 5.6(d) we see an another extruded surface. This time the curve was drawn through the skateboarder in the top-down view. As a result he is spread out over much of the surface. By applying red, foreground paint over the skateboarder on this curved surface, the user quickly indicates foreground pixels across many parts of the spatio-temporal volume. Cutting the volume with such surfaces is especially useful for marking thin moving structures that may be difficult to paint in a standard video frame. Moreover, because graph-cut optimization is designed to regularize the shape of regions, such thin structures usually require more red foreground paint to be extracted properly.

The user can invoke the graph-cut optimization at any time by clicking a button in the interface. After several seconds of computation the resulting segmentation is shown with the foreground element tinted purple. Figure 5.6(e) shows an intermediate segmentation result for the skateboarder. The user can scroll through the entire video to visually check how well the foreground object is extracted, and add more paint as necessary. In this example, the user spots a hole at the edge of the skateboarder's shirt and adds red, foreground paint so that the graph-cut optimization will properly include the entire shirt in the next iteration.

5.5.2 Graph Cut

A graph cut optimization algorithm [14] is invoked to assign each pixel in the video to be either foreground (F) or background (B). To use graph-cut for video segmentation, we need to design two aspects of the problem:

(1). The graph topology, defined by two elements: nodes and bidirectional links. Based on the mean shift results, a node can be a pixel, a 2D spatial region on a frame or a 3D spatio-temporal region. The full collection of nodes in the graph should exactly cover the original video with no overlapping nodes. In other words, if a 3D (or 2D) region is selected for inclusion in the graph, none of its children should be selected. Similarly if a pixel (or 2D) node is selected, none of its ancestors should be included. Links are created between all pairs of "neighboring"nodes. Two nodes are neighbors if any pixel in one node is adjacent to any pixel in the other.

(2). The global cost function over the graph to minimize. This consists of two types of costs, *data costs* for assigning each node to either F or B, and *link costs* for each link. A link incurs a cost

only if the nodes at each end of the link are assigned different labels. We will explore these costs in some detail soon.

Given the graph topology and the data and links costs, the graph cut algorithm chooses a set of labels (F or B) for each node that minimizes the total labelling cost on the whole graph. Once nodes are labelled, the labels are propagated down to pixels for the final segmentation result.

Graph Topology

Figure 5.7 illustrates the hierarchy resulting from the mean shift pre-segmentation. All pixels are initially unlabelled and are indicated as white in the diagram. As the user paints pixels as either background (blue) or foreground (red) these labels are propagated up each tree structure. Nodes above the pixel label can take on both labels if their children are marked with conflicting labels.

At each invocation of the graph cut procedure a new graph is built dynamically. Each tree of nodes from the mean shift procedure is processed top down. When a node is found that has no conflicting labels, it is added to the graph, and its children are not considered. Thus, only the highest level nodes with a single color are added to the graph. These are shown with yellow rectangles in Figure 5.7. The set of nodes thus marked are guaranteed to be a minimal set of consistently marked nodes that exactly covers the video volume. For efficiency, higher level nodes fully painted as background are simply removed from further consideration (the gray box in Figure 5.7).

All pairs of nodes in the graph are then tested to see if they are neighbors. If so, a link is added to the graph connecting the pair. This completes the topological structure for the graph. In practice almost all nodes are the top level 3D nodes. In rare instances they broken by the user's actions when adjacent foreground and background colors are very similar. This process assures that the user can override any errors in the initial mean shift process.

Data and Link Costs

After the graph is constructed, the next step is to define a cost function E for the graph cut algorithm to minimize. We introduce in our work, the notion of combining the global costs from previous work with new local data and link costs within the graph cut framework. Figure 5.8 illustrates the structures of the cost function.

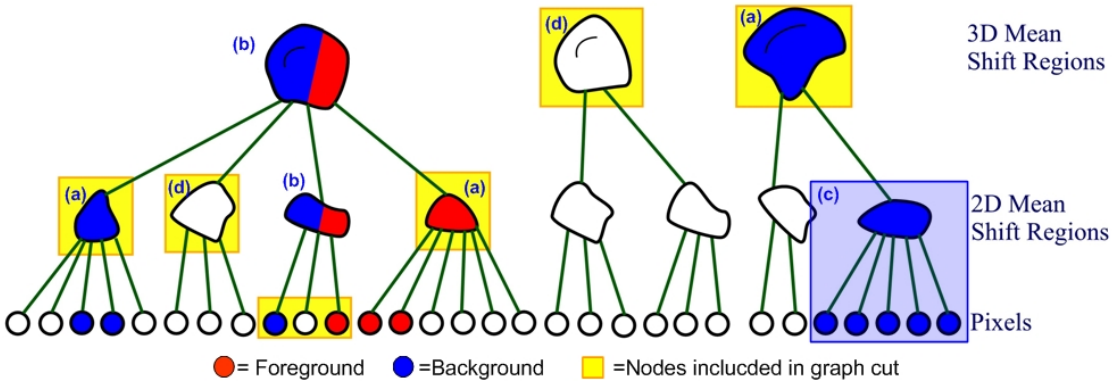


Figure 5.7: Our system dynamically builds graph on the video hierarchy for segmentation. (a) Nodes that contain only red or blue paint are fully constrained. (b) Mixed nodes are left out of the graph due to conflicting paint strokes in their children. (c) If an entire subtree is marked as background (highlighted in gray) it is removed from further consideration. (d) The graph-cut algorithm must assign a label to each white node in the graph. To apply graph-cut we construct a graph containing the highest level nodes of a single color (highlighted in yellow).

At the highest level, the total cost is represented as a Gibbs energy

$$E(X, Z, \Gamma) = \sum_i D(x_i, z_i, \gamma_i) + \lambda_1 \sum_{neighbors(i,j)} L(x_i, x_j, z_i, z_j) \tag{5.1}$$

Note: henceforth, for brevity, all the arguments will be left out unless they take on a specific value.

D is the data term which assigns a cost to label node i with label x_i, color z_i, and the user assigned label (if any) γ_i. In our case x_i equals either background, B, or foreground, F, and the user labels are either foreground $\mathcal{F}$ (often shown as red) or background $\mathcal{B}$ (blue) or $\emptyset$ (white).

L assigns a cost to each link between neighboring nodes i and j. This cost is 0 if the labels $x_i == x_j$. Otherwise, for the link cost is related to the color gradient, ∇, between nodes i and j.

Higher level nodes are composed of many individual pixels. The data and link costs between nodes must represent the full cost of assigning a complete node to a particular label or the full link of assigning two neighboring nodes to different labels. Thus, $D(i) = \sum_{p \in i} D(p)$ and $L(i,j) = \sum_{l \in (i,j)} L(l)$. In other words, the data costs are scaled by the volume of the node. The link costs are the sum of individual link costs at the pixel level over all links between the nodes. These quantities are computed and stored at preprocess time.

λ_1 balances the data and link costs.

Data Costs

As described in section 5.5.1 the user paints within the video volume to indicate specific pixels are either foreground or background. These values are then propagated up the mean shift hierarchy as shown in Figure 5.4. If a node i has been painted as background, $\mathcal{B}$, or foreground, $\mathcal{F}$, then

$$D(x_i = B, z_i, \gamma_i = \mathcal{B}) = 0,\ D(x_i = F, z_i, \gamma_i = \mathcal{B}) = \infty$$
$$D(x_i = B, z_i, \gamma_i = \mathcal{F}) = \infty,\ D(x_i = F, z_i, \gamma_i = \mathcal{F}) = 0$$

In other words, the user's indications are guaranteed to be respected by the graph cut.

For the vast majority of nodes, there is no direct indication from the user. In this case, individual data costs, D_B and D_F, are determined for assigning a node to background or foreground. These are then balanced as in previous work

$$D(x_i = B) = \frac{D_B}{D_B + D_F} \tag{5.2}$$

$$D(x_i = F) = \frac{D_F}{D_B + D_F} \tag{5.3}$$

As in [110] we use the pixels painted by the user to build two Gaussian mixture models (GMM), one for the colors found in the background and the other for colors found in the foreground. Our models use 5 Gaussians. A color of a node, z_i, (the mean color in the case of a higher level node) is tested against each GMM to return a likelihood of that color belonging to either the background or foreground. The global data costs, $D_{B,g}$ and $D_{F,g}$ are taken as the complements of the likelihoods.

$$D_{B,g}(x_i = B) = 1 - \sum_{k=1}^{5} w_k e^{(-(z_i-\mu_k)^T \Sigma_k^{-1}(z_i-\mu_k)/2)} \tag{5.4}$$

where the w_k are the weights corresponding to the fraction of samples closest to the k^{th} component of the GMM, and μ_k and Σ_k are the mean color and covariance of the k^{th} component of the GMM. The same equation applies to the global foreground model, $D_{F,g}$, except that the GMM is computed from those samples indicated as foreground by the user.

Unlike previous work, we also include a local background color model in our data cost. If a video was captured from a tripod, or has been stabilized, the background color will often vary little from frame to frame at any specific location. We extract a static clean plate from the video by

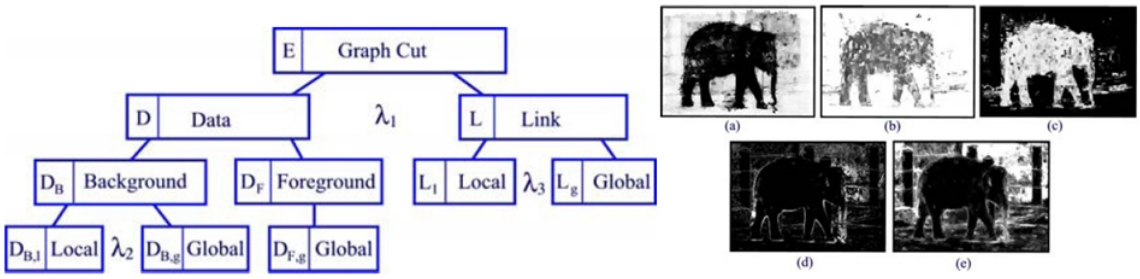


Figure 5.8: Left: graph-cut cost structure. The costs are described in Section5.5.2 with the symbols in the left of each box. Right: Visualizations of the Costs: (a) Local Background Data Cost ($D_{B,l}$), (b) Global Background Data Cost ($D_{B,g}$), (c) Global Foreground Data Cost ($D_{F,g}$), (d) Local Link Cost (L_l), (e) Global Link Cost (L_g) Note: white represents low cost in (a)-(c) and high cost in (d) and (e).

applying temporal filters [3] or using a simple median filter across all frames. However, for most sequences, even after video stabilization, the background is not exactly stationary due to camera shake, moving background objects, and camera noise. Therefore, we consider the pixels from the static clean plate as well as pixels marked by the user as background to build an additional local background color model. For a pixel, p, our local model is given by:

$$D_{B,l}(x_p = B) = 1 - e^{(-d(z_p)/(2\eta_{cp}^2))} \tag{5.5}$$

where the distance function $d(z_p)$ is taken as the minimum color difference between the pixel color, z_p, and the pixel in the clean plate or any pixel in its pixel span with user label $\mathcal{B}$. The term η_{cp} models noise and is set a priori. We have found that a value of 5 (in a 0-255 color space) to work well in practice. Since our local background color model cannot be evaluated directly for higher level nodes, we compute it as the sum over all child pixels. For efficiency we pre-compute the local background term at each pixel after generating the static clean plate.

The final background data term is the linear combination,

$$D_B(x_i = B) = \lambda_2 D_{B,l} + (1 - \lambda_2) D_{B-g} \tag{5.6}$$

where λ_2 reflects the confidence in the local background model. It is set locally based on the number, N, of $\mathcal{B}$ marked pixels in the span:

$$\lambda_2 = e^{(-(N+1)\cdot 100/NumFrames)} \tag{5.7}$$

Link Costs

Link costs are non-zero only when the nodes across the link are set to opposite labels, B and F. The link costs thus operate to try to keep the segmentation coherent across space and time.

Previous graph-cut based segmentation approaches make an a priori observation that the transitions between foreground and background will on average exhibit higher gradients than an average gradient between any two pixels. Therefore, they assign a static exponential function based on the color difference,, or gradient ∇, between the two nodes the link connects. We adopt this approach for our global link cost

$$L_g = e^{(-\nabla^2/(2\eta^2_{link}))} \tag{5.8}$$

where η_{link} represents the variance of the gradients across all links in the video. In practice we set it to 20.

In complex video sequences, both the foreground and the background may have complex patterns and strong edges which will encourage the segmentation to cut along edges inside the foreground object as well as within the background. We have developed a local link model to better handle such cases. We utilize the mean μ_∇ and variance σ_∇ of the link gradients in each linkspan. Our model is independent of the user's input and is computed once at preprocess time. We encourage a link to be cut if it has both a higher than average gradient in its span, and is an unusual occurrence in the span. Thus the local link cost is computed as

$$\begin{aligned} \text{if} \quad & \nabla_{ij} > \mu_{\nabla ij} \\ & L_l(x_i \neq x_j) = e^{((\nabla_{ij} - \mu_{\nabla ij})^2/2\sigma^2_{\nabla ij})} \\ \text{else} \quad & 1 \end{aligned} \tag{5.9}$$

The global and local link costs are also taken as a weighted average $L(x_1 \neq x_2) = \lambda_3 \cdot L_g + (1 - \lambda_3)L_l$. In practice we set λ_3 to 0.3.

Figure 5.8 also shows visualizations of the various costs on a frame of an elephant video. Note how the local costs, (a) and (d) help to isolate the foreground and the best links across which to cut.

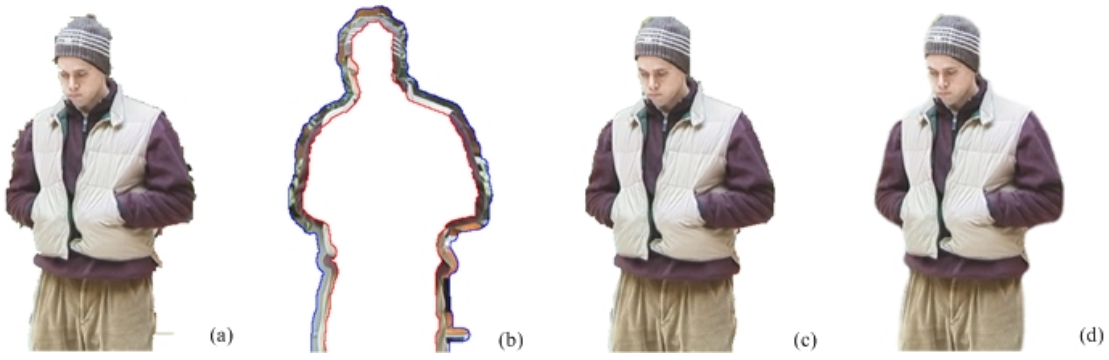


Figure 5.9: (a) Initial segmentation result of a frame. (b) Applying pixel-based graph-cut segmentation with local color models in the boundary area. (c) Refined foreground boundary. (d) Final alpha matting result.

5.6 Post-processing

The interactive segmentation process results in a coarse spatio-temporal foreground region. However, these regions are often noisy both spatially and temporally. An example is shown in figure 5.9(a). In an automatic post-process, we first refine the foreground boundaries and then determine a coherent spatio-temporal alpha matte.

5.6.1 Refinement Graph-cut

To refine the foreground boundary produced by the interactive segmentation we run a pixel-level graph-cut optimization within a narrow band of the initial boundary. We first erode and dilate the initial boundary by 3 pixels to remove small holes and thin, thread-like structures. We then build a pixel-level graph containing all pixels within a narrow spatio-temporal band of the boundary, as shown in Figure 5.9(b). The width of this boundary band is set to 10 pixels spatially and 1 frame temporally.

We construct the graph by treating each pixel within the band as a node, and each node is connected to its immediate spatial/temporal neighbors. If a pixel is on the inner boundary of the band it is set to $z = F$, and likewise outer boundary pixels are set to $z = B$ (shown as red and blue pixels in figure 5.9(b)). Otherwise, we set the data cost of each unknown pixel based on its foreground and background likelihoods computed locally, using the Bayesian image matting

approach of Chuang et al. [23]. We sample the 16 closest known foreground pixels from a local spatio-temporal neighborhood, and use these pixels to estimate a Gaussian foreground color model to compute the foreground data cost. We do the same thing to estimate a background color model and data cost for each pixel.

The link cost is set to a constant for all links in the graph. This encourages the graph-cut optimization to cut through a small number of links to separate the unknown region into a foreground subregion and a background subregion as shown in figure 5.9(c). We use the refined boundaries for alpha matting.

5.6.2 3D Contour Mesh Construction

To extract a spatio-temporally coherent matte based on the refined boundaries, we first parameterize the boundary and build a consistent 3D contour mesh across the entire video volume.

We begin by separately parameterizing the foreground boundary for each frame. The closed boundary for frame t is parameterized into a set of contour points $C(t) = c_t^1, ...c_t^n$, where the number of vertices n is fixed for all frames. For illustration, we assume here that the foreground object is one connected component without holes.

Next, we build correspondences between contour points across adjacent frames using both shape and local color information. For c_t^i (the ith contour point on frame t) and c_{t+1}^j, we compute a correspondence cost as:

$$g(c_t^i, c_{t+1}^j) = g_s(c_t^i, c_{t+1}^j) + w \cdot g_c(c_t^i, c_{t+1}^j) \tag{5.10}$$

where g_s is the shape distance and g_c is a local "color distance" between the two points. The shape distance g_s is computed as the magnitude of the difference of the shape context vectors [7] of each point. If $S(c_t^i)$ and $S(c_{t+1}^j)$ are the shape context vectors for each point, then $g_s(c_t^i, c_{t+1}^j) = \|S(c_t^i) - S(c_{t+1}^j)\|$.

To increase the accuracy of the correspondence we also include a local color distance term g_c in the correspondence cost. Foreground colors can be expected to be more stable than background colors in close proximity of the contour. For each contour point c_t^i, we select the 16 nearest foreground pixels and compute their mean color $M(c_t^i)$. The color distance $g_c(c_t^i, c_{t+1}^j)$ is computed as

$\|M(c_t^i) - M(c_{t+1}^j)\|$. We normalize g_s and g_c to $[0, 1]$ and set the weight w to be 4 to produce good results.

We use dynamic programming to find the match that minimizes the total correspondence cost; the point-wise correspondence cost summed over all contour points, on all frames. For later use, we also record the frame-to-frame correspondence cost for each frame, $G(t, t + 1)$, taken as the sum over all frames of the matching contour point correspondence costs. We now have a consistent 3D surface mesh over the foreground object.

5.6.3 Trimap Generation

We then generate a trimap for the foreground object for subsequent alpha matting. We once again mark pixels within 10 pixels spatially and within 1 pixels temporally of the boundary between foreground and background. Motion blur is automatically accounted for in this way. When the foreground moves, it leaves behind (and ahead) a trail in the trimap. This implicitly creates wider unknown regions in the trimap in locations where the motion is more intensive.

5.6.4 Spatio-Temporal Border Matting

Applying image matting techniques frame-by-frame will generate noisy results since the small errors generated on each individual frame will be temporally incoherent and are more prone to appear as artifacts to human eyes which are sensitive to temporal aliasing. We thus extend the *border matting* approach introduced in the GrabCut system [110] to generate spatio-temporally smooth mattes.

Figure 5.10 illustrates the intuition behind our spatio-temporal border matting algorithm. See [110] for details of the algorithm within a single frame. In the diagram the contours from the refinement step are shown in black. Border matting defines an α profile along each normal to the contour. This contour has $\alpha = 0$ along the outside edge, and $\alpha = 1$ along the inside edge and varies smoothly in between as defined by an α profile function (see inset of Figure 5.10). The parameters of the α profile are given by an offset $\triangle$ (the distance between the pink circles, $\tilde{c}$, from the contour points, c) and a width σ (shown as the pink error bars). The pink dotted line passing through the $\tilde{c}$'s represents where α takes on value 0.5.

The original border matting optimizes over the $\triangle$'s and σ's (equation (12) in [110]) to create

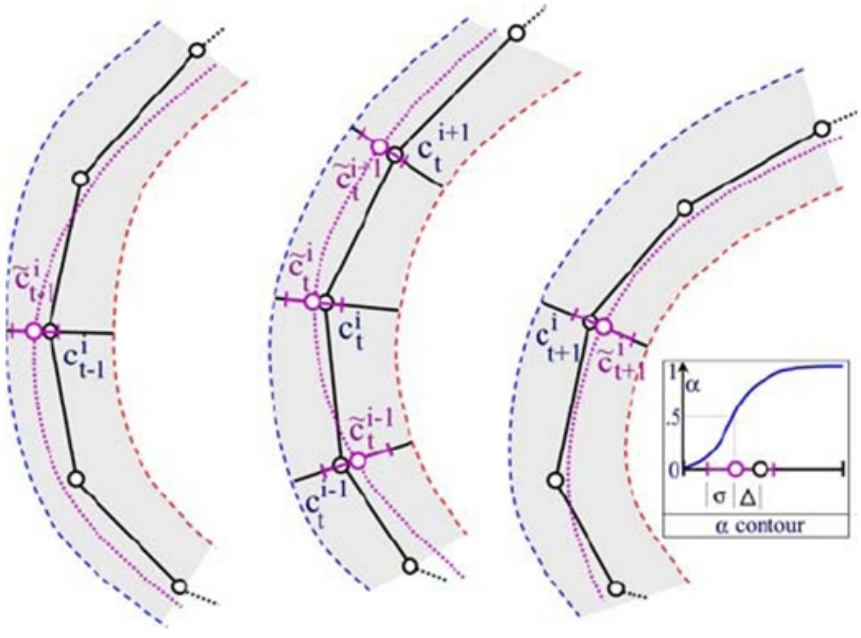


Figure 5.10: Three corresponding portions of trimaps and contours at three successive frames, $t-1$, t, and $t+1$.

an α matte that best explains each pixel's color from nearby foreground and background pixels and the local α value. At the same time, it tries to minimize the variation in the $\triangle$ values, and σ values along the contour to maintain spatial coherence.

To achieve temporal smoothness as well we add a temporal term to equation (12) in [110] and optimize over all contours at all times. Our temporal term is defined as:

$$\sum_t w(t,t+1) \sum_i \| \triangle \kappa \|^2 + \lambda (\triangle \sigma)^2 \tag{5.11}$$

$$\triangle \kappa = (\tilde{c}_t^{i-1} - 2\tilde{c}_t^i + \tilde{c}_t^{i+1}) - (\tilde{c}_{t+1}^{i-1} - 2\tilde{c}_{t+1}^i + \tilde{c}_{t+1}^{i+1})$$

$$\triangle \sigma = \sigma_t^i - \sigma_{t+1}^i$$

where $\triangle\kappa$ is a shape term similar to that defined in the rotoscoping system [4]. It measures the change in the curvature of the contour over time. The $\triangle\sigma$ term is used to encourage σ to change smoothly. λ is a weight which we set to 100.

$w(t,t+1)$ penalizes correspondence errors based on the correspondence cost, $G(t,t+1)$, we

computed earlier (see section 5.6.2). $w(t, t+1)$ is defined as

$$w(t,t+1) = \begin{cases} 0 & : \quad G(t,t+1) \geq T_G \\ 1 - \frac{G(t,t+1)}{T_G} & : \quad G(t,t+1) < T_G \end{cases} \tag{5.12}$$

where T_G is a predefined threshold. If the foreground object changes topology, the correspondence cost will be high and the weight will decrease to zero. In this way we only apply temporal smoothness constraints within blocks of video frames where the foreground motion and appearance is homogenous, and do not apply smoothness constraints across frames where frame correspondences are erroneous or less reliable. The approach allows our system to produce good results even when the video sequence contains intensive motions.

The optimization process extracts a smooth spatio-temporal matte from the input sequence. We estimate foreground colors by using the "foreground pixel stealing" approach described in the GrabCut system to avoid color bleeding artifacts.

5.7 Failure Modes and Solutions

We have exercised our system on many difficult videos. The type of automation we describe is quite powerful, but will fail in some cases. We describe next ways to call on earlier methods to extend the use of the system to almost any video. The beauty of an interactive system such as the one described, is that in the worst case, the user is still in control and given enough effort can direct the system to a satisfactory answer.

Video will not stabilize - Optical Flow

If there is significant camera translation, then parallax will prevent a video from being stabilized. In this case, an optical flow algorithm [10] will run which returns the apparent motion of each pixel from one frame to the next. We then modify the original pixel lattice by connecting each pixel through its flow vector to a new pixel in the next frame, as well as to all its immediate neighbors. Then, the system runs as before except that the local data and link costs are not used. We show one such result later.

Foreground is similar to background - Rotoscoping

The graph-cut based approach is efficient to segment sequences where the differences between foreground colors and background colors can be captured by the color models. However, this will

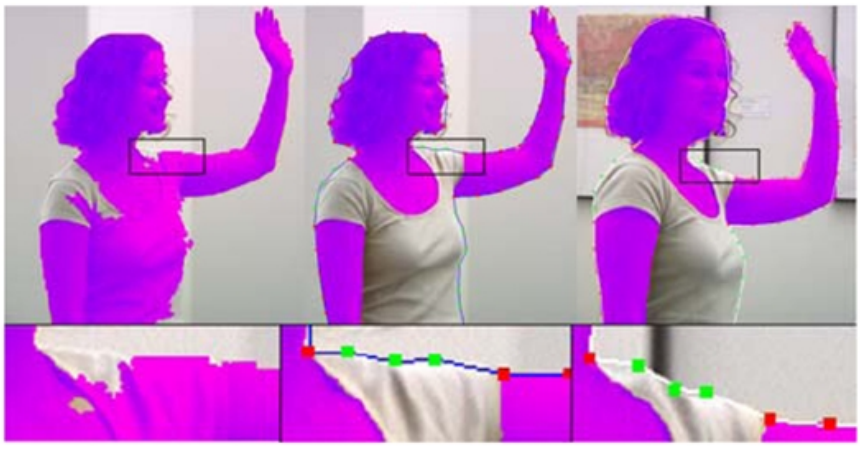


Figure 5.11: Left: initial segmentation result on one frame. Graph-cut segmentation alone cannot robustly segment the girl's shirt due to color ambiguity. Middle: the user segments the skin and hair regions out, and adds a rotoscoping curve around the boundary. Right: the rotoscoping curve has been robustly propagated to the 32nd frame and the green vertices have been optimized.

not be always true. An example is shown in figure 5.11(left), where the girl's white shirt is ambiguous with the white wall in the color space. The pre-segmentation cannot preserve the accurate boundary of her shirt, thus the user needs to draw a number of strokes along the boundary of shirts to break segments into pixels, which makes the system less efficient.

In this case we can call on rotoscoping methods [4] constrained by our partial results. As shown in figure 5.11(middle), we first treat the girl's skin and hair regions as foreground and segment them from the sequence using the graph-cut based system. Since the skin and hair colors are distinct from those of other regions, they can be extracted out very efficiently with only a few strokes from the user.

The user then indicates a rotoscoping curve around the body, including the shirt. Note that there are two kinds of vertices on the rotoscoping curve. The red vertices are attached to the boundaries of the pre-segmented regions. Based on correspondences across frames between the pre-segmented regions, the red vertices are easily propagated across time without resorting to the optimization process in the normal rotoscoping process. The green vertexs are "unknown" ones corresponding to the shirt region, they are the only vertices we need to optimize.

To propagate the rotoscoping curve from frame t to frame $t+1$, we first locate all the red

vertexes by using the temporal correspondence of the pre-segmented regions. The green vertices are initialized based on their relative positions to the red vertexes by only minimizing the shape energy terms described in the rotoscoping process [4]. The green vertices are optimized to follow strong edges while maintaining the constraints imposed by the red vertices at their ends. The green vertexs are automatically initialized very close to their final locations, thus the optimization can be performed more robustly, as shown in figure 5.11(right).

Foreground still to similar to background - Recurse

In the ballet sequence (see Figure 5.12) the shoe color matches the floor so well they were very hard to segment. In this case one can first mark off all but the small portion of the video containing the shoes. Operating on this subset of the video is more successful since the foreground and background color models can much more tightly focus on the shoe and floor colors. Once done, the shoes are accepted as foreground. The rest of the video is brought into consideration and the remainder of the foreground object is segmented.

All else fails - Trust user

The beauty of an interactive system such as the one described, is that in the worst case, the user is still in control. In some cases when the foreground includes very thin structures in which almost every pixel is some combination of the foreground and background colors no system we know of will work robustly. The tail of the elephant presented such a challenge and required a few thin strokes on about every fifth frame to be included in the foreground object.

5.8 Results

We have run Interactive Video Cutout on a number of videos of between 100 and 200 frames in length. Preprocess times to compute the mean shift hierarchy and neighborhood relationships took approximately 30 minutes to an hour. Postprocess time including refinement graph cut and matting took about 30 seconds per frame or about 1 to 2 hours per example.

User time depends on the complexity of the scene; more complexity, thin structures, and fast motion may require more strokes. The user also has to wait for the graph-cut computation between iterations. Due to the efficiencies from the hierarchical mean shift, our graph-cut computation took only 7 to 15 seconds, and thus fit nicely within the interactive framework. As sessions progressed,

Figure 5.12: Left: Original frame from four example video sequences. In order from top to bottom the sequences are named Ballet, Skateboarder, Man in Cap and Stairs. Right: Three frames from each sequence showing just the extracted foreground objects.

the graph-cut time increases slightly due to the increased number of samples used to compute the color models.

The performance of our system on seven different video sequences is shown in Table 5.1.

The video, Amira (1) was reported to have taken about 40 minutes of rotoscoping in [4]. Because of the strong edges and almost constant colors in this sequence, we were able to fully extract the foreground by painting only a few strokes on a single frame of the video.

More challenging examples are shown in Figure 5.12. In the skateboarder example there are many strong edges in the background which move slightly in the video despite its being captured with a tripod. The foreground character moves very rapidly across the screen and back. Nevertheless, a successful matte is extracted with about 20 minutes of user time. The elephant seen in Figure 1 presented special difficulty due to the similarity of the elephant's color with its background, and the presence of the relatively thin trunk and tail structures. The high frequencies in the background coupled with the uneven stabilization of the video also make this a challenging example. Approximately 40 minutes of user time was needed to extract the elephant. The ballet sequence presented additional problems due to fast motion and the fact that the feet almost matched the floor color. We

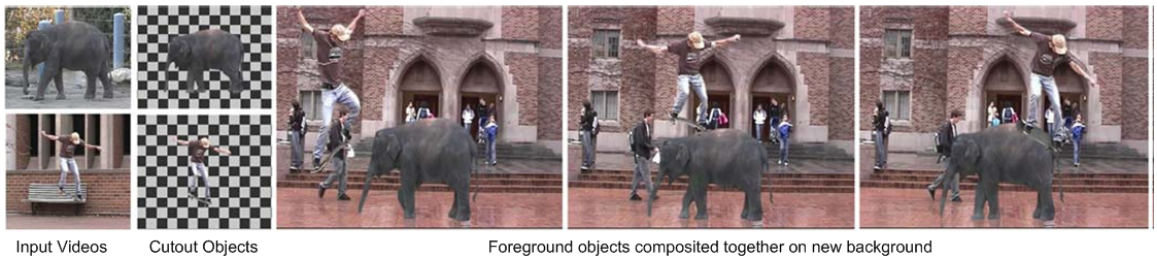


Figure 5.13: A novel video created by our system in which the skateboarder skates on the elephant in front of the student center of the University of Washington.

Table 5.1: Video sequences and timings for each stage of the algorithm.

Sequence	Size	Preproc.	Graph-cut	Artist time	Postprocessing
Skateboarder	$720 * 480 * 175$	30 min	12.50 sec	20 min	40 min
Elephant	$720 * 480 * 100$	20 min	9.10 sec	40 min	30 min
Man in Cap	$640 * 480 * 150$	30 min	16.50 sec	20 min	35 min
Ballet	$640 * 480 * 150$	25 min	11.50 sec	50 min (twice)	30 min
Amira (1)	$640 * 480 * 100$	15 min	7.05 sec	2 min	50 min[a]
Amira (2)	$640 * 480 * 80$	12 min	5.00 sec	15 min	35 min[b]
stairs	$720 * 480 * 100$	20 min	8.50 sec	20 min	30 min

[a] Interactive trimap generation, Bayesian matting

[b] Bayesian matting

completed this sequence in two passes, extracting the feet first and then the rest of the dancer while constraining the feet to be foreground. Each pass took about 50 minutes.

Once an alpha matted foreground object is extracted from a video, there are a number of possible applications. The most straightforward application is to composite the extracted foreground onto a new background video. Figure 5.13 shows an example of compositing a skateboarder and an elephant onto a background video taken outside the library at the University of Washington.

5.9 Discussion

We have demonstrated an interactive system for quickly extracting alpha matted foreground objects from videos. We have also touched on a couple of applications for the extracted objects.

Although our system works well, as we mention, previous work also is helpful in special circumstances. No single solution is likely to work on all inputs. Thus an integrated set of tools will someday need to be developed to fully exploit all the ideas that have been presented.

There are other problems as well we have not tackled. One may want to extract an object that passes behind thin structures. In this case, one would need to extract these blocking objects and fill in the occluded pixels with, for example, texture synthesis.

Despite the inherent difficulties of working with video, we believe our interactive video cutout system demonstrates an example of how a well designed user interface and efficient algorithms opens new possibilities for video editing.

Chapter 6

APPLICATION: VIDEO TOONING

Once an alpha matted foreground object is extracted from a video, there are a number of possible applications that can be applied on it. The most straightforward application is to compose the extracted foreground to a new video sequence as we show in the previous chapter. In this chapter, we will demonstrate a system that can abstract the extracted video objects into a variety of cartoon styles [1].

6.1 Introduction

Animated imagery brings life to the screen. The stylized abstraction of reality one sees in animation adds an immediate impact that cannot be captured by simply pointing a video camera at a scene. But such animation is both labor intensive and requires considerable artistic skill. We show that once the video objects have been extracted by using the cutout system described in the previous chapter, a set of new rendering techniques can be used to lower these barriers by allowing video objects to be transformed into a cartoon-like style.

Stylized rendering of video, which we dub *Video Tooning*, is an active area of research in non-photorealistic rendering (NPR). Our work was motivated in part by the still image stylization and abstraction approach presented by DeCarlo and Santella [37]. Unfortunately, a direct frame-by-frame application of this approach to stylize video results in a very incoherent temporal result. Our goal then has been to produce temporally coherent stylization while also allowing a user significant freedom in choosing the final look of the video.

Generally, there are three major criteria that a successful video tooning system should meet:

1. The result sequence should maintain spatio-temporal consistence to avoid significant jumps in frame transitions;

[1] This system has been published in [138]

2. The content of the video should be abstracted in such a way as to respect the higher level semantic representation.

3. The artist should be able to express control over the style of the result.

When NPR methods designed for static images are applied to video sequences on a frame-by-frame basis, the results generally contain undesirable temporal aliasing artifacts. We overcome the coherence problem by directly using the extracted spatio-temporal video objects, which create a 3D data volume and directly cluster the pixels in the three dimensional space (x,y,t). This avoids many of the robustness problems of optical flow methods that track pixel or object movements only between successive frames.

Cartoon animations are typically composed of large regions which are semantically meaningful and highly abstracted by artists. A region may simply be constantly colored as in most cel animation systems, or it may be rendered in some other consistent style. To achieve similar results, we further allow the user to decompose the extracted video objects into semantic regions by iteratively using the video cutout system.

Semantic 3D regions are further processed into polyhedral surface representations and smoothed. From these, we create a set of *edge sheets* from portions of surfaces. The user is able to add strokes within regions on keyframes. These are also propagated through time guided by the semantic regions. These result in smooth "*stroke sheets*" within the solid regions.

At this point, the video is represented as a set of 3D polyhedral semantic regions, 2D edge sheets along the surface of the regions and stroke sheets within the regions. These primitives, when sliced at a frame time yield solid areas and curves along their edges and within the interior. We present a variety of rendering options for these primitives to construct a frame of the stylized video.

- Iteratively use the video cutout system to decompose video objects into large semantic regions;

- Reconstruct and smooth the surfaces of semantic regions and determine corresponding edge and stroke sheets;

- At each frame time slice regions and sheets to yield area and curve primitives;

- Render these primitives in the desired style to create the final stylized video frame and output.

6.2 Related Work

We are not the first to present methods to stylize video. Some approaches apply NPR rendering methods frame-by-frame. Recent films such as *A Waking Life* and *Avenue Amy* were painstakingly modified one frame at a time. Although the stylized look of this work has some appeal, the jitter produced by the frame-by-frame method may or may not have been the goal of the artists. In any case, the tedious workload made these productions very expensive to complete.

In 1997, Litwinowicz [87] proposed an automatic approach to produce painterly animations from video clips. Optical flow fields were used to push brush strokes from frame to frame in the direction of pixel movements. Hertzmann et al. [59] modified each successive frame of the video by first warping the previous frame to account for optical flow changes and then painting over areas of the new frame that differ significantly from its predecessor. This was extended in [58] by guiding paint strokes with a general energy term consisting of both pixel color differences and optical flow. Since the energy function defined in this approach is difficult to optimize, they use a relaxation algorithm combined with search heuristics. This formulation allows the user to specify painting style by varying the relative weights of energy terms. The basic energy function yields an economical style that conveys an image with few strokes. The system also allows as fine user control as desired: the user may interactively change the painting style, specify variations of style over an image, and/or add specific strokes to the painting.

One can also view part of our work as extending the in-betweening problem in keyframe animation. A good recent addition to this problem for NPR is the work by Kort [77]. The SnakeToonz system [1] is an interactive system that allows children and others untrained in cel animation to create two-dimensional cartoons from video streams and images. After recording video material the user sketches contours directly onto the first frame of video. These sketches initialize a set of spline-based active contours which are relaxed to best fit the image and other aesthetic constraints. Small gaps are closed, and the user can choose colors for the cartoon. The system then uses motion estimation techniques to track these contours through the image sequence. The user remains in the process to edit the cartoon as it progresses. Although the goals of this work are similar to ours, our

methods work in more unconstrained environments and provide more stylistic choices to the user through the use of edge and stroke sheets.

Some recent video processing techniques treat video as a space-time volume of image data. In the Stylized Video Cubes approach [75], a set of "rendering solids" were created in the volume as a function defined over an interval of time; when evaluated at a particular time within that interval, each rendering solid provided parameters necessary for rendering an NPR primitive. Although this system resulted in interesting abstractions of the underlying video, it had no facilities to respect larger semantic regions as they evolve through time.

DeCarlo and Santella's paper on *Stylization and Abstraction of Photographs* [37] employs a mean shift segmentation to cluster pixels. In their system, images were transformed into a style using a combination of line-drawing and filling large regions with constant color. For abstraction, the system used eye-tracking data to determine where to remove extraneous details and to highlighting important objects. We achieve a similar goal on video sequence.

6.3 3D Surface Construction : Why?

Given the pixelized representation of the semantic regions produced by the interactive cutout system presented in the previous section, we then convert them to 3D polyhedral surfaces as the first step of the Tooning process. The new representation serves two purposes: we can smooth the reconstructed surfaces further using traditional object smoothing operations. In addition, surface reconstruction makes the computation of edge sheets possible, which are used to render temporally coherent strokes in the stylized results. We also show the use of stroke sheets within regions to allow modification of the region interiors.

A side benefit of the continuous representation, which we do not explore here, is that the resulting shapes are now resolution independent in both space and time. Thus, final rendering can be performed at any spatial or temporal resolution and compression/transmission methods no longer need to deal with discrete frame times.

In addition to the surface geometry, each semantic region is annotated with a color, and an edge importance, I_s. The latter value is set between, 1 - always draw an edge around this region, and 0 - do not contribute to the likelihood of drawing an edge. The edge importance of the background is

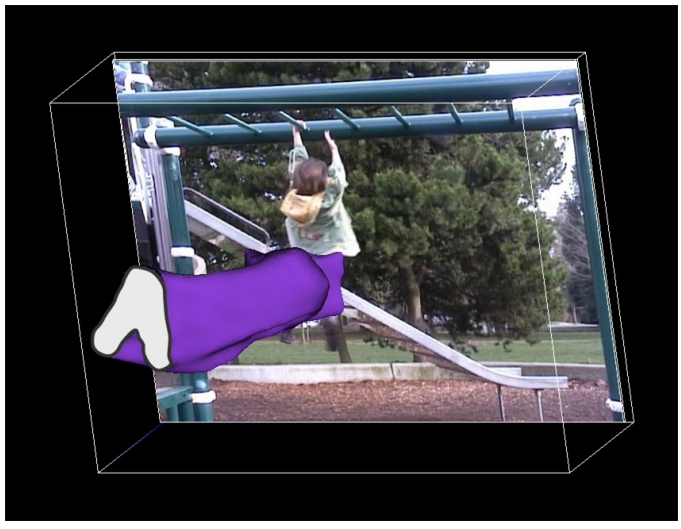


Figure 6.1: A smoothed semantic region sliced at time t.

set to 0.

6.4 Semantic Region Surface Construction

We use the marching cubes algorithm [91] to convert the pixelized data into surface data resulting in polygonal surfaces separating the semantic regions. Our goal is then to smooth the region volumes without introducing any gaps in the video volume. Gaps are avoided by having regions share the set of vertices forming their separating walls. One smoothing step moves each vertex to a weighted average of itself and 0.25 times the mean position of its connected neighbors. The smoothed regions can easily be rendered as solid colored polygons at any time t by intersecting them with a plane perpendicular to the time axis (Figure 6.1).

6.5 Edge Sheets

We also want to add solid strokes to the final rendering much like inked lines in a drawing. Selecting lines and their location on a frame-by-frame basis causes a lack of temporal coherence. To avoid this, we construct a set of smooth two dimensional sheets, or *edge sheets* embedded in the 3D video volume. We slice these sheets at each frame time to extract a curved line for rendering.

6.5.1 Edge Sheet Construction

Edge sheets are derived from the surface representations of the 3D semantic regions. Each pair of adjacent regions will share one or more sections of their surfaces. Sets of contiguous shared triangles between each pair of regions are copied into their own vertex/edge data structure. This forms two dimensional edge sheets embedded in the 3D video volume. Small sheets containing less than a user set number of triangles are discarded.

The remaining polygonal sheets are smoothed in two ways: 1) The boundaries are low pass filtered to avoid jagged edges that could cause temporal artifacts, and 2) internal vertex positions are averaged with their adjacent vertices to provide geometric smoothness.

An edge importance value denoted as I_e for each sheet is set to either the max or the difference of the two region importance values, I_r, of the semantic regions it separates. I_e is compared to a user set threshold between 0 and 1 to decide if the sheet should be used to generate edge strokes at rendering time.

6.5.2 Rendering Edge Sheets

Edge sheets are two dimensional sheets embedded in the 3D video volume. When sliced by a plane at a particular frame time, the edge sheets produce smooth curves that approximately follow the surfaces of the regions. The smoothing step may pull some edges slightly away from the exact boundary between colored regions but this provides a good balance between stroke smoothness and region shape. Figures 6.2 show examples of the inclusion of edge sheets.

Rendering a sheet at some time t involves first intersecting the sheet with a plane at time t to produce a curve. The curve can then be drawn with a number of styles. In [37], the overall thickness was set simply by the length of the stroke in the 2D frame and had a profile that tapered it at its ends. We begin in a similar way by defining a basic style for the line that defines its profile along its length in the spatial domain parameterized by arc length. Many drawing and sketching systems provide such a choice, such as [64].

In our case, we have the added information provided by the edge sheet as a whole that we can leverage to modify the stroke color and thickness or other properties. In addition to an edge's length in a particular frame, the edge also has a limited duration in time. We will use the current time t

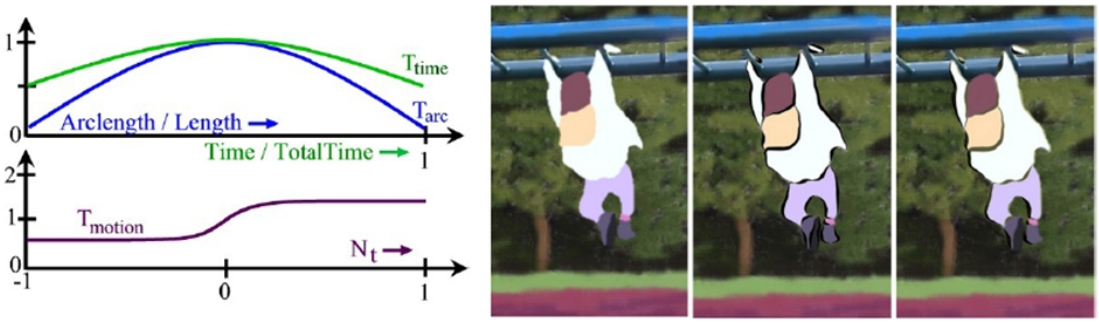


Figure 6.2: Left: Variables to determine edge thickness. Right: rendering results with solid regions with no edges; edges modulated by thickness; edges modulated by an implied lighting direction and thickness modified by motion.

relative to the beginning, t_s, and end, t_e of an edge sheet's existence to modify the edge thickness. More specifically, as $(t - t_s)/(t_e - t_s)$ varies from 0 to 1, the thickness will grow to a maximum at 0.5 while being thinner at the beginning and end of its lifetime.

At each vertex, the sheet also has an implied normal, $N = (N_x, N_y, N_t)$, taken as the average of the normals of adjacent triangles. The direction of the normal is taken to point outwards from the region with the higher I_r.

N_t indicates the rate of an edges's motion across the frame. In particular, a positive value of N_t indicates a *trailing edge* that we will thicken during rendering, and a negative N_t indicates a *leading edge* that is thinned.

The sum effect on thickness of a point on an edge based on its position along its arclength, the frame time relative to its lifetime, and the edges motion is summarized as

$$Thickness = T_{base} * T_{arc} * T_{time} * T_{motion} \tag{6.1}$$

T_{base} is set by the user and represents the thickness of the center of a still edge at the middle of its existence in time. The other terms vary as shown in Figure 6.2. The formulas of these terms are given below. As the graphs imply strokes thin at their ends both in space and time.

$$T_{arc} = 1 - 3.2 \cdot (s - 0.5)^2, \quad s \in [0, 1] \tag{6.2}$$

$$T_{time} = 1 - 2 \cdot (s - 0.5)^2, \quad s \in [0, 1] \tag{6.3}$$

$$T_{motion} = \begin{cases} -0.5 & : \quad -1 \leq N_t < -0.2 \\ 0.5 + sin(N_t \cdot \frac{5\pi}{2}) & : \quad -0.2 \leq N_t < 0.2 \\ 1.5 & : \quad 0.2 \leq N_t \leq 1 \end{cases} \tag{6.4}$$

Finally, the spatial components of the normal (N_x, N_y) can be used to shade the edge based on a dot product with a virtual *light direction* given by the user. We'll denote this dot product D. For example, we may wish to make edges facing the upper right brighter and those facing down and to the left darker. An example can be seen in Figure 6.2 (right).

For those who prefer to visualize the edge sheet as whole, one can imagine a curved sheet that is thin along all its edges. It is thickest in the center both along its spatial and temporal extent. It also tends to be thicker in portions that face along the time axis as opposed to facing backwards in time. Finally, the whole sheet is lit from an infinite point source in some $(x, y, 0)$ direction.

6.6 Filling the Region Interiors

In addition to drawing edges, we also fill the interiors of slices of the regions. There are three ways the interiors can get filled; by direct pixel coloring, dividing the regions into subregions and then coloring, or by filling the regions with paint-like strokes. In fact, all three can be combined through standard compositing if desired.

6.6.1 Stroke Sheets

Our system also allows the user to lay down paint strokes within regions at keyframes (see Figure 6.3) and have them automatically interpolated to create temporally coherent motion of the strokes. In much the same way that the edge sheets are created, we create 2D *stroke sheets* that are embedded fully within a semantic region. A user draws strokes within semantic regions on a keyframe, defining the stroke skeleton, color, and style [64]. On subsequent keyframes intersecting the same semantic region, the user must draw the new skeletons of each stroke.

Between keyframes, k_1 and k_2, strokes are *flowed* forward in time from k_1 and backward in time from k_2. Figure 6.3 illustrates how to flow the corresponding positions of a stroke in frames following k_1. We only consider one point C_p on the stroke (each stroke is sampled into 15 points

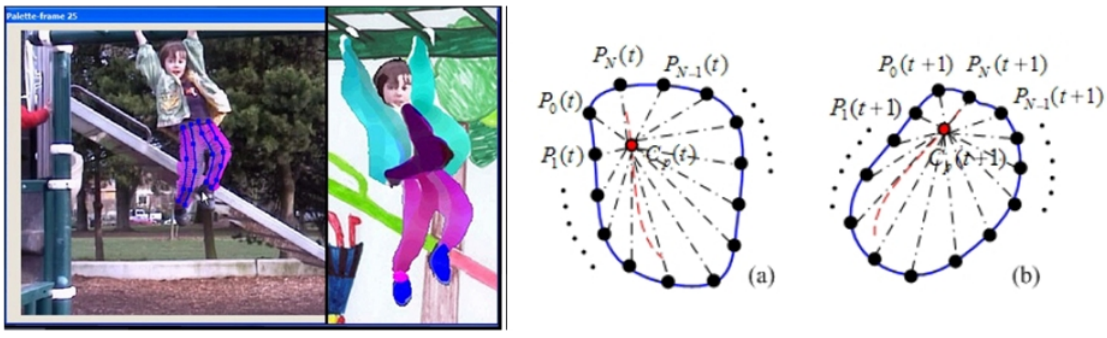


Figure 6.3: Left: User adding a paint stroke at a keyframe and the resulting strokes in a frame of final animation. Right: Illustration of ”pushing” a within-region stroke from one frame to the next, as shown in (a) and (b).

along its length in our system). From the previously described semantic region interpolation, we have N sample points along the boundary of the semantic region in which the stroke lies, denoted as $P_i(t), i = 1, ..., N$. By computing the distances d_i between C_p and the $P_i(t)$, we get a vector of weights $< w_i = (1 - d_i/d_{max})^2 + 0.01 >$, which describes the relative position of C_p within the region.[2]

On the next frame, we examine the corresponding points along the boundary, $P_i(t + 1), i = 1, ..., N$. We then compute the new location of each point $P_i(t + 1)$ as a weighted average of these points using $< w_i >$ as the weights. Each control point along the stroke is processed in the same way.

In the same way the strokes are flowed backwards from k_2. The final position is a linearly weighted average of the forward and backward flow, with the first weight dropping from 1 to 0 as time goes from k_1 to k_2 and from 0 to 1 for the reverse.

The interpolation of the strokes creates a two dimensional stroke sheet lying within the semantic region (although the final rendering of the strokes may overlap region boundaries). These sheets are sliced at a time t to provide a skeleton for a stroke to be rendered.

[2]Although this formula does not have all the niceties of barycentric coordinates it has performed well. We do not know of a general barycentric-like formula for irregular non-convex polygons. A good choice for convex polygons is [98].

Figure 6.4: Construction of the final frame. From left to right: original frame, regions as solid shapes, head rendered with segmentation result, edges on all region boundaries, edges only at important boundaries based on edge importance for each region, final frame that now includes subregions within the shirt to show shadowing.

6.6.2 Subregions

In some cases we have found it useful to allow the user to define their own subregions. The same interface used to define the semantic regions is used to allow the user to draw subregions on keyframes, with the only exception being that the result is constrained to lie fully within a specified semantic region. This allows a second color region within a larger region in the final rendering. This was used for the shadowing within the shirt seen in Figure 6.4.

6.6.3 The Background

The background is defined as a single semantic region including all portions of the video lying outside the user defined semantic regions. The background can be filled just like any other semantic region.

In our examples, the camera was still and the foreground objects (people) moved across the field of view. In these cases it is quite easy to extract a constant background frame with a median filter of each pixel through time. This can then be rendered as is or modified with a paint program or simply replaced. The foreground regions are then rendered over this background. A number of different background styles can be seen in our examples.

6.7 Discussion

We have presented a system for interactively transforming video to a cartoon-like style. Our system solves the main challenge of providing temporal stability by leveraging the spatio-temporal video objects generated by the video cutout system. We have shown how the cutout results together with the artist's input can provide a variety of non-photorealistic styles.

As future work, there are many more stylistic choices to explore for rendering the solid regions, edges, and paint strokes. We also hope to explore the use of layers as were exploited in the movie *Waking Life* particularly if the source video is derived from hand held cameras and thus subtle parallax issues are present. There are also many enhancements to the user interface that should be added.

One interesting future direction to explore is vectorized encoding of the result. The monkey bars video contains about 120,000 3D vertices vs. approximately 120 million pixels in the original video, a ratio of 1:1000. We also expect to be able to leverage mesh simplification methods to further lower this ratio. There is clearly a lot of coherence to leverage for compression, and there is also the added benefit of resolution independent encoding.

Chapter 7

APPLICATION: MOTION MODULATION

The video tooning system we presented in the prevoius section can stylize the appearance of video objects. In this chapter, we demonstrate a simple yet general *cartoon animation filter* that can modulate and bring life to the motion of the extracted video objects as well. We will first introduce the filter in detail, then describe how to apply it to the specific target of video objects[1].

7.1 Introduction

Cartoon animation is the art of manipulating motion to emphasize the primary actions while minimizing irrelevant movements. Expert animators carefully guide the viewers' perceptions of the motion in a scene and literally bring it to life. Some of the principles of cartoon animation are well known. As outlined in Thomas and Johnston's "Disney Animation - The Illusion of Life" [70] and repeated by Lasseter [79], good animators add features not seen in the real world, including anticipation and follow-through to the motion with related squash and stretch to the geometry. Yet, most of us do not possess the time and skills necessary to keyframe such effective animated motion by hand. Instead we rely on recording devices such as video cameras or motion capture (MoCap) systems to faithfully record *realistic* motion. However such realistic motion often appears disappointingly wooden and dead in comparison to good cartoon animation.

To modulate the realistic motion presented in the extracted video objects, we present a simple yet general cartoon animation filter that can add both anticipation/follow-through, and squash and stretch, to it. Mathematically, the filter can be described as:

$$x^*(t) = x(t) - \overline{x''}(t) \tag{7.1}$$

where $x(t)$ is the motion signal to be filtered, and $\overline{x''}(t)$ is a smoothed (and possibly time shifted) version of the second derivative with respect to time of $X(t)$. An equivalent way to describe the

[1]This work has been published in [137]

filter is as a convolution of the motion signal with an inverted Laplacian of a Gaussian (LoG) filter, sometimes known as an *unsharp* filter[2].

$$x^*(t) = x(t) + x(t) \otimes -LoG \tag{7.2}$$

The linear nature of the filter provides simplicity and speed, and should not be expected to do all one may desire from a general motion editor. Instead, it provides a tool that can be coded in an afternoon and then applied to a broad class of existing motion signals.

Although the idea of enhancing animation by linearly filtering the motion signals [129, 15] is not in itself new, our work extends these previous techniques and makes the following novel contributions:

Unified Approach: We use the same filter to produce anticipation, follow-through and squash and stretch. Anticipation and follow-through are a natural byproduct of the negative lobes in the LoG filter. Applying differing time shifts to the filter at the boundaries of objects generates squash and stretch.

Simple User Interface: We automatically generate good default values for almost all of the parameters of our filter, leaving the user to specify only the overall strength of the exaggeration. Although the default parameters cannot always produce the kind of motion a skilled animator would create, our approach allows novice users to easily enhance and enliven existing motions.

7.1.1 Related Work

While the importance of cartoon style exaggeration is well recognized, none of the previous techniques combine a single unified approach with a simple one parameter user interface. For example, simulation, both physically-based [41] and stylized [19, 65], and surface deformation [141] are common techniques for generating squash and stretch. These do not handle anticipation and follow-through, force users to set many parameters and are complicated to implement. Moreover these techniques require an underlying 2D or 3D model and therefore cannot be directly applied to some types of motion signals including video.

[2]The unsharp filter is often defined as the difference of Gaussians, or DoG filter, which is similar is shape and effect to the LoG.

Motion signal processing is another approach for exaggerating motion, that applies signal processing techniques to motion curves. Our methods lie in this class of techniques. Unuma et al. [129] introduced the approach in 1995 and used it to interpolate and extrapolate between Mocap walking sequences (e.g. brisk, tired, fast, ...) in the frequency domain. As a result the control over motion exaggertation is very indirect. Bruderlin and Williams [15] also process MoCap data in the frequency domain. Adopting the user interface of a graphic equalizer they provide slider controls over gains of frequency bands for joint angles. While they demonstrate that carefully adjusting particular frequency bands can generate anticipation and follow-through, the controls are extremely low level and un-intuitive. Users must mentally map controls over frequency bands into effects in the time domain. They also limit their techniques to MoCap data only, do not address squash and stretch effects.

Collomosse's VideoPaintbox [27] includes a combination of techniques for genereating cartoon style animation from video. After segmenting the video into individual objects, a complex case-based algorithm with several (4 to 6) user set parameters is used to modify the motion curves and generate anticipation and follow-through. A completely separate deformation-based approach is used to squash and stretch the objects. While the VideoPaintbox can generate nicely stylized motions, the complexity of both the implementation and interface make it less suitable for quickly enhancing motion signals.

7.2 The Cartoon Animation Filter

7.2.1 The Filter

The animation filter as defined in the introduction subtracts a smoothed (and potentially time shifted) version of the motion signal's second derivative back into the original motion. More specifically, the second derivative of the motion can be convolved with a Gaussian and then subtracted from the original motion signal.

$$\overline{x''}(t) = x''(t) \otimes Ae^{(-(((t/\sigma)\pm\Delta t)^2))} \tag{7.3}$$

where $x''(t)$ is the second derivative at time of $x(t)$, the amplitude, A, controls the strength of the filter. A second parameter, σ controls the Gaussian variance, or width, of the smoothing kernel. As we will show, the time shift, Δt, can be used to create stretch and squash effects by applying the

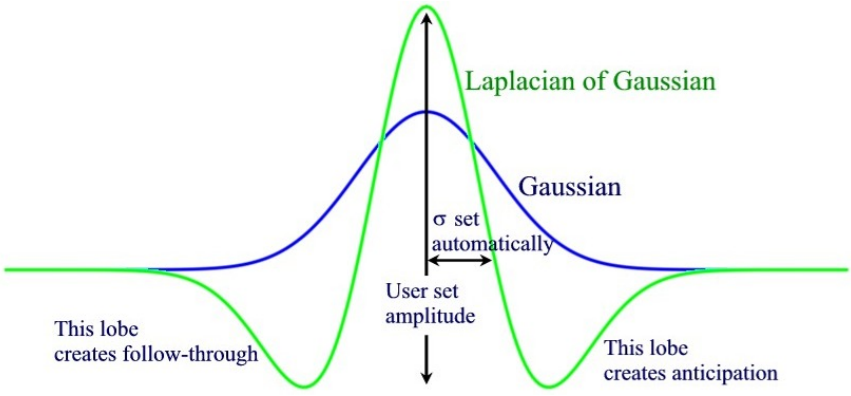


Figure 7.1: The profile of the anticipation filter.

filter shifted forward in time for the leading edge of the acceleration and shifted backward in time for the trailing edge.

More compactly, since convolving the second derivative with a Gaussian is equivalent to convolving the original signal with the second derivative (Laplacian) of the Gaussian, LoG, we implement the filter as such. The inverted LoG filter, is similar to the *unsharp* filter in the image domain. As with any sharpening filter, it produces a *ringing* which is often considered an undesirable artifact in images. In our case, the same ringing produces a desired anticipation and follow-through effect.

Figure 7.2 shows an example of applying the filter to a simple translational motion. In this example a ball stands still, then moves with constant speed to the right, then stops. By applying the cartoon animation filter with no time shift to the centroid of the ball, we add anticipation and follow-through to the motion (i.e., it will move slightly to the left of the starting location before going right, and will overshoot the stop point). These effects are due to the negative lobes on the inverted LoG filter that precede and follow the main positive lobe in the center of the filter (see Figure 7.1).

In principle an expert user could manually set any of the parameters (A, σ, Δt) of the filter. We have developed automated algorithms for setting σ and Δt so that novice users can simply specify the strength of the filter A.

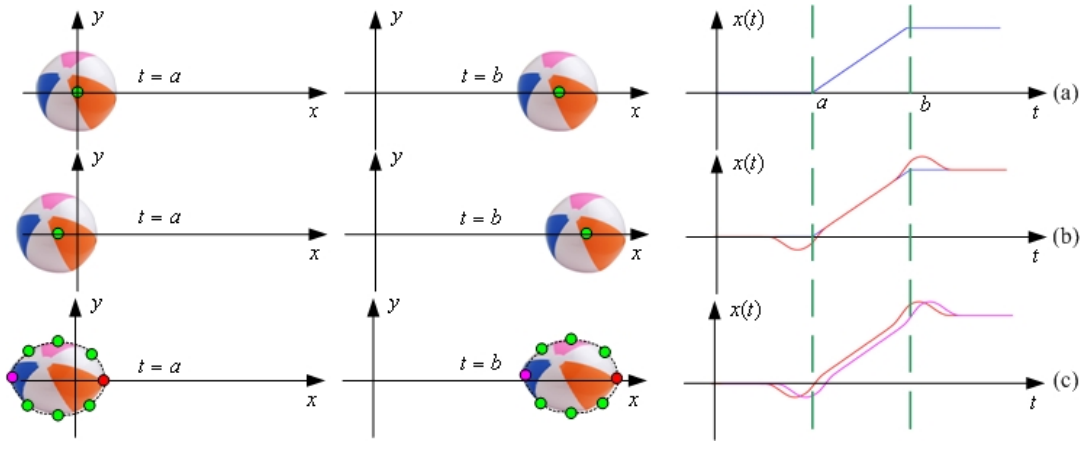


Figure 7.2: (a). A simple 1D translation motion on a ball. (b). Filtered motion on the centroid of the ball, the filter adds anticipation and follow-through effects to the motion. (c). By applying the filter on the outline vertices, the ball exhibits the deformation effect.

7.2.2 Choosing the Filter Width

The width of the filter is defined by the standard deviation, σ, of the Gaussian. Intuitively, we would like the dominant visual frequency, ω^*, of the motion to guide the frequency response of the filter. As the frequency changes, we would like the filter width to change as well. To do this we use a moving window over 32 frames centered at t and take the Fourier transform of the motion in the window. We then multiply the frequency spectrum by the frequency and select the maximum as the dominant visual frequency, ω^*.

$$\omega^*(t) = \max_\omega \ |X(\omega)|\omega \tag{7.4}$$

Or equivalently, we can take the maximum of the Fourier transform of the velocity $x'(t)$.

$$\omega^*(t) = \max_\omega \ |\mathcal{F}(x'(t))| \tag{7.5}$$

Equation 7.5 expresses the fact that we are concerned with the dominant frequency in the velocity changes. The width of the LoG filter thus varies over time and is defined by $\sigma(t) = 2\pi/\omega^*(t)$.

Figure 7.3 illustrates how the dynamically modified σ is able to exaggerate all parts of the motion in a uniform way. The blue curve shows the Z-component of hip motion of a "golfswing" MoCap sequence. As we can see the dominant frequency of the motion dynamically changes overtime. A fixed width LoG filter (the green curve) exaggerates the latter part of the motion but fails to

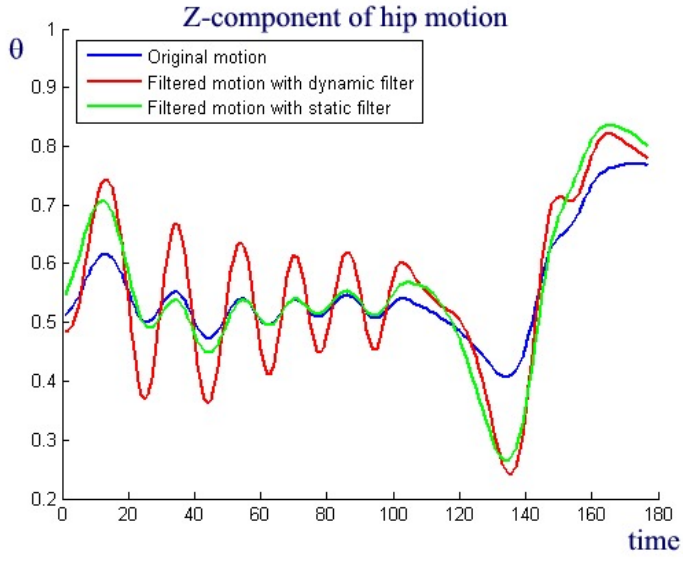


Figure 7.3: By dynamically adapting σ the animation filter is able to exaggerate all parts of the motion.

consistently modify the earlier portion. By dynamically setting σ our filter exaggerates the motion throughout the animation.

7.2.3 Squash and Stretch

Deformation is another important effect in animation to emphasize or exaggerate the motion. Squash and stretch is achieved by slightly time shifting the same LoG differentially for the boundary points of the object.

We use the ball shown in Figure 7.2 to illustrate the idea, and the same approach can be applied to more complicated shapes. Instead of representing the object using its centroid, we represent it by a set of vertices along its outline, as shown in Figure 7.2c. For a vertex p, we calculate the dot product of a vector from the centroid to the vertex, $\vec{B_p}$, normalized by the maximum length vector, and the acceleration direction as $s_p = \tilde{\vec{B}_p} \cdot |\vec{x''}|$, and time shift the LoG filter based on s_p as

$$LoG_p(t) = LoG(t - \Delta t) \tag{7.6}$$

where

$$\Delta t = s_p \cdot \sigma(t) \tag{7.7}$$

$$\tilde{\vec{B}_p} = \vec{B_p}/max_p B \tag{7.8}$$

When $s_p > 0$ (the red vertex in 7.2c), the vertex is on the leading edge of the acceleration will anticipate the upcoming acceleration. On the contrary, if $s_p < 0$ (the purple vertex in 7.2c), the vertex is on the trailing edge of the acceleration, and thus it will be affected later. Since we add time shifts differentially to each vertex, the object will deform, as shown in Figure 7.2c.

7.2.4 Area Preservation

At each point in time, the difference in the deformation between the leading and trailing vertices will create a stretch or squash effect along the line of acceleration. To approximately preserve area we scale the object in the direction perpendicular to the acceleration inversely to the squash or stretch.

7.3 Filtering Video Objects

We first extract video objects using the interactive video cutout system. Once we segment out the object region on each frame t, we parameterize the outline into a polygon S_t and use the set of polygons as the representation of the motion, as shown in Figure 7.4.

We then apply our filter to the polygons. The ideal way to do this is to build correspondence between polygons in adjacent frames, and for each vertex calculate and filter its motion trajectory across time. However, this approach is impractical since calculating correspondence across frames is erroneous, especially when the video object changes its topology. Applying the filter on inaccurate registration results will cause unacceptable distortion of the object.

In our work we apply the filter to the video object in a more robust and reliable way. On each frame we compute the centroid of the outline polygon C_t as the mass of the object, and apply the filter to the single motion trajectory $C_t, t = 1, ..., T$ to get the deformed trajectory $\hat{C}_t$. We calculate a mass shift Δ_t at each frame as $\hat{C}_t - C_t$. We then deform the outline S_t based on the mass shift.

Maintaining constraints. This simple application of the animation equation will often result in deviations from sematic constraints. For example, the skateboarder may no longer be standing on the

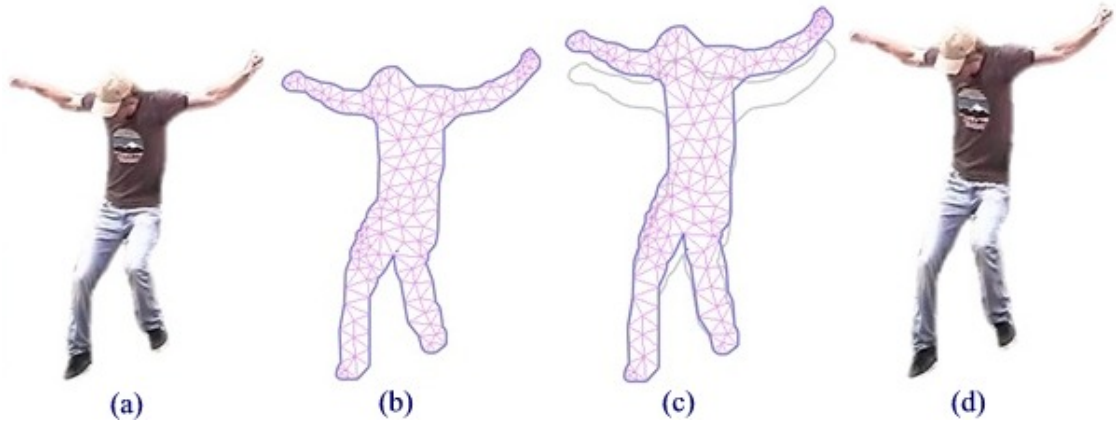


Figure 7.4: Illustration of deforming video objects. (a) Extracted object. (b) Parameterized object mesh. (c) Deformed object mesh. (d) Deformed object by texture mapping.

skateboard. To maintain constraints, we specify that the vertices on the bottom of the feet must retain their original paths. For each constrained vertex, the difference between the constrained position and the position after filtering is spread to the other vertices with weights inversely proportional to the distance from the constrained vertex.

Texturing deformed objects. To texture a deformed video object, we first triangulate the original outline to create a 2D object mesh [116]. For each vertex inside the object, we compute a mean value coordinate [45] based on its relative location to the vertices on the outline. Once the outline is deformed, we use the calculated mean value coordinates to re-locate each vertex inside the object, resulting in a deformed object mesh (Figure 7.4d). We then perform linear texture mapping to create a deformed object based on the deformed mesh.

The filter stretches the skateboarder when he jumps onto the chair, and squashes him when he lands on the ground. These squash and stretch effects significantly exaggerate the motion and make it more alive. Figure 7.5 shows two other examples.

7.4 Discussion

We have demonstrated a simple, one parameter filter that can simultaneously add exaggeration, anticipation, follow-through, and squash and stretch to video objects. We have tried to maintain a

Figure 7.5: Applying the filter to the monkeybar (left) and stairs (right) sequence. Top: original frames. Bottom: corresponding frames with filtered motion.

balance between simplicity and control that favored simplicity. Thus, the application of the cartoon animation filter probably is not satisfactory for hand crafted off-line animation systems although it may be useful for previews.

How to efficiently add constraints to the video objects remains to be a difficult problem. In our examples the video sequences are captured by static cameras thus the backgrounds are still, which makes the job of manually specifying constraints easier. For sequences involving dynamic backgrounds, an intelligent user interface needs to be designed for efficiently specifying constraints.

Chapter 8

CONCLUSION AND FUTURE WORK

In this thesis, we have presented a set of novel algorithms to solve the basic matting equation (1.1), both on images and video. Combining these algorithms with novel user interfaces, we have developed interactive systems for efficiently extracting foreground objects from images and video, which have been demonstrated to be much more practical and efficient compared with previous approaches. We have also demonstrated techniques to apply a variety of applications on the extracted video objects, mainly for rendering purposes. In this chapter, we review our technical contributions and suggest areas for future work.

8.1 Conclusion

8.1.1 Foreground Extraction from Images

Image matting has been known to be an ill-posed problem. In Chapter 2, we first analyze the weakness and failure modes of previous approaches, especially the *"sampling trap"*, i.e. using naive color sampling methods to estimate foreground and background colors for unknown pixels, or using propagation-based methods to avoid color sampling under weak assumptions on image statistics. We argue both methods are not enough to generate good results for complex images. We propose a new *Robust Matting* algorithm to try to explicitly avoid this problem. In our approach we also sample foreground and background colors for unknown pixels, but more importantly, analyze the validity of these samples. Only good samples are chosen to contribute to the data term of the matting energy function. The energy function we define also contains a neighborhood term to ensure the smoothness of the matte. To justify this approach, we present an extensive and quantitative comparison between this algorithm and a number of previous approaches, leading to a benchmark for future matting research.

Although the Robust Matting algorithm generates higher quality results than previous matting approaches, it still works in an offline fashion. In the offline setting the user is required to first

specify a trimap, then invoke the algorithm to compute a matte. If the initial matte is not satisfactory, the user then needs to refine the trimap and run the matting algorithm again. This process is very inefficient for the user since the wait time in each iteration is usually long. To solve this problem we develop a *Soft Scissors* system in Chapter 3, the first interactive tool for extracting high quality matte in an online setting. This system is built upon the Robust Matting algorithm, thus has the capability of extracting accurate mattes for complex images. More importantly, the system is able to incrementally update the matte given newly added user input so that the matte can be estimated instantly when the user is roughly specifying the foreground edge. The foreground colors of real mixed pixels are also computed in realtime so the final composite can be interactively revealed along with the matte. We also design an intelligent user interface where the width and boundary conditions of the scissoring stroke can be automatically adjusted to fit the fuzzyness of the foreground boundary ahead of the current mouse position. We show that the Soft Scissors noticeably outperforms previous matting approaches both in quality and in efficiency, which makes it a practical tool for image matting and compositing.

The Robust Matting algorithm and the Soft Scissors system, along with previous matting approaches, try to estimate the alpha matte from a single input image. Once the matte for the foreground object is extracted, it then be used to re-composite the foreground object onto a new background. In this process matting and compositing are treated as independent processes. We propose a new matting algorithm in Chapter 4 called *Compositional Matting*, which combines these two steps together into a single optimization process, in order to produce more natural composites with less user input for complex images. In the compositional matting setting we treat both the original image and the new background image as input, and directly solve for the final composite. The key idea behind this approach is that if the new background is very similar to the old background, estimating an accurate alpha matte is unnecessary and often introduces additional visual artifacts. Instead, a smooth transition will produce a natural composite. The compositional matting algorithm is able to separate these regions where the old and new backgrounds are similar from other ones where the old and new backgrounds are different, and apply different optimizations in these two types of regions to automatically generate the composite with least amount of visual artifacts.

8.1.2 Foreground Extraction from Video

Extracting foreground objects from video sequences requires not only accurate foreground segmentation on each individual frames, but more importantly, consistent segmentation across adjacent frames to avoid jittery results. In other words, video segmentation requires spatio-temporal smoothness, which makes it a harder problem than just a sequence of image segmentations. In Chapter 5 we present a novel interactive video cutout system to achieve this goal. The major novelty of the system lies in the fact that it treats a video sequence as a spatio-temporal 3D data set, instead of a series of 2D image frames which is commonly used in previous video processing systems. To allow the user to interact with the 3D video cube, we present a novel user interface where the user can manipulate and slice the video data at any view point, and indicate foreground and background regions within the video by painting of surfaces within the spatio-temporal volume. To allow the system to generate a segmentation result in interactive time (a few seconds), we propose a hierarchical mean-shift over-segmentation preprocess, which builds a hierarchical data structure of the video on which the optimization methods can be applied efficiently. We also define new spatially local color and edge models within a graph-cut framework combined with the global models in previous work. By combining the hierarchical data structure, the novel user interface and the improved graph-cut optimization, our system allows the user to interactively and efficiently extract the foreground object from a video sequence.

8.1.3 Applications

Once the foreground objects has been extracted, a variety of applications can be applied on them. The most straightforward one is to re-insert the foreground onto a new background to create a novel composite. We have shown a number of still composites we created in Chapter 3 and Chapter 4, and some video composites in Chapter 5.

We have applied more interesting applications on video objects. In Chapter 6 we develop a video tooning system to transfer a video sequence into a non-photorealistic cartoon style. To achieve this we process the semantic 3D regions generated by the cutout system and transfer them into polyhedral surface representations. We further create a set of edge sheets from portions of surfaces as rendering primitives. These primitives, when sliced at a frame time yield solid areas and curves along their

edges and within the interior. We then apply a variety of rendering options for these primitives to construct a frame of the stylized video.

The tooning system provides an interactive tool to stylize the appearance of the extracted video objects. In Chapter 7 we propose a cartoon animation filter for stylizing motion. The filter is simple yet general, but can be used to produce anticipation, follow-through and squash and stretch in a unified framework, and thus enhance and enliven existing motions. We apply this filter to the extracted video objects, and then re-compose the modulated version back to the original footage to create a novel video where the motion of objects is more alive.

8.2 Future Work

Many open questions remain, as well as exciting avenues for future work. We feel that the techniques and methods we propose in this dissertation can be used as basis for more intelligent and easier-to-use systems in the future.

8.2.1 Interactive v.s. Automatic

In our current systems we require the user to roughly specify where the foreground object is in an image or video sequence. The user input provides constraints and semantic information about the foreground, which leads the algorithm to the segmentation that is consistent with what the user desires. However, this may not be desirable in some cases. For example, if the user has a large number of images with similar foreground objects, specifying foreground in each image is tedious to do.

Learning-based object recognition is another active research area in the field of computer vision and machine learning. Given a small set of training images of a target object, object recognition systems can extract good features, such as shape, color and texture, that best explain the object, and use these features to train discriminant models of the object. Given a test image, these systems will analyze the features extracted from it and use the discriminant models to determine whether or not the object is presented in the image, and the rough location of it if presented.

We can potentially build a system that combines object recognition techniques with our segmentation methods for a specific type of foreground objects. Given a set of images with similar

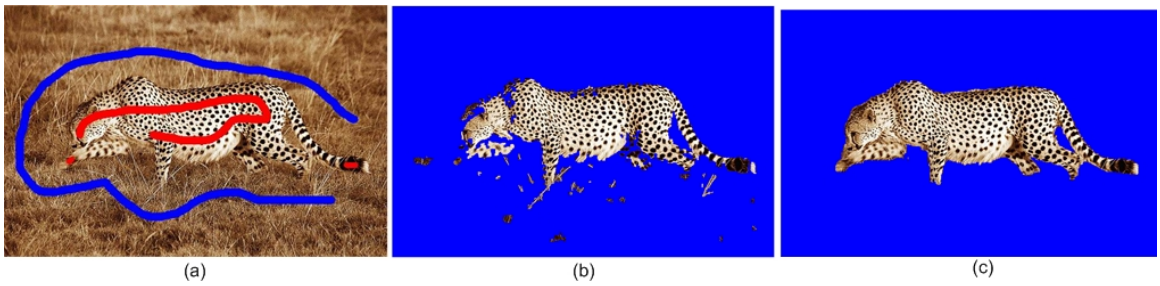


Figure 8.1: (a). Original image. (b). Graph-cut segmentation using only color as the feature. (c). Graph-cut segmentation using both color and texture as the feature. Note that the segmentation is significantly improved by using texture information.

foregrounds, we can first perform our interactive segmentation system to generate training examples. Then, we apply object recognition techniques to build models of the foreground. Then, for a new test image, we first apply object recognition to roughly locate the object, and use it as input to our segmentation algorithms. In this way we can achieve fully automatic foreground extraction.

8.2.2 Extracting Good Features

Although the systems we propose in this dissertation are robust and generally work well in most of the test examples we have tried, they do not always work as desired. One major reason is that most of the systems use color as the only feature for a pixel. These system thus will fail when the foreground and the background have very similar colors, as shown in Figure 4.10.

We could potentially improve our systems by introducing more features, for instance texture, to better discriminate the foreground from the background. An example is shown in Figure 8.1. The leopard in Figure 8.1(a) has very similar colors to the background thus if we only use color as feature, it is hard to segment it from the background, as shown in Figure 8.1(b). However, the texture of the leopard is significantly different from the background thus using texture information can achieve a better segmentation, as shown in Figure 8.1(c). One can imagine improving our systems by introducing texture information for better segmentation of such types of foreground objects.

8.2.3 Matting on Video

The interactive video cutout system we proposed in Chapter 5 employs a border matting method to make a smooth transition from the foreground to the background. It assumes a strong profile on the alpha matte, thus can only deal with smooth foreground edges and cannot handle large fuzzy regions. One can imagine a hybrid system that uses the cutout system to create a trimap for the foreground, and use robust matting algorithm to estimate an accurate matte in each frame.

Ideally, if the matting result on each single frame is accurate, the resulting video should be temporally smooth naturally. However, since in practice matting errors may be present, explicit temporal-smoothness term might be needed in the matting step to ensure the results are temporally smooth. We feel that there is a trade-off to be made between the accuracy on each single frame and the temporal-smoothness of the results. Furthermore, the smoothness assumption may be able to help us correct matting errors in problematic frames using adjacent good frames.

8.2.4 GPU Implementation

Many aspects of our systems can be potentially implemented on the GPU to further reduce the running time and give the user more rapid feedback. In particular, the large linear systems in the Random Walk solver described in Chatper 2, 3 and 4 could be solved on the GPU by adopting the methods described in [13]. However, the Laplacian matrices in our linear systems contain 27 non-zero sub-bands, which is impractical to solve by directly using existing approaches. Special optimization must be developed to make this happen.

8.2.5 Other Applications

We have shown in this dissertation a number of applications that can be applied on the extracted foreground objects, such as compositing the foreground onto a novel background, stylizing the appearance of the foreground object as well as the motion of it.

There are still many other areas that we have not explored. For example, for video sequences once the foreground object is extracted from a video sequence, we can compress and encode the foreground and background into different layers for object-based encoding. Furthermore we can choose different bitrates for the foreground and the background for low bandwidth communication.

Separating foreground from background can also help achieve a more accurate motion estimation thus may result in a higher compression ratio.

For still images, extracted foregrounds can be used as semantic representations, which will definitely benefit image query and search tasks. We can also stylize the image on the object level using similar rendering techniques we used in the Video Tooning system.

8.2.6 Authenticity of Digital Media

Our systems provide a large freedom for users to edit and manipulate their digital photographs and video. This turns the functionality of digital photographs and video from objective recording to subjective artistic expression. However, this may not be always desired, since most people rely on these digital media to learn what is happening worldwide. People thus want the photographs in the newspaper, the news clips playing on the TV to be original, un-manipulated.

This raises a new problem of how to evaluate the authenticity of digital photographs and video we receive. As shown in many examples in this dissertation, our systems are capable of seamlessly compose the foreground onto a new background without noticeable artifacts, which makes the problem a much harder one. An initial attempt to solve this problem was made in [57], where given a JPEG image, the double quantization effect hidden among the DCT coefficients is examined to determine whether or not the image has been manipulated. However this approach could be easily disarmed by a simple image resizing. We expect solving this problem to become a new research topic in the near future, and our systems can be used as test tools in this area.

BIBLIOGRAPHY

[1] Aseem Agarwala. Snaketoonz : A semi-automatic approach to creating cel animation from video. In *Proceedings of NPAR 2002*, 2002.

[2] Aseem Agarwala. SnakeToonz: A Semi-Automatic Approach to Creating Cel Animation from Video. In *NPAR 2002: Second International Symposium on Non Photorealistic Rendering*, pages 139–146, June 2002.

[3] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. In *Proceedings of ACM SIGGRAPH*, pages 294–302, 2004.

[4] Aseem Agarwala, Aaron Hertzmann, David H. Salesin, and Steven M. Seitz. Keyframe-based tracking for rotoscoping and animation. In *Proceedings of ACM SIGGRAPH*, pages 584–591, 2004.

[5] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, January 1989.

[6] William A. Barrett and Alan S. Cheney. Object-based image editing. In *Proceedings of ACM SIGGRAPH*, pages 777–784, 2002.

[7] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 2002.

[8] E.P. Bennett and L. Mcmillan. Proscenium: A framework for spatio-temporal video editing. In *Proceedings of ACM Multimedia*, pages 177–183, 2003.

[9] P.J. Besl and N.D. McKay. A method for registration of 3-d shapes. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[10] M. J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, 1996.

[11] Michael J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, 1996.

[12] Andrew Blake and Michael Isard. *Active Contours*. Springer-Verlag, 1998.

[13] Jeff Bolz, Ian Farmer, Eitan Grinspun, and Peter Schiroder. Sparse matrix solvers on the gpu: conjugate gradients and multigrid. In *Proceedings of ACM SIGGRAPH*, 2003.

[14] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[15] Armin Bruderlin and Lance Williams. Motion signal processing. In *Proceedings of SIGGRAPH 95*, pages 97–104, 1995.

[16] Ian Buck, Adam Finkelstein, Charles Jacobs, Allison Klein, David H. Salesin, Joshua Seims, Richard Szeliski, and Kentaro Toyama. Performance-Driven Hand-Drawn Animation. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 101–108, June 2000.

[17] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *CACM*, 19:564–569, October 1976.

[18] Vincent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *International Journal of Computer Vision*, 22(1):61–79, 1997.

[19] Stephen Chenney, Mark Pingel, Rob Iverson, and Marcin Szymanski. Simulating cartoon style animation. In *NPAR 2002: Second International Symposium on Non-Photorealistic Rendering*, pages 133–138, 2002.

[20] C. Christoudias, B. Georgescu, and P. Meer. Synergism in low-level vision. In *Porc. of 16th International Conference on Pattern Recognition*, pages 150–155, Quebec City, Canada, 2002.

[21] Yung-Yu Chuang, Aseem Agarwala, Brian Curless, David Salesin, and Richard Szeliski. Video matting. In *Proceedings of ACM SIGGRAPH*, pages 243–248, 2002.

[22] Yung-Yu Chuang, Aseem Agarwala, Brian Curless, David H. Salesin, and Richard Szeliski. Video matting of complex scenes. *ACM Transactions on Graphics*, 21(3):243–248, 2002.

[23] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *Proceedings of IEEE CVPR*, pages 264–271, 2001.

[24] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, volume 2, pages 264–271, December 2001.

[25] Michael F. Cohen. Interactive spacetime control for animation. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 293–302, 1992.

[26] J. P. Collomosse, D. Rowntree, and P. M. Hall. Stroke surfaces: A spatio-temporal framework for temporally coherent non-photorealistic animations. *University of Bath, Technical Report CSBU 2003-01 (June 2003)*, 2003.

[27] John Collomosse. *Higher Level Techniques for the Artistic Rendering of Images and Video*. PhD thesis, University of Bath, 2004.

[28] D. Comaniciu. An algorithm for data-driven bandwidth selection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(2):281–288, 2003.

[29] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[30] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Porc. of IEEE Conf. on Comp. Vis. and Pat. Rec (CVPR00)*, pages 142–151, San Francisco, CA, 2000.

[31] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. The variable bandwidth mean shift and data-driven scale selection. In *Proc. IEEE 8th Int. Conf. on Computer Vision*, Canada, 2001.

[32] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models — their training and application. *Computer Vision and Image Understanding*, 61(1):38–59, 1995.

[33] COREL CORPORATION. Knockout user guide. 2002.

[34] Paolo De Lucia. Personal communication, 2002.

[35] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH 97*, pages 369–378, 1997.

[36] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics*, 22(3):848–855, 2003.

[37] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. In *Proceedings of SIGGRAPH 2002*, pages 769–776, 2002.

[38] Doug DeCarlo and Anthony Santella. Stylization and abstraction of photographs. *ACM Transactions on Graphics*, 21(3):769–776, July 2002.

[39] Douglas DeCarlo and Dimitris Metaxas. Optical flow constraints on deformable models with applications to face tracking. *International Journal of Computer Vision*, 38(2):99–127, 2000.

[40] Daniel DeMenthon. Spatio-temporal segmentation of video by hierarchical mean shift analysis. In *Porc. of Statistical Methods in Video Processing Workshop*, Copenhagen, Denmark, 2002.

[41] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, July - September 1997.

[42] Jean-Daniel Fekete, Érick Bizouarn, Éric Cournarie, Thierry Galas, and Frédéric Taillefer. Tictactoon: A paperless system for professional 2D animation. In *Proceedings of SIGGRAPH 95*, pages 79–90, 1995.

[43] S. Fels and K. Mase. Interactive video cubism. In *Proc. of the Workshop on New Paradigms for Interactive Visualization and Manipulation (NPIVM)*, pages 78–82, 1999.

[44] Max Fleischer. Method of Producing Moving Picture Cartoons, 1917. US Patent no. 1,242,674.

[45] M. S. Floatern. Mean value coordinates. *Comp. Aided Geom. Design*, 20:19–27, 2003.

[46] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice in C, 2nd Edition*. Addison Wesley, 1997.

[47] K. Fukunaga and L.D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. Information Theory*, 21:32–40, 1975.

[48] Michael Gleicher. Image snapping. In *Proceedings of SIGGRAPH 95*, pages 183–190, 1995.

[49] Daniel Goldman. Computer graphics supervisor, Industrial Light & Magic, personal communication, 2003.

[50] Leo Grady. Multilabel randomwalker image segmentation using prior models. In *Proceedings of IEEE CVPR*, pages 763–770, 2005.

[51] Leo Grady. Random walks for image segmentation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2006.

[52] Leo Grady, Thomas Schiwietz, Shmuel Aharon, and Rudiger Westermann. Random walks for interactive alpha-matting. In *Proceedings of VIIP 2005*, pages 423–429, 2005.

[53] Yu Guan, Wei Chen, Xiao Liang, Ziang Ding, and Qunsheng Peng. Easy matting. In *Proceedings of Eurographics*, 2006.

[54] J. Hall, D. Greenhill, and G. Jones. Segmenting film sequences using active surfaces. In *International Conference on Image Processing (ICIP)*, pages 751–754, 1997.

[55] A. Hampapur, R. Jain, and T. Weymouth. Digital video segmentation. In *Proc. ACM Multimedia 94*, pages 357–364, San Francisco, CA, 1994. ACM.

[56] Michael Harville. A framework for high-level feedback to adaptive, per-pixel, mixture-of-gaussian background models. *IEEE Trans. Image Processing*, pages 1459–1472, 2004.

[57] Junfeng He, Zhouchen Lin, Lifeng Wang, , and Xiaoou Tang. Detecting doctored jpeg images via dct coefficient analys. In *Proceedings of IEEE CVPR*, 2006.

[58] A. Hertzmann. Paint by relaxation. In *Proc. Computer Graphics International 2001*, pages 47–54, 2001.

[59] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. In *Proceedings of NPAR 2000*, pages 7–12, 2000.

[60] Aaron Hertzmann and Ken Perlin. Painterly rendering for video and interaction. In *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 7–12, June 2000.

[61] Michael Hoch and Peter C. Litwinowicz. A semi-automatic system for edge tracking with snakes. *The Visual Computer*, 12(2):75–83, 1996.

[62] S. C. Hsu, I. H. H. Lee, C. W. Lou, and S. L. Siu. Software. *Creature House (www.creaturehouse.com)*, 1999.

[63] S. C. Hsu, I. H. H. Lee, and N. E. Wiseman. Skeletal strokes. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 197–206. ACM Press, 1993.

[64] Siu Chi Hsu and Irene H. H. Lee. Drawing and animation using skeletal strokes. In *Proceedings of SIGGRAPH '94*, pages 109–118, 1994.

[65] Takeo Igarashi, Tomer Moscovich, and John F. Hughes. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics*, 24(3):1134–1141, 2005.

[66] ADOBE SYSTEMS INCORP. Adobe photoshop user guide. 2002.

[67] Michal Irani. Multi-frame correspondence estimation using subspace constraints. *International Journal of Computer Vision*, 48(3):173–194, 2002.

[68] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of computer vision*, 29(1):5–28, 1998.

[69] Jiaya Jia, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Drag-and-drop pasting. In *Proceedings of ACM SIGGRAPH*, 2006.

[70] Ollie Johnston and Frank Thomas. *The Illusion of Life: Disney Animation*. Disney Editions, 1995.

[71] Nebojsa Jojic and Brendan J. Frey. Learning flexible sprites in video layers. In *2001 Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, volume 1, pages 199–206, 2001.

[72] N. Joshi, W. Matusik, and S. Avidan. Natural video matting using camera arrays. In *Proceedings of ACM SIGGRAPH*, pages 779–786, 2006.

[73] Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. *ACM Transactions on Graphics*, 21(3):755–762, July 2002.

[74] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

[75] Allison W. Klein, Peter-Pike J. Sloan, Adam Finkelstein, and Michael F. Cohen. Stylized video cubes. In *Proceedings of SCA 2002*, 2002.

[76] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimizedvia graph cuts? *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(2):147–159, 2004.

[77] Alexander Kort. Computer aided inbetweening. In *NPAR 2002: Second International Symposium on Non Photorealistic Rendering*, pages 125–132, 2002.

[78] V. Kwatra, A. Shoedl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. In *Proceedings of ACM SIGGRAPH*, pages 277–286, 2003.

[79] John Lasseter. Principles of traditional animation applied to 3d computer animation. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, pages 35–44, 1987.

[80] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. In *Proceedings of ACM SIGGRAPH*, pages 689–694, 2004.

[81] Anat Levin, Dani Lischinski, and Yair Weiss. A closed form solution to natural image matting. In *Proceedings of IEEE CVPR*, 2006.

[82] Liyuan Li, Weimin Huang, Irene Yu-Hua Gu, and Qi Tian. Statistical modeling of complex backgrounds for foreground object detection. *IEEE Trans. Image Processing*, pages 1459–1472, 2004.

[83] Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. Lazy snapping. In *Proceedings of ACM SIGGRAPH*, pages 303–308, 2004.

[84] Richard Linklater, 2001. *Waking Life*. 20th Century Fox.

[85] Richard Linklater. Waking Life DVD. *Twentieth Century Fox Home Video*, 2001.

[86] Dani Lischinski, Zeev Farbman, Matt Uyttendaele, and Richard Szeliski. Interactive local adjustment of tonal values. In *Proceedings of ACM SIGGRAPH*, 2006.

[87] Peter Litwinowicz. Processing images and video for an impressionist effect. In *Proceedings of SIGGRAPH 1997*, Computer Graphics Proceedings, Annual Conference Series, pages 151–158. ACM, ACM Press / ACM SIGGRAPH, 1997.

[88] Peter Litwinowicz. Processing Images and Video for an Impressionist Effect. In *Proceedings of SIGGRAPH 97*, pages 407–414, August 1997.

[89] Peter Litwinowicz and Lance Williams. Animating images with drawings. In *Proceedings of SIGGRAPH 94*, pages 409–412, 1994.

[90] Feng Liu and Michael Gleicher. Automatic image retargeting with fisheye-view warping. *Proceedings of ACM UIST*, pages 153–162, 2005.

[91] W.E. Lorensen and H.E. Cline. Marching cubes: a high resolution 3d surface reconstruction algorithm. In *Proceedings of SIGGRAPH 1987*, pages 163–169, 1987.

[92] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 674–679, April 1981.

[93] H. Luo and A. Eleftheriadis. Spatial temporal active contour interpolation for semi-automatic video object generation. In *International Conference on Image Processing (ICIP)*, pages 944–948, 1999.

[94] Leonard Maltin. *Of Mice and Magic: A History of American Animated Cartoons*. McGraw-Hill Book Company, 1980.

[95] Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Loring S. Holden, J. D. Northrup, and John F. Hughes. Art-based rendering with continuous levels of detail. In *Proceedings of NPAR 2000*, 2000.

[96] Morgan McGuire, Wojciech Matusik, Hanspeter Pfister, John F. Hughes, and Fredo Durand. Defocus video matting. In *Proceedings of ACM SIGGRAPH*, pages 567–576, 2005.

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

[98] M. Meyer, H. Lee, A. Barr, and M. Desbrun. Generalized barycentric coordinates on irregular polygons. *Journal of Graphics Tools*, 7(1):13–22, 2002.

[99] Y. Mishima. Soft edge chroma-key generation based upon hexoctahedral color space. In *U.S. Patent 5,355,174*, 1993.

[100] Tomoo Mitsunaga, Taku Yokoyama, and Takashi Totsuka. AutoKey: Human assisted key extraction. In *SIGGRAPH 95 Conference Proceedings*, pages 265–272, 1995.

[101] E. Mortensen and W. Barrett. Intelligent scissors for image composition. In *Proceedings of ACM SIGGRAPH*, pages 191–198, 1995.

[102] E. N. Mortensen. Vision-assisted image editing. 33(4):55–57, November 1999.

[103] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[104] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew P. Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of ACM SIGGRAPH 2000*, pages 209–218, 2000.

[105] P. PREZ, A. BLAKE, and M. GANGNET. Jetstream: Probabilistic contour extraction with particles. In *Proc. Int. Conf. on Computer Vision*, pages 524–531, 2001.

[106] Patrick Prez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *Proceedings of ACM SIGGRAPH*, pages 313–318, 2003.

[107] L. J. Reese. Intelligent paint: Region-based interactive image segmentation. Master's thesis, Department of Computer Science, Brigham Young University, Provo, UT, 1999.

[108] Barbara Robertson. Life Lines. *Computer Graphics World*, 25(2):12–17, 2002.

[109] Gary Ross, 1998. *Pleasantville*. New Line Cinema.

[110] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut - interactive foreground extraction using iterated graph cut. In *Proceedings of ACM SIGGRAPH*, pages 309–314, 2004.

[111] M.A. Ruzon and C. Tomasi. Alpha estimation in natural images. In *Proceedings of IEEE CVPR*, pages 18–25, 2000.

[112] Bob Sabiston. Bob Sabiston's animation software tutorial. In *Waking Life DVD*, additional feature, 2001.

[113] Philip J. Schneider. An algorithm for automatically fitting digitized curves. In *Graphics Gems*, pages 612–626, 797–807. 1990.

[114] Thomas W. Sederberg, Peisheng Gao, Guojin Wang, and Hong Mu. 2D shape blending: An intrinsic solution to the vertex path problem. In *Proceedings of SIGGRAPH 93*, pages 15–18, 1993.

[115] Thomas W. Sederberg and Eugene Greenwood. A physically based approach to 2D shape blending. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, pages 25–34, July 1992.

[116] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation, computational geometry: Theory and applications. *Computational Geometry: Theory and Applications*, 22(1-3):21–74, 2002.

[117] Alvy Ray Smith and James F. Blinn. Blue screen matting. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 259–268, 1996.

[118] A.R. Smith and J.F. Blinn. Blue screen matting. In *Proceedings of ACM SIGGRAPH*, pages 259–268, 1996.

[119] Scott Stewart. Confessions of a roto artist: Three rules for better mattes, 2003. http://www.pinnaclesys.com/SupportFiles/Rotoscoping.pdf.

[120] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. In *Proceedings of ACM SIGGRAPH*, pages 315–321, 2004.

[121] Jian Sun, Yin Li, Sing-Bing Kang, and Heung-Yeung Shum. Flash matting. In *Proceedings of ACM SIGGRAPH*, 2006.

[122] Jian Sun, Lu Yuan, Jiaya Jia, and Heung-Yeung Shum. Image completion with structure propagation. In *Proceedings of ACM SIGGRAPH*, pages 861–868, 2005.

[123] R. Szeliski. Fast surface interpolation using hierarchical basis functions. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, 12(6):513–528, 1990.

[124] D. Terzopoulos and R. Szeliski. Tracking with kalman snakes. In A. Blake and A. Yuille, editors, *Active Vision*, pages 3–20. MIT Press, Cambridge, MA, 1992.

[125] B. Thomas. *Walt Disney: An American Original*. Fireside Books (Simon and Schuster), 1976.

[126] L. Torresani and C. Bregler. Space-time tracking. In *European conference on computer vision*, pages 801–802, 2002.

[127] L. Torresani, D.B. Yang, E.J. Alexander, and C. Bregler. Tracking and modeling non-rigid objects with rank constraints. In *Computer Vision and Pattern Recognition*, pages 493–500, 2001.

[128] Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. Keyframe control of smoke simulations. *ACM Transactions on Graphics*, 22(3):716–723, 2003.

[129] Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier principles for emotion-based human figure animation. In *Proceedings of SIGGRAPH 95*, pages 91–96, 1995.

[130] Vladimir Vezhnevets and Vadim Konouchine. Grow-cut - interactive multi-label n-d image segmentation. In *Proceedings of GraphiCon 2005*, 2005.

[131] M. Wand and M. Jones. *Kernel Smoothing*. Chapman & Hall, 1995.

[132] Jue Wang, Maneesh Agrawala, and Michael Cohen. Soft scissors: an interactive tool for realtime high quality matting. In *Proceedings of ACM SIGGRAPH*, 2007.

[133] Jue Wang, Pravin Bhat, Alex Colburn, Maneesh Agrawala, and Michael Cohen. Interactive video cutout. In *Proceedings of ACM SIGGRAPH*, 2005.

[134] Jue Wang and Michael Cohen. An iterative optimization approach for unified image segmentation and matting. In *Proceedings of ICCV 2005*, pages 936–943, 2005.

[135] Jue Wang and Michael Cohen. Optimized color sampling for robust matting. In *Proceedings of IEEE CVPR*, 2007.

[136] Jue Wang and Michael Cohen. Simultaneous matting and compositing. In *Proceedings of IEEE CVPR*, 2007.

[137] Jue Wang, Steven Drucker, Maneesh Agrawala, and Michael Cohen. The cartoon animation filter. In *Proceedings of ACM SIGGRAPH*, 2006.

[138] Jue Wang, Yingqing Xu, Harry Shum, and Michael Cohen. Video tooning. In *Proceedings of ACM SIGGRAPH*, 2004.

[139] Y. Weiss and W.T. Freeman. On the optimality of solutions of the max-product belief propagation algorithm in arebitrary graphs. *IEEE Trans. on Information Theory*, 47(2):303–308, 2001.

[140] Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 22, pages 159–168, 1988.

[141] Brian Wyvill. *Animation and Special Effects*, chapter 8, pages 242–269. Morgan Kaufmann, 1997.

[142] Hitoshi Yamauchi, Jorg Haber, and Hans-Peter Seidel. Image restoration using multiresolution texture synthesis and image inpainting. *Proceedings of Computer Graphics International*, pages 120–125, 2003.

[143] Li Zhang, Guillaume Dugas-Phocion, Jean-Sebastien Samson, and Steven M. Seitz. Single view modeling of free-form scenes. In *2001 Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, volume 1, pages 990–997, 2001.

[144] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. In *Proceedings of ACM SIGGRAPH*, pages 600–608, 2004.

VITA

Jue Wang was born and grew up in Mianyang, Sichuan Province, southwestern of China. He moved to Beijing to attend Tsinghua University, where he obtained his B.E. and M.S. in Department of Automation, with a major in Pattern Recognition and Intelligent Systems, in 2000 and 2003. He then moved to Seattle to join the Electrical Engineering Department at the University of Washington, and received his PhD in autumn 2007. He then joined Adobe Systems Incorporated as a research scientist.