2010-08-10

Interactive Object Selection and Matting for Video and Images

Brian L. Price
Brigham Young University - Provo

Interactive Object Selection and Matting for Video and Images

Brian L. Price

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Bryan S. Morse, Chair
Parris K. Egbert
Dan R. Olsen
Michael D. Jones
William A. Barrett

Department of Computer Science

Brigham Young University

December 2010

ABSTRACT

Interactive Object Selection and Matting for Video and Images

Brian L. Price

Department of Computer Science

Doctor of Philosophy

Video segmentation, the process of selecting an object out of a video sequence, is a fundamentally important process for video editing and special effects. However, it remains an unsolved problem due to many difficulties such as large or rapid motions, motion blur, lighting and shadow changes, complex textures, similar colors in the foreground and background, and many others. While the human vision system relies on multiple visual cues and higher-order understanding of the objects involved in order to perceive the segmentation, current algorithms usually depend on a small amount of information to assist a user in selecting a desired object. This causes current methods to often fail for common cases. Because of this, industry still largely relies on humans to trace the object in each frame, a tedious and expensive process.

This dissertation investigates methods of segmenting video by propagating the segmentation from frame to frame using multiple cues to maximize the amount of information gained from each user interaction. New and existing methods are incorporated in propagating as much information as possible to a new frame, leveraging multiple cues such as object colors or mixes of colors, color relationships, temporal and spatial coherence, motion, shape, and identifiable points. The cues are weighted and applied on a local basis depending on the reliability of the cue in each region of the image. The reliability of the cues is learned from any corrections the user makes. In this framework, every action of the user is examined and leveraged in an attempt to provide as much information as possible to guarantee a correct segmentation. Propagating segmentation information from frame to frame using multiple cues and learning from the user interaction allows users to more quickly and accurately extract objects from video while exerting less effort.

Keywords: Video segmentation, image segmentation, image matting, color modeling

ACKNOWLEDGMENTS

I thank my professors, William Barrett and Bryan Morse, for all their help and guidence throughout my schooling. Dr. Barrett started me along this path and taught me much about what it takes to be a researcher. Dr. Morse has continued this and especially helped me in finishing my work and producing my published papers. I appreciate his always being there to talk over ideas and questions.

Another individual who deserves my thanks is Scott Cohen. He has been extremely helpful in providing ideas and always made himself available. He contributed a lot to my papers, both in helpimg me to see the high-level, big picture ideas as well as getting down into the gritty details of the math and methods and make sure that the details are there.

I would also like to thank my gradutate committee for their suggestions for my work and the time they put in reading my work. Additionally, I thank Adobe Systems for their funding.

Most of all, I'd like to thank my beautiful wife Lesley for her support and encouragement. She has always been their for me, and has been even more excited than I have been about my publications and success. Lesley, I love you and I'm so grateful that you are in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Video segmentation is the process of selecting an object in every frame of a video sequence, as illustrated in Figure 1.1. Video segmentation is fundamentally important in many applications involving video. For example, video editing and special effects rely heavily on video segmentation. In order to delete an object from a scene or place an object from one video sequence into another, the object must be segmented. The brightness or color of an object may need to be changed without changing the overall scene, which requires segmenting the desired object first. Video segmentation is used to divide a scene into layers so that new objects (real or synthetic) may be placed between the segmented layers to suggest their depth. Video segmentation is also involved in many other applications such as object recognition, 3D reconstruction from video, video compression, etc. In fact, any problem that requires knowledge of specific objects in video also requires video segmentation.

Selecting an object in a video sequence is a surprisingly difficult problem. Video segmentation shares the difficulties inherent in image and volume segmentation. The colors of the object and background may be similar, which creates weak edges and confuses color models. Blurring also degrades the edges between the object and background. Complex textures may introduce many gradients that will confuse edge detectors. Compression artifacts may make it difficult to identify the true edge. In addition to these difficulties common to segmentation in images and volumes, video has many other problematic factors that frequently occur. The object may move enough relative to its size that there is no overlap between successive frames. Other objects moving in the background may cause confusion. The camera

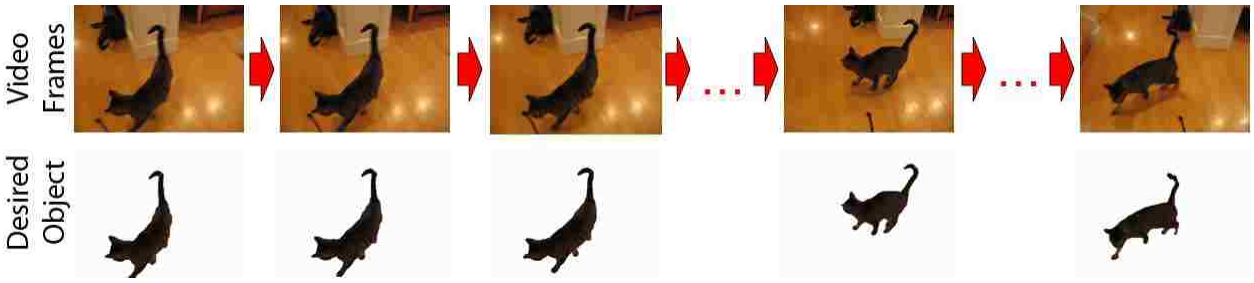


Figure 1.1: Video segmentation example. The top row of images are the frames from the video sequence. The selected object, the cat, is shown in the row below.

movement may be erratic. Lighting changes and shadows may alter the color properties of the object and background, while even simple movements in 3D space may greatly change the 2D object boundary shape. Motion blur destroys object edges and contaminates the foreground objects. Occlusions may temporarily hide pieces of the object. Each of these problems is quite common, and a given video sequence can easily exhibit most or all of these qualities, making video segmentation a difficult problem.

Because of the inherent difficulty of the problem, video segmentation algorithms are often interactive, with a user indicating the desired object, generally by placing a stroke over the desired object or by drawing around the border of the object. The algorithm then determines the location of that object throughout the video. It may accomplish this by solving the segmentation on a spatiotemporal volume, much like in volume segmentation, or by selecting the object in the current frame using an image segmentation approach and then propagating the segmentation to other frames. In an interactive paradigm, the key goal of video segmentation is to minimize the work the user must perform in order to select the object.

There are currently many methods for segmenting video. Some methods, like Keyframe Rotoscoping [1] and Snakes [32], track the boundary of the object using shape and color cues. Methods based on graph cut [9, 10], such as Interactive Video Cutout [80], Video Object Cut and Paste [41], and Live Surface [2], use regional color information and

gradients (edges) to formulate the segmentation as a graph problem. The geodesic framework of [4] uses color as a basis for geodesic distance for computing the segmentation.

While many approaches exist for segmenting video, they are still largely inadequate. Most techniques are too slow and require excessive preprocessing time (for example, Interactive Video Cutout [80] requires about 30 minutes of preprocessing time and 20-30 minutes of user interaction time to segment 2-3 seconds of video, with a $\sim$10 second delay between each user interaction). Current techniques also cannot handle most of the problem cases often seen in video, making them very inaccurate and requiring excessive amounts of user interaction to correct the problems. Because of this, industry often resorts to hiring artists to manually segment objects in video by drawing curves around their boundary, a tedious, time-consuming, expensive task. A ten-second video clip can easily take days or even weeks to segment by hand, costing industry a great deal of money and tying up artists in menial tasks.

A key reason why current segmentation algorithms fall short is that current methods use only a few of the many informative cues available when deciding the segmentation. For example, in Figure 1.2, the ballerina's feet have a similar color to the floor, so color cues will give little information about the selection, while the shape and location of the feet are more reliable. The ballerina's arm, on the other hand, moves such that much of it is not overlapping between frames, rendering spatiotemporal coherence cues less effective. If an algorithm utilizes a limited amount of information such as color and gradient (as many graph-cut approaches do), then it would struggle on areas such as the feet of the ballerina.

For a high-quality segmentation, computing a selection by labeling each pixel as either object or background is insufficent. For many pixels, the color of the pixel does not belong only to the object or to the background, but rather is a blend of foreground and background colors. This occurs when an object boundary is blurred, when the edge between two objects lies on a pixel, when an object is semi-transparent, or when an object is smaller than an individual pixel. This is illustrated for a simple case in Figure 1.3, where the pixels along a

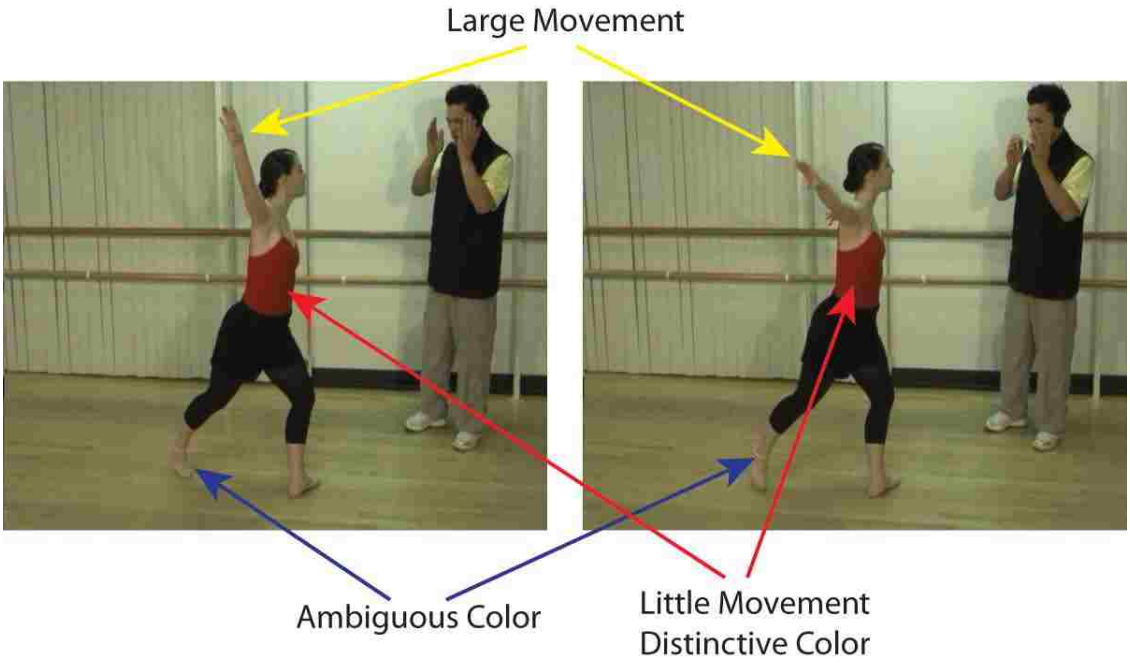


Figure 1.2: Two successive video frames are shown. Different regions of the video frames exhibit different difficulties.

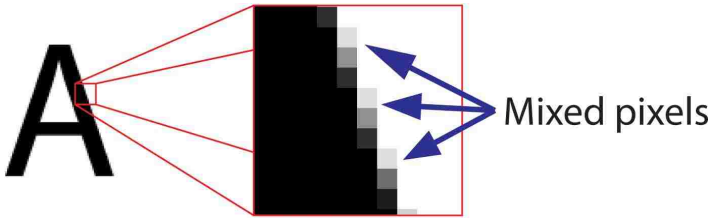


Figure 1.3: Although the letter "A" is black and the background is white in the image on the left, gray values exist on the boundary of the image. Because these pixels overlap both the letter "A" and the background, the colors of each are blended to produce the pixel color.

boundary contain the blended colors of both the black letter "A" and the white background. Without separating the foreground and background colors at a pixel many common video edits, such as those involving hair or shadows, will look unrealistic. For example, the toys in Figure 1.4 need the foreground and background colors separated to capture the hair correctly.

To address this problem, the process of *matting* computes a "soft" segmentation where the color of any pixel may be divided into foreground and background components and a blending coefficent called the *alpha value*. The equation governing matting is very simple

(a) Image (b) Alpha Matte (c) Close-up

Figure 1.4: The toys in image (a) have a lot of hair. The ground truth matte is shown in (b), with a close-up in (c). In order to correctly segment hair like in (a), matting must be performed.

since the color at a pixel is a linear combination of the foreground and background colors:

$$I = \alpha F + (1 - \alpha)B \tag{1.1}$$

where I is the color at a pixel, F is the foreground color, B is the background color, and α is the blending coefficient. Unfortunately, matting is a difficult problem to solve correctly because it is severely underconstrained, with three variables (the RGB color I) being used to solve for seven unknowns (two RGB values F and B and α) at each pixel. Many matting algorithms exist for single images, which usually require a known foreground, a known background, and an unknown region to be specified (a trimap), and then use a method to estimate the alpha values in the unknown region. Single-image matting algorithms still struggle on complex images. Generally matting is applied to video by segmenting and then applying a single image matting algorithm to each frame, which does not preserve coherency in the matte over time. Mattes for video have also been solved by treating the video as a spatiotemporal volume, which gives some consistency in the matte but can fail in many instances.

1.1 Leveraging Multiple Types of Information

This dissertation introduces a new approach to selecting objects in arbitrary video sequences accurately and with minimal user interaction. To achieve this segmentation, the system attempts to propagate as much information as possible from one frame to the next. By combining multiple types of information, this system is more likely to handle problem cases like those in Figure 1.2 than approaches using limited information.

A key factor in the success of such a system is the set of information cues that the algorithm uses. Possible cues that can be extracted from video include color, gradient, color adjacency relationships, shape, temporal coherence, camera motion, object motion, and easily-identifiable points. This research provides several novel cues as well as improvements to other cues to improve the quality of information used.

An important factor in dealing with multiple cues is how to combine the information. The most useful information can vary from sequence to sequence, from frame to frame, and even from one area of a frame to the next as shown in Figure 1.2. To account for this, the proposed system uses each piece of information in the area that it is most useful. It also utilizes user corrections to re-evaluate its local weighting between cues.

Additionally, this dissertation looks at how this approach applies to different problem domains. The approach of using better information, more information, and selectively applying the information where it is most useful can not only be used in video segmentation but also extends to related problems such as image segmentation, image matting, and multiple similar image segmentation. Several such extensions are included.

1.2 Organization

This dissertation consists of five previously published or soon to be published papers. They are organized as follows.

Chapter 2 presents a novel video segmentation system that adaptively combines multiple cues to allow for easy object selection with minimal user interaction. This chapter was published under the title *LIVEcut: Learning-based Interactive Video Segmentation by Evaluation of Multiple Propagated Cues* in the *IEEE International Conference of Computer Vision*, October 2009 [50].

Chapter 3 expands the novel cues from [50] of modeling color adjacency relationships to identify the edges desired for segmentation. The method is more fully explained and generalized and some improvements are made. The algorithm is also extended to segmented sets of similar images (where the same object is in the same scene, although some change has occurred). This chapter will appear under the title *Color Adjacency Modeling for Improved Image and Video Segmentation* in the *International Conference on Pattern Recognition*, August 2010 [51].

Chapter 4 extends the idea of selectively combining different methods to the problem of single-image segmentation. An analysis is given of the strengths and weaknesses of graph-cut segmentation methods [9, 10] and geodesic segmentation [4]. These two methods are then combined into a single formulation with the contribution of each weighted globally and locally based on the estimated performance of each term. This chapter was published under the title *Geodesic Graph Cut for Interactive Image Segmentation* in the *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2010 [52].

Chapter 5 addresses the problem of image matting. In order to properly segment hair, blurred objects, or transparent objects from video sequences, video matting is needed. Unfortunately, video matting and image matting are still largely unsolved problems. This chapter presents a novel method for computing the matte of an image while simultaneously computing the estimated foreground and background colors. By simultaneously approaching the problem of alpha estimation and foreground/background estimation, more information may be applied to help inform the algorithm. This chapter was published under the title

Simultaneous Foreground, Background, and Alpha Estimation for Image Matting in the *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 2010 [53].

Chapter 6 revisits the video segmentation system presented in Chapter 2. In this paper, additional detail, motivation, and validation are given for many of the cues and new cues are introduced. The learning method is also improved and better validated. This chapter will soon be submitted for publication.

1.3 Major Contributions

This dissertation makes many contributions to the solution of the problem of video segmentation as well as of the related problems of image segmentation and matting. The major contributions of this work are as follows.

1. It introduces a means of adaptively applying information at a pixel level to solve video segmentation problems and evaluates the application using user corrections.
2. It introduces a new color adjacency relationship model that allows a segmentation algorithm to focus on edges relevant to the selection.
3. It provides other novel or improved cues for use in segmentation, such as a local color model, point tracking information, and shape.
4. It introduces a novel image segmentation technique that leverages the strengths of geodesic segmentation and graph-cut segmentation, weighting the application of each at a global and a local level, to improve selection results.
5. It introduces a novel image matting algorithm that computes the foreground and background along with the alpha matte to provide improved foreground/background estimations while maintaining state-of-the-art results for the alpha matte.

Chapter 2

LIVEcut Video segmentation

A key goal of this research is to introduce a novel method of video segmentation that adaptively combines multiple informative cues. This chapter presents an approach to this problem as published in the *IEEE International Conference on Computer Vision* (ICCV) in October 2009 under the title *LIVEcut: Learning-based Interactive Video Segmentation by Evaluation of Multiple Propagated Cues* [50]. The cues that this system uses are briefly described, and one method of combining that information in a learning paradigm is given. Further details and improvements are presented in Chapter 6.

2.1 Abstract

Video sequences contain many cues that may be used to segment objects in them, such as color, gradient, color adjacency, shape, temporal coherence, camera and object motion, and easily-identifiable points. This paper introduces LIVEcut, a novel method for interactively selecting objects in video sequences by extracting and leveraging as much of this information as possible. Using a graph cut optimization framework, LIVEcut propagates the selection forward frame by frame, allowing the user to correct any mistakes along the way if needed. Enhanced methods of extracting many of the features are provided. In order to use the most accurate information from the various potentially-conflicting features, each feature is automatically weighted locally based on its estimated accuracy using the previous implicitly-validated frame. Feature weights are further updated by learning from the user corrections required in the previous frame. The effectiveness of LIVEcut is shown through timing com-

parisons to other interactive methods, accuracy comparisons to unsupervised methods, and qualitatively through selections on various video sequences.

2.2 Introduction

Video segmentation is an essential process in many video applications. It is required for video editing and special effects whenever objects must be moved, deleted, individually edited, or layered. It is also used in object recognition, 3D reconstruction from video, and compression. Despite recent research in the area, industry still largely relies on chroma keying and manual rotoscoping, emphasizing the need for an effective, easy-to-use video segmentation tool.

This need remains due to the surprising difficulty of the problem. Video segmentation shares the difficulties of image segmentation, such as overlapping color distributions, weak edges, complex textures, and compression artifacts. In addition to these challenges, video may contain erratic camera and/or object movement, motion blur, and occlusions. Objects may move enough that there is no overlap between successive frames. Other moving objects may cause confusion. Lighting changes and shadows alter the color distributions, and movements in 3D space may greatly change an object's 2D projected boundary. A given video sequence can easily exhibit many of these challenges.

Many different kinds of information can be gleaned from successive video frames to aid object selection. Such features include color, gradient, adjacent color relationships, shape, spatiotemporal coherence, camera motion, object motion, and trackable points. The relative importance of the cues differs depending on the sequence, the frame, and even the location in the frame. For example, in Figure 2.1 a color model can easily distinguish the cat from the light brown floor but would struggle separating the tail from the similarly-colored bag. A shape feature, however, could separate the tail and bag. An algorithm that intelligently applies all of these cues based on specific circumstances will perform better than one relying only on a subset of these cues or on a static combination of all of them.

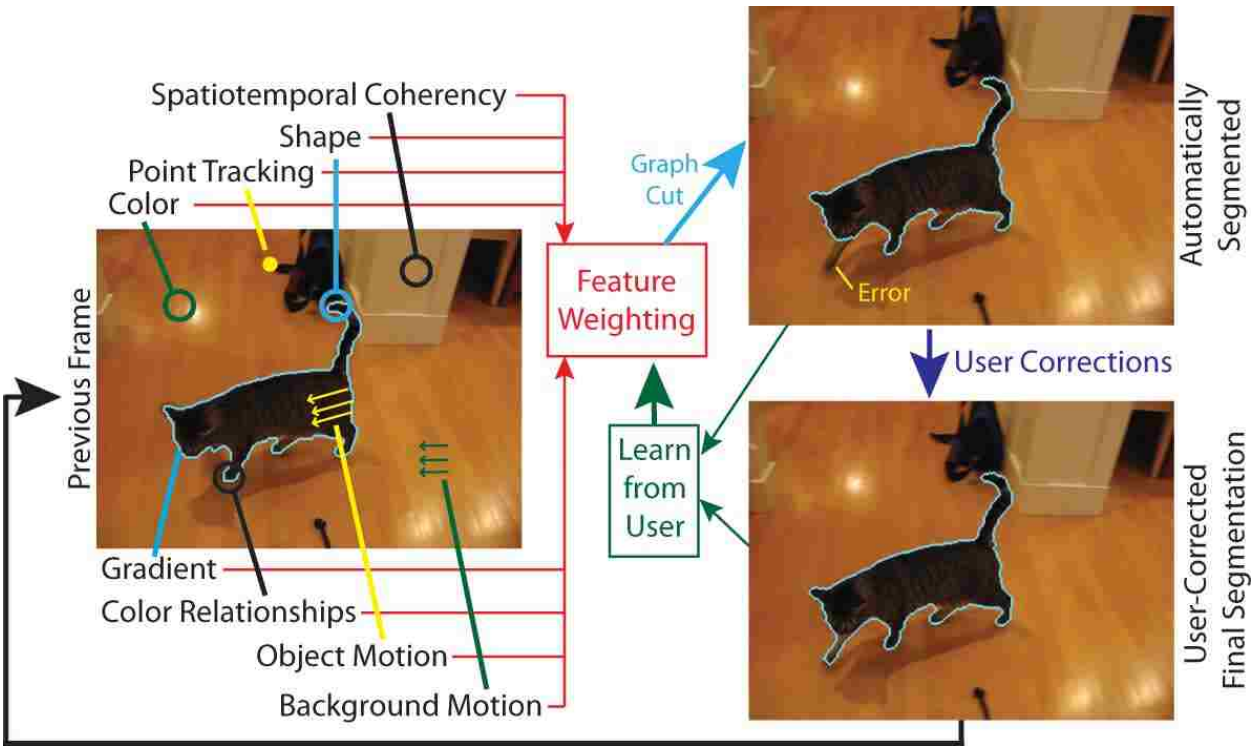


Figure 2.1: From an initial segmented frame, a variety of features are extracted. These are automatically locally weighted based on estimated correctness and used to segment the next frame. If errors occur, the user may correct them, and the system learns which features are providing good information. The corrected frame is used to continue propagating the segmentation.

Despite the importance of each kind of information, most current algorithms do not use all these features. Algorithms that segment the video as a spatiotemporal volume [3, 4, 9, 80] can generally only extract information from the pixels under the user strokes to model the foreground and background. These methods have no information about some of these features such as shape or boundary information, and have limited knowledge of other features such as foreground and background color. By allowing the user to segment one frame and then propagating this information to other frames, these features can be used.

In this paper, we introduce LIVEcut, a frame-by-frame interactive video segmentation method designed to maximize the information propagated from one frame to the next. As shown in Figure 2.1, LIVEcut extracts various features, locally weights them based on likely effectiveness, and resolves them using graph-cut optimization. LIVEcut also learns

automatically from user corrections how well each cue performed and weights their importance accordingly. Our local weighting allows LIVEcut to selectively apply the cues that will most effectively segment the object. Contributions are also made in the extraction of many of the individual cues. These include full foreground and background local color models, color adjacency models, separate foreground and background motion models, point tracking information, and a new shape prior.

This chapter is organized as follows. Section 2.3 describes related work. Section 2.4 describes the basic interaction and framework of our system, and then analyzes each informative cue used. This is followed with an explanation of how to automatically weight the different cues and learn from user corrections in Section 2.5. Section 2.6 gives our results, and our conclusion and future work are given in Section 2.7.

2.3 Related Work

Many approaches have been taken in interactive video segmentation. Some approaches focus on either boundary or region information only. Agarwala et al. [1] performs boundary tracking using splines that follow object boundaries between keyframes using both boundary color and shape-preserving terms. Other methods of boundary tracking include [31, 32]. Bai and Sapiro [4] use region color to compute a geodesic distance to each pixel to form a selection. These approaches perform well when a single type of cue is sufficient for selecting the desired object.

Many current techniques use graph cut to segment the video as a spatiotemporal volume. Graph cut, as formulated in [9], solves for a segmentation by minimizing an energy function over a combination of both region and boundary terms. It has been shown to be effective in the segmentation of images [40, 60] and volumes [3].

Boykov and Jolly [9] introduced a basic approach to segmenting video as a spatiotemporal volume. Their graph connects pixels in a volume, which implicitly includes

spatiotemporal coherence information. Graph cut is applied using a region term based on a color model of the pixels under the user strokes and a boundary term based on gradient.

Wang et al. [80] builds on this approach by allowing users to segment video by drawing strokes on arbitrary slices of the spatiotemporal volume. While this permits a user to mark several frames at once, it requires a steep learning curve to know how to carve the volume so that the right pixels are visible along the slice. The method uses a global color model based on the user strokes as well as a local color model for static backgrounds in addition to gradient values.

In Li et al. [41], users segment every tenth frame, and graph cut computes the selection between the frames using global color models from the key-frames, gradient, and coherence as its primary cues. The user may also manually indicate areas to which local color models are applied. While this method performs well, it requires the manual segmentation of many frames in addition to corrections.

In methods where the video is treated as a spatiotemporal volume [3, 4, 9, 80], the only information known for certain about the object and background are in the user-marked pixels. This provides very limited knowledge about the object interior and no knowledge about the boundary. While [41] is an exception to this, it requires the user to manually segment many frames. These methods contrast our own, where frame-by-frame propagation allows for the computation of complete features. We also provide an interactive paradigm of moving through the video sequentially, which is arguably the most natural for video.

In parallel with our own work, Yin and Collins [85] proposed an automated video segmentation system that includes color, gradient, color adjacency, and shape information in a graph cut framework. They dynamically reweight these terms from frame to frame, but do so on a global basis without regard to user corrections.

Some unsupervised video segmentation methods have also combined various cues [17, 73, 84, 78]. While unsupervised techniques generally perform well at roughly separating motion layers, they do not produce the high-quality results required for many applications.

The object of interest may also not correspond to a motion layer, leaving these methods incapable of generating the desired result.

2.4 LIVEcut Video Segmentation

While the methods described in Section 2.3 provide good means of segmenting video, each relies only on a few cues to make decisions. LIVEcut extracts much more information about the sequence and uses this to improve the segmentation. The user marks the object in the first frame of the sequence using the stroke-based method employed by most graph-cut methods, and LIVEcut propagates various cues taken from the full frame to the next frame as described in this section. These cues are automatically weighted locally and resolved using graph-cut optimization (Section 2.5). As the user proceeds through the sequence, the implicit verification of the previous frame allows LIVEcut to use the entire previous frame once again to segment the current frame.

We emphasize the importance of the frame-by-frame propagation in extracting information for segmentation. In methods where the sequence is segmented as a spatiotemporal volume [80], the only information the algorithm knows for certain about the foreground and background are the pixels located directly below the user stroke. This provides very limited knowledge about the object interior and no knowledge about the boundary characteristics. The interactive paradigm of moving through the video sequentially not only is the most natural for video, but allows the complete characteristics of each frame to be used in segmenting the next. While this is also true of keyframe-based systems such as [41], these require the user to completely segment many frames instead of allowing the algorithm to quickly compute this information for the user.

2.4.1 Graph cut framework

Before explaining the specific features we propagate from frame to frame, we present the overall framework in which the features are resolved. For this, we use minimum graph-cut

optimization. Graph cut computes a segmentation over a set of pixels P by minimizing the equation

$$E(\mathcal{L}) = \sum_{x_i \in P} R(x_i, \mathcal{L}_i) + \lambda \sum_{(x_i, x_j) \in N} B(x_i, x_j) \ |\mathcal{L}_i - \mathcal{L}_j| \tag{2.1}$$

where $\mathcal{L} = (\mathcal{L}_i)$ is a binary vector of labels and $\mathcal{L}_i$ is the label (0 for background, 1 for foreground) for pixel x_i, $R(x_i, l)$ is a region cost term based on the label l, $B(x_i, x_j)$ is a boundary cost term, λ is a relative weighting of R and B, and N is the set of pairs of neighboring pixels.

Our region term $R(x_i, l)$ is the sum of all cues that apply to an individual pixel. Given a set of unary cues U,

$$R(x_i, l) = s(x_i, l) + \sum_{u \in U} \alpha_u(x_i) \ w_u(x_i, l) \tag{2.2}$$

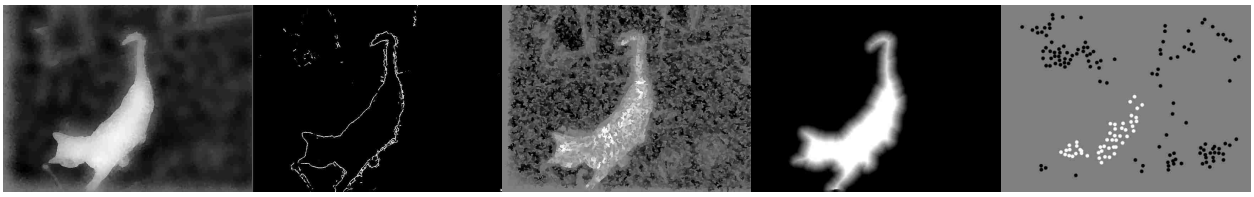
where $w_u(x_i, l)$ is the cost of labeling pixel x_i with label l according to cue u, $\alpha_u(x_i)$ is a scalar giving the certainty of w_u at x_i, and $s(x_i, l) = 0$ if the pixel was labeled l by a user stroke and ∞ if labeled $\bar{l}$ (the other label).

Our boundary term $B(x_i, x_j)$ is given by

$$B(x_i, x_j) = w_a(x_i, x_j) \ w_g(x_i, x_j). \tag{2.3}$$

and encourages selection boundaries in the current frame to occur at image edges with similar color profiles to the selection boundaries in the previous frame. The unary terms (color w_c, spatiotemporal coherency w_h, shape w_s, and point tracking w_p) and binary terms (gradient w_g and color adjacency w_a) are defined in Sections 2.4.3-2.4.8.

In order to increase the speed of the algorithm, we apply our algorithm to an oversegmentation of the image produced using [76]. The segmentation is then refined on the pixel level similar to [40], except the "band" around the boundary is made up of the pixels in the



(a) Color (b) Color adjacency (c) Temporal coherence (d) Shape (e) Point tracking

Figure 2.2: Visualization of the graph-cut terms for the frame with a cat from Figure 2.6. White indicates foreground likelihood, black background, and mid-gray neutral, except for the color adjacency, where white indicates an object boundary and black indicates no boundary.

oversegmentation regions which border the coarse-level cut. While the following terms are defined according to pixels, they can all be directly extended to oversegmented regions.

2.4.2 Object and background motion

Motion is an important cue in video segmentation. By considering the motion of the object, more precise local information may be used to segment it. By removing camera motion, better local information can be used for the background where it is static.

Many methods account for the camera motion by aligning the frames in a preprocessing step [41, 80]. However, since the foreground object will often exhibit different motion patterns than the background, aligning the background will not correctly align the foreground.

Since we know the segmentation of the previous frame, we can align the foreground and background separately. The background is aligned by locating good points to track [65], then computing and applying a homography.

While the foreground can be tracked in the same manner, problems can occur if the foreground does not have enough trackable points to generate a good homography due to large movements or little texture. To account for these cases, we use a novel method to roughly align the foreground.

We use an iterative closest point-style algorithm [6] to match pixels x_i in the selection M on the current frame I to pixels y_j in the next frame I_{next} with one affine transformation A.

The iteration alternates between (a) finding the best matches $\{(x_i, y_{m(i)})\}$ for a given A, and (b) finding the best A to align matches $\{(x_i, y_{m(i)})\}$. In (a), we match points in (xy position $\times$ RGB color) space so that points in M are matched to points in I_{next} that are similar in color and position after applying A. For each $x_i \in M$, we solve a nearest neighbor problem $y_{m(i)} = \arg\min_{y_j} ||(Ax_i, \gamma I(x_i)) - (y_j, \gamma I_{\text{next}}(y_j))||_2^2$ ([49]), where RGB values are in $[0, 1]$ and γ is the sum of the frame width and height. For (b), we solve $A = \arg\min_A \sum_{i=1}^{n} ||Ax_i - y_{m(i)}||_2^2$ ([62]).

We begin the iteration with the identity transformation, although perhaps a better starting value could be obtained by a prediction based on motion in previous frames. We run a maximum of 25 ICP iterations and declare that the algorithm has converged if the positions of all the transformed points do not move by more than 0.1 pixels. Our algorithm is robust enough that it can use a small subset of uniformly sampled points from M and still find a good affine transformation. We set the number of selection points to track to the maximum of 200 and 2% of the total points in M. An optional input of the largest allowable interframe motion allows us to reduce the set of potential matches in I_{next} to something less than the entire frame. If the number of points in M that we track is m and the number of potential matching locations in I_{next} is n, then the building the search kd-tree requires $O(n \log n)$ time and computing all the nearest neighbors in step (a) requires $O(m \log n)$ time for each ICP iteration. For SD-size videos, our novel foreground alignment procedure typically runs in tenths of a second with the specified size subset of M and a maximum interframe motion of 50 pixels.

The object and background motions are not included as a term in graph cut. Rather, they are used to spatially transform the locality information of the other cues. While this transformation does not completely capture non-rigid motion, it improves the locality of the foreground information and works well in practice.

2.4.3 Gradient

Image gradients are important for encouraging selection boundaries to fall on image edges. As in [40], we use color difference as a boundary term:

$$w_g(x_i, x_j) = \frac{1}{||C(x_i) - C(x_j)||^2 + 1}. \tag{2.4}$$

where $C(x_i) \in [0, 255]^3$ is the color at x_i. Gradient boundary terms are standard practice in graph-cut segmentation.

2.4.4 Color

A color-model region term encourages pixels to be labeled according to the color distribution of the model. Because most graph-cut algorithms [9, 40, 80] do not have access to a full segmentation of a frame, only the pixels under the user strokes are used to create the model. This limited sample does not always accurately represent the color properties of the image. These algorithms must also by necessity use a global color model, which does not differentiate colors located in different regions of the image. While [41] can use a local color model, it only does so over a small window if manually indicated by the user.

A contribution of LIVEcut is that it uses a *local* color model generated from the *entire* previous frame, which can distinguish between colors in different regions of the image. Such a color model is shown in Figure 2.2a, where the cat is likely foreground while the similarly-colored backpack and rope are not. The local color model is generated by creating a (l, u, v, x, y) vector p_i for each pixel x_i in the previous frame (where (l, u, v) is the color and (x, y) is the motion-adjusted location). The probability of the pixel being foreground is then computed by a 5D Fast Gauss Transform [83]. The probability is assigned to the cost term by

$$w_c(x_i, l) = P(p_i|\bar{l}) \tag{2.5}$$

where $P(p_i|\bar{l})$ is the normalized probability of the location and color of x_i given the label $\bar{l}$.

2.4.5 Color adjacency

Not only are the colors indicative of the objects, but the relationship of adjacent colors is as well. Certain color pairs may only exist within the object (background), while others only cross the object boundary. For example, the ballerina in Figure 2.6 contains a strong red-to-black edge in her clothing that only exists within her interior and never across her boundary. Ideally, a method should distinguish which transitions exist along the boundary and which do not.

While some methods have modeled the color profile of the object edge, such as [47] and its extension to video [31], they do not handle strong gradients within objects where a cut could occur. Cui et al. [19] modifies gradient strength based on color relationships but requires the color to be heavily quantized and does not specify exactly how the locality of edges is implemented.

We introduce a new color-adjacency model to weight the importance of image gradients. The model is computed using a Fast Gauss Transform [83], similar to the color model. Adjacent pixels are represented by an 8D vector $e_{ij} = (l_i, u_i, v_i, l_j, u_j, v_j, x, y)$ where (l_i, u_i, v_i) is the color of pixel x_i, (l_j, u_j, v_j) the color of the x_j, and (x, y) their motion-adjusted location. A model I is generated for all edges that are in the interior of either the foreground or background, and another model B is generated for all edges along the boundary. These probabilities are combined into a boundary reweighting factor by

$$w_a(x_i, x_j) = \left(1 + \left|\frac{P(e_{ij}|I) - P(e_{ij}|B)}{P(e_{ij}|I) + P(e_{ij}|B)}\right|\right)^{2\eta} \tag{2.6}$$

where $P(e_{ij}|l)$ is the probability of e_{ij} given the label l. η gives the sign of the numerator: 1 if $P(e_{ij}|I) \geq P(e_{ij}|B)$ and -1 otherwise. Equation 2.6 creates a scalar ranging from 0.25 if the model indicates a pure boundary ($P(e_{ij}|I)=0, P(e_{ij}|B)=1$) to 4 for a pure interior edge ($P(e_{ij}|I)=1, P(e_{ij}|B)=0$), with a factor $w_a=1$ for equal interior and boundary probabilities

($P(e_{ij}|I) = P(e_{ij}|B)$). Figure 2.2b shows the effect of the color adjacency model. The cat's outline is clearly highlighted as the desired boundary, while other edges are suppressed.

2.4.6 Spatiotemporal coherency

Videos usually exhibit a high amount of coherency between frames. Spatiotemporal-volume approaches [41, 80] implicitly capture this coherency through edges across frames. With our frame-by-frame approach, coherency between frames can be included without explicitly representing the labeled pixels from the previous frame. Rather, we assign a high region cost to label x_i as l if there is a nearby pixel (after motion adjustment) in the previous frame labeled $\bar{l}$ that has a similar color:

$$w_h(x_i, l) = \sum_{y_j \in N_{\bar{l}}(x_i)} \frac{1}{||C(x_i) - C(y_j)||^2 + 1} \tag{2.7}$$

where $N_{\bar{l}}(x_i)$ is the set of all neighbors of x_i from the previous frame that are labeled $\bar{l}$. Figure 2.2c shows the cost map for the spatiotemporal coherency where the cat is likely foreground since it overlaps with the previous frame. The blockiness is due to the oversegmentation regions.

2.4.7 Shape

When an object passes over a similarly colored background, no edge exists upon which to place the boundary. In these cases, the shape of the object is vital. Including a shape term in the features can handle such cases.

Recently there has been interest in including shape priors into graph cut [23, 35, 72, 77, 30]. The common approach is to align the shape to the image by user interaction and/or automated means, and then include a term in the cost function based on distance to the shape or a mismatch score.

In LIVEcut, because we have tracked the object motion forward, we already have an estimate of the motion-adjusted object shape Φ (where $\Phi(x_i) = 1$ if x_i is in the object mask and 0 otherwise) and its boundary Ω (where $\Omega = \partial\Phi$). We compute the distance from each pixel to the boundary after adjusting for object motion using

$$d_\Omega(x_i) = \min_{p \in \Omega}(||p - x_i||). \tag{2.8}$$

Our shape term is an extension to [77] but takes distance into account:

$$w_s(x_i, l) = |l - \Phi(x_i)| \min(d_\Omega(x_i)/M, 1) \tag{2.9}$$

where M is the maximum allowable distance (we use $M = 10$). If the estimated shape mask does not match the labeling of a pixel, this term penalizes the labeling based on the pixel's distance to the predicted shape boundary up to a threshold M. Using a small M, if the boundary is only off by a few pixels, it will have a minimal cost added. This cost function combined with the estimation of the object motion comprise a novel shape prior for graph cut. The resulting costs produced by the shape cue are shown in Figure 2.2d.

2.4.8 Point tracking

For most pixels in a typical video sequence, it is difficult to precisely determine the corresponding point in the next frame. However, easily-trackable points give nearly certain information about their labeling (see Figure 2.2e). While many algorithms make use of such points, video segmentation methods based on graph cut currently do not. We use [43, 65] to track these points and assign a penalty to labeling x_i as l if x_i is within a distance D (we use $D = 5$) of a tracked point that was labeled $\bar{l}$ in the previous frame:

$$w_p(x_i, l) = \begin{cases} 1 & \text{if } d_{\Theta_{\bar{l}}}(x_i) \leq D \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

where $\Theta_{\bar{l}}$ is the set of tracked points labeled $\bar{l}$. Any points that were not reliably tracked are removed from Θ_l. We also filter out any points too close to the object boundary (within 10 pixels), because points near the boundary may potentially spill over onto the other side.

2.5 Automatically Weighting Cues

While a variety of cues can be used for video segmentation, some features will perform more reliably than others given a specific sequence, frame, or even location within the frame. In order to best leverage the various cues, we evaluate and learn from their performance.

We automatically weight the region terms in graph cut on a local basis, as shown in Equation 2.2 by the α_u factors. In this manner, the most effective cues will have a stronger effect. Each α_u is a combination of an automatic scaling β_u based on the estimated effectiveness of that term locally (Section 2.5.1) and a weighting ρ_u that is learned through user corrections (Section 2.5.2):

$$\alpha_u(x_i) = \beta_u(p_i)\ \rho_u(x_i). \tag{2.11}$$

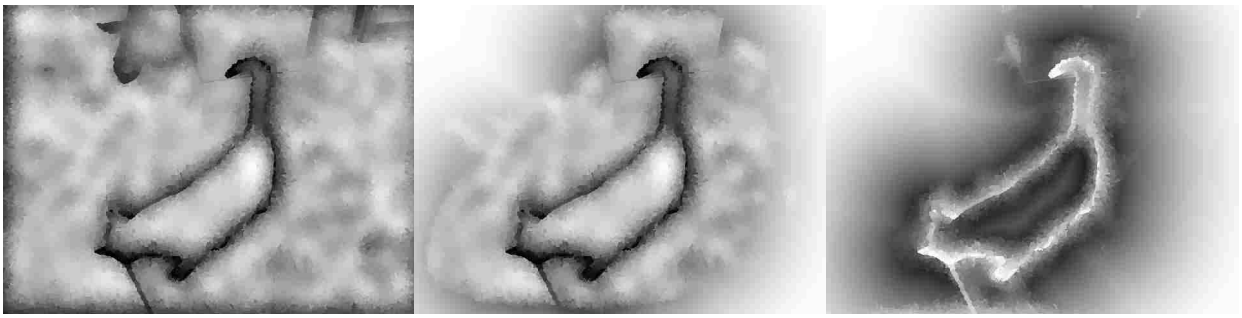
2.5.1 Setting estimated effectiveness

For each region term, LIVEcut assesses its own performance on the previous frame to estimate the accuracy of that feature for each pixel. This ensures that each feature is weighted strongly in the areas where it is most effective.

For the color term, the accuracy can be estimated by applying the model to the frame that generated it (i.e. the previous frame) by

$$\beta_c(x_i) = P^{prev}(p_i|\mathcal{L}_i^{prev}) \tag{2.12}$$

where the superscript *prev* indicates that the probability and label are from the previous frame and p_i is the (l, u, v, x, y) color and location vector for pixel x_i. Figure 2.3a shows



(a) Color (b) Coherency (c) Shape

Figure 2.3: Visualization of the estimated accuracy of the (a) color, (b) spatial coherency, and (c) shape terms. The grayscale value of a pixel indicates the local weight β for that pixel and cue.

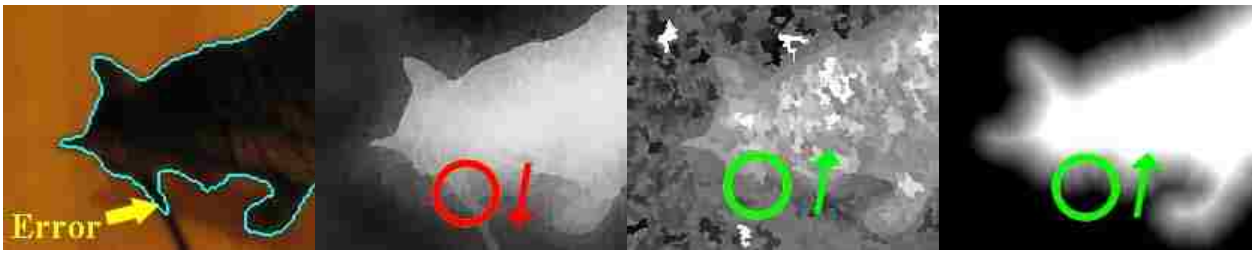
the estimated effectiveness for one frame of the cat sequence. Note that while the weighting is generally high (shown by brighter pixels), it is lower near where the cat crosses the rope due to the overlapping color models.

The coherency term in Equation 2.7 returns a large value when a pixel is similar in color to a neighboring pixel of the opposite label in the previous frame. This works well except when near object boundaries where the foreground and background colors are similar, because the costs for each label are then similar. This can be detected by a low probability in the color model near the boundary. We weight the coherency term accordingly:

$$\beta_h(x_i) = \max(P^{prev}(p_i|\mathcal{L}_i^{prev}), \min(d_\Omega^2(x_i)/D^2, 1)) \tag{2.13}$$

where D is a distance threshold (we use 0.25(image width + height)). This equation includes both a color term and a distance term. Near the boundary, the distance term is small, so the color term dominates, and the weight is high if the foreground and background colors are dissimilar. Far from the boundary, the distance term dominates. This is illustrated in Figure 2.3b, where near the object the weighting looks similar to that of the color model, while away from the object it resembles a distance map.

The shape term is most important for localizing boundaries where the similarity of foreground and background colors weakens the effectiveness of the color, spatiotemporal,



(a) Error (b) Color (c) Coherency (d) Shape

Figure 2.4: (a) An error occurred in the propagated segmentation. Since the (b) color cue was incorrect, its weight is decreased. The (c) coherency and (d) shape cues were correct, so they are increased. The grayscale value in (b-d) visualizes the label the cue suggests, with white indicating foreground and black background.

gradient, and color adjacency terms. It is also effective far from the boundary where the labeling is more certain. Based on these ideas, we weight the shape term using

$$\beta_s(x_i) = \max(1 - P^{prev}(p_i|\mathcal{L}_i^{prev}), \min(d_\Omega^2(x_i)/D^2, 1)). \tag{2.14}$$

Note the similarity to $\beta_h(x_i)$ in Equation 2.13, except that the color probability has been subtracted from 1. This can be seen in Figure 2.3c, where the shape term looks like the coherency term except that it is inverted near the object. The shape prior will thereby be more heavily weighted in areas where color does *not* effectively identify the boundary, such as the cat and rope overlap in Figure 2.3.

For the point-tracking term, we have already removed points that were not reliably tracked, so we have high confidence in the remaining points. We therefore weight all points equally: $\beta_p(x_i) = 1$.

2.5.2 Learning from user corrections

While the automatic weighting usually works well, it may be incorrect at times and require further user correction. LIVEcut handles this automatically by learning from the corrections. The user corrects any mistakes by marking them with strokes before proceeding to the next frame. Since each region term gives a value in favor of the foreground $\mathcal{F}$ and the background

$\mathcal{B}$, each suggests a label for each pixel. More precisely, if $w_u(x_i, \mathcal{F}) - w_u(x_i, \mathcal{B}) > 0$, then the term w_u would label x_i as background on its own, and vice versa. By comparing the initial propagated selection to the selection after corrections, we can determine which features were correct at each pixel and use that to weight their future performance. In Figure 2.4, since the color weight for foreground $w_c(x_i, \mathcal{F})$ was greater than the background weight $w_c(x_i, \mathcal{B})$, the future weight of those color terms are weakened. The coherency and shape terms suggested the correct labeling and are strengthened.

We initialize ρ_u to a constant value for all x_i. Let S_u^i be the initial propagated segmentation suggested by term w_u alone, and let S^f be the final segmentation after the user has corrected any mistakes. If $S_u^i(x_i) \neq S^f(x_i)$, w_u suggested an incorrect labeling for x_i and its weight should be discounted by ρ_u in the next frame,

$$\rho_u^{next}(x_i) = \begin{cases} \rho_u(x_i) + \delta_0 & \text{if } S_u^i(x_i) = S^f(x_i) \\ \rho_u(x_i) - \delta_1 & \text{if } S_u^i(x_i) \neq S^f(x_i) \end{cases} \tag{2.15}$$

where all segmentations S are from the current frame. δ_0 and δ_1 are constant increments for ρ_u. We use $\delta_0 = 0.4$ and $\delta_1 = 0.8$, and ρ is initialized to 1.

2.6 Results

For an interactive segmentation system, the real measure of success is the amount of user time required to perform a selection. Accordingly, we report the time required to select objects in several sequences. The segmentations can be judged qualitatively by the examples shown in Figure 2.6. In order to better evaluate the accuracy of LIVEcut, we also compare it to automatic segmentation techniques, despite the disadvantage this gives to an algorithm designed for interactive use.

Video	Size	Graph Cut Time	User Time
Bass Guitar	960×540×72	0.055 / 3.53 sec	38 min
Cat	640×480×56	0.038 / 1.88 sec	5 min
Flamingo	960×540×76	0.055 / 2.82 sec	30 min
Footballer	720×576×19	0.053 / 1.99 sec	5 min
Lemurs	960×540×86	0.047 / 3.11 sec	36 min

Table 2.1: Timing results from several sequences. The graph-cut time first gives the time to process an interaction on one frame, and then the time to propagate information to the next frame. "Footballer" is courtesy of Artbeats (www.artbeats.com).

		Other Techniques					LIVEcut	
Video	Size	Method	Pre-process	Graph Cut Time	User Time	Post-process	Graph Cut Time	User Time
amira	640×480×35(80)*	[80]+[1]	12 min	5 sec	15 min	35 min	0.054 / 1.62 sec	7 min
ballerina	640×480×150	[80]	25 min	11.5 sec	140 min	30 min	0.051 / 1.76 sec	74 min
elephant	720×480×100	[80]	20 min	9.1 sec	40 min	30 min	0.036 / 2.39 sec	38 min
manincap	640×480×150	[80]	30 min	16.5 sec	20 min	35 min	0.034 / 1.86 sec	23 min
stairs	640×480×63(100)*	[80]	20 min	8.5 sec	20 min	30 min	0.028 / 1.71 sec	13 min

Table 2.2: Comparison of LIVEcut to [80]. The graph-cut time for LIVEcut lists first the time to process an interaction on one frame, and then the time to propagate the selection to the next frame. The '*' indicates that the video we obtained differed in length to that reported in [80] (shown in parentheses). The postprocess time for [80] consists of pixel-level refinement, but also includes matting, which is not reported for the our method. LIVEcut does not need any pre-processing time.

2.6.1 Timing and qualitative results

Table 2.1 gives timing results over several challenging video sequences. The "footballer" sequence exhibits large motions, a drastically changing object shape, and a partial occlusion from another moving object. "Bass guitar" and "lemurs" both contain overlapping color models, boundaries where there is no gradient information, and motion blur. While much of the body of the "flamingo" is easy to segment, the legs are narrow, exhibit large movements, are often heavily blurred, and have a similar color to the background. Using LIVEcut, a user is able to segment the objects without excessive interaction. The selection from several frames of these sequences can be seen in Figure 2.6. We apply the robust matter [81] to our output to account for mixed pixels on boundaries.

We compare LIVEcut to [80] using videos from this paper in Table 2.2. The user time to acquire binary segmentation results similar in quality to these techniques is comparable or less in these examples. The time the user must wait between each interaction for the selection to update is also less, providing a better interactive experience. Our algorithm also does not

Sequence	41	43	50	51	54
LIVEcut Error %	2.30	5.93	1.07	1.18	0.45
[84] Error %	0.80	0.02	1.31	1.06	0.33
Sequence	56	58	60	IU	JM
LIVEcut Error %	2.47	0.24	14.96	2.96	31.37
[84] Error %	0.93	0.79	6.33	2.56	0.27

Table 2.3: Comparison of [84] to LIVEcut using automatic segmentation (i.e. *without* allowing user corrections).

need the large preprocessing time that [80] requires. We were able to segment "amira" with LIVEcut, while [80] required the help of [1] to do so. We also were able to segment the "ballerina" as one object, while [80] required one pass for the feet and another for the body. Finally, our user interaction is simpler, requiring only drawing strokes on individual frames and allowing sequential processing of the video, while [80] also requires rotating and slicing through a spatiotemporal volume.

Note that [80] requires postprocessing time to refine the segmentation to the pixel level, but also includes time to compute the matte, which is excluded in the timing results of the other methods. Also note that for two sequences, indicated by a '*', the video sequences we obtained were of shorter length than those reported in [80]. The original lengths are reported in parentheses.

2.6.2 Accuracy and stability

For interactive segmentation systems, accuracy is difficult to measure since a user can always achieve perfect accuracy given enough time. To demonstrate accuracy, we perform automatic segmentations and compare to the unsupervised method from [84] on their database. In doing so, LIVEcut faces a large disadvantage. LIVEcut was designed to assume that the previous frame was correctly segmented by the user, and proceeds under that assumption. Furthermore, LIVEcut receives no user training, while [84] is trained on similar data. While this test neutralizes many of the strengths of LIVEcut, it allows us to show the algorithm's accuracy and stability.

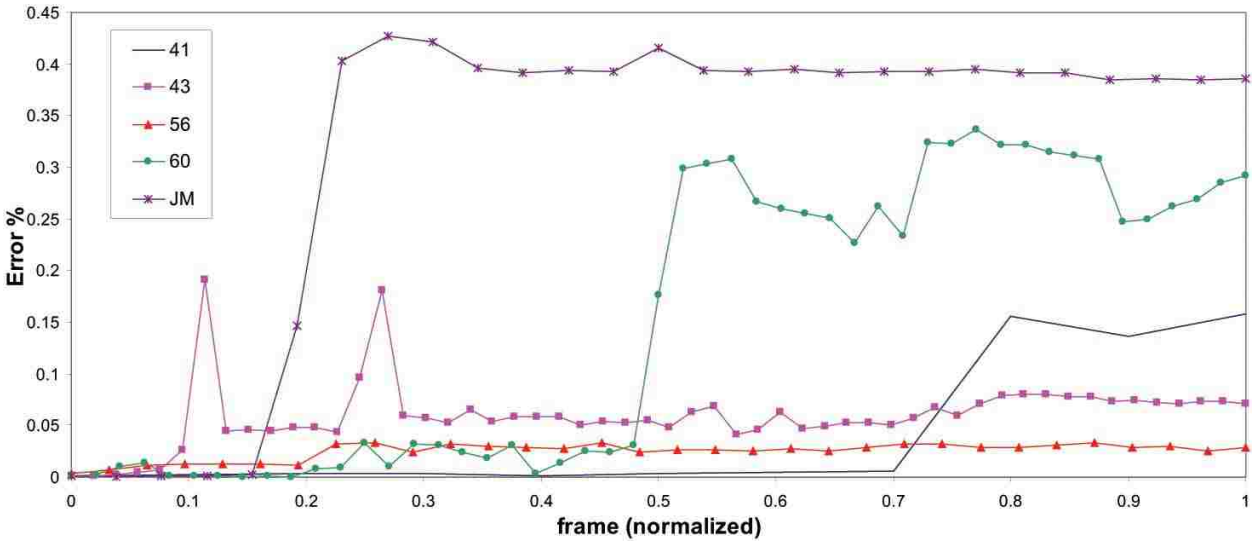


Figure 2.5: Accuracy of several sequences from Table 2.3.

For this test, we segmented the first frame of ten sequences, each of size 320×240 with an average length of over 350 frames. We then computed the segmentation *without* additional user interaction and compare to the results from [84] in Table 2.3. For several of the videos (50, 51, 54, 58, IU), we have comparable or better results. For the others, the accuracy over time is shown in Figure 2.5. Our segmentation error in each case was very low until an abrupt increase due to a change in the scene. For three of the cases, the error is low until the subject moves his hand in front of his body. In these cases, LIVEcut assumes that the hand is an occluding object that it should not segment and does not recover the entire object once the hand leaves. In the other cases, a rapid motion confuses our algorithm. Note that in each case, the error is quite stable after the initial mistake because LIVEcut accurately tracks what it assumes is the new state of the object.

To better show the accuracy and stability on these sequences, we resegmented the video allowing corrections only on or near the frames where large errors occur. Table 2.4 shows that the accuracy is now similar to or less than [84] while allowing very few corrections. While LIVEcut can achieve similar results to unsupervised methods with little or no

Sequence	41	43	56	60	JM
LIVEcut Error %	0.55	1.41	1.37	3.17	2.26
# frames corrected	1	5	1	2	7

Table 2.4: Accuracy of LIVEcut on sequences from Table 2.3 after corrections on the number of frames shown.

corrections, these methods could not produce the high quality results from LIVEcut shown in Figure 2.6.

2.7 Conclusion

We have presented a new method for interactively segmenting video sequences by propagating multiple cues from one frame to another. These cues are automatically weighted according to their predicted importance on the specific video sequence being segmented, and are further weighted based on learning from user corrections. Many of the cues also include novel improvements in the context of video segmentation using graph cut.

While propagating multiple weighted cues is effective in segmenting video, further improvements can be made. LIVEcut only uses cues from the previous frame together with the accumulated learning. However, more global information about the entire video sequence may assist the segmentation. Improved learning techniques may better weight the graph-cut terms.

Figure 2.6: Several examples of object selections using LIVEcut.

Chapter 3

Color Adjacency Modeling

A portion of Chapter 2 briefly described a novel contribution of our work, a color adjacency model that allows the segmentation algorithm to focus on edges that are similar to the desired object boundary and ignore edges that are internal to the foreground or the background. This chapter further develops that idea, gives several improvements, and validates its effectiveness. This chapter is presented as it will appear in the *International Conference on Pattern Recognition* (ICPR), August 2010 under the title *Color Adjacency Modeling for Improved Image and Video Segmentation* [51].

3.1 Abstract

Color models are often used for representing object appearance for foreground segmentation applications. The relationships between colors can be just as useful for object selection. In this paper, we present a method of modeling color adjacency relationships. By using color adjacency models, the importance of an edge in a given application can be determined and scaled accordingly. We apply our model to foreground segmentation of similar images and video. We show that given one previously-segmented image, we can greatly reduce the error when automatically segmenting other images by using our color adjacency model to weight the likelihood that an edge is part of the desired object boundary.

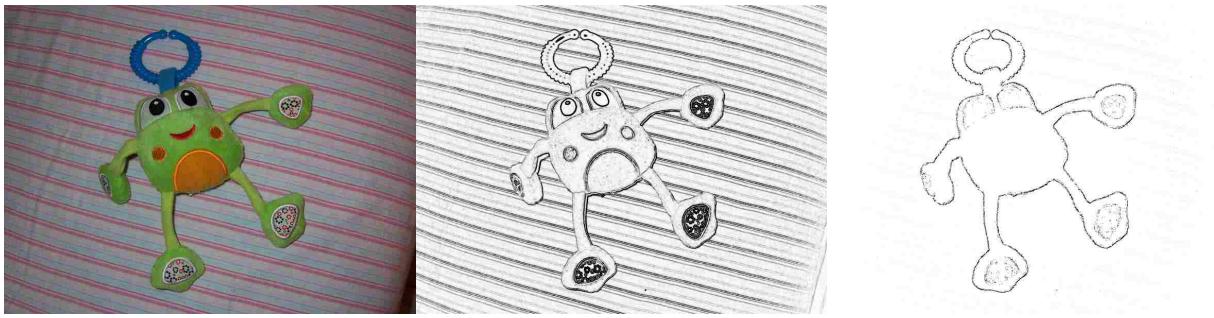
3.2 Introduction

Object selection is an important task for many image and video editing applications. A common component in many foreground segmentation algorithms is color, which is used to generate descriptions of the object of interest and the background. There are many ways to implement color models, such as histograms [47], Gaussian mixture models [60], clustering [40], and so on.

While often overlooked, the relationships between colors in an image can also be used for segmentation when boundary characteristics are known. While object boundaries are usually placed on edges in an image, in most images many edges do not correspond to the desired boundary. Such edges produce attractive locations for a segmentation algorithm to place the object boundary incorrectly. A color adjacency model can be used to help distinguish between desirable and undesirable edges for segmentation boundaries.

For example, assume we want to select the frog toy in Fig. 3.1(a). There are many gradients that do not belong to the object boundary, such as the lines in the background pattern (Fig. 3.1(b)). Color transitions such as these lines, the edge between the frog's head and the blue attachment, or the frog's green body and orange underbelly, create strong edges that may be a desirable object boundary for a given segmentation algorithm. However, since these color transitions occur only within the object or background and never across the object boundary, these edges could be ignored. By modeling not only the colors themselves, but also the color transitions, we may be able to identify such edges and not consider them as potential object boundaries (Fig. 3.1(c)).

We introduce a method for modeling color adjacency relationships in order to improve the performance of object selection tasks, including the following contributions. First, we present our color adjacency model. Second, we provide several enhancements to the model, specifically allowing for global or local color information and providing means of decontaminating mixed pixels at object boundaries. Finally, we incorporate our model into graph-cut segmentation algorithms and demonstrate improved performance in video and similar-image



(a) Image (b) Edges (c) Reweighted

Figure 3.1: An image (a) may contain edges (b) that are not part of the desired object boundary. By training a color adjacency model on a similar image, edges are reweighted (c).

segmentation tasks. By similar images, we mean images of the same background scene and foreground object but with possible differences due to changes such as the camera position and angle, pose of the object, or small variations in the background scene. This can occur in multiple shots of the same scene or in successive frames of video.

3.3 Related Work

While many segmentation algorithms use color models [40, 60], especially when propagating segmentation information to similar images or successive video frames, few have incorporated color adjacency relationships. Recently, [19] used a form of color adjacency modeling to help transfer information from a segmented image to similar images by modifying local edge weights. However, the probability of a transition is not modeled directly but instead derived from foreground/background color probabilities. If two colors both appear in an object, the probability of a transition between those two colors within the object is erroneously considered to be high when that specific transition is not present in the object, such as illustrated in Fig. 3.2. This problem also affects methods that use foreground/background probability gradients (e.g., [4]).

Intelligent Scissors [47] describes an "on-the-fly" training method that reweights edges. For a given partial segmentation boundary, the gradient magnitude of the most recently generated boundary pixels are stored in a histogram to represent the desired edge

Figure 3.2: Importance of color adjacencies. Blue-green transitions occur only on the boundary of the man, even though both colors occur in both foreground and background.

characteristics. This is used to reweight the cost for the lowest-cost path generation. While this training approach works well, it is limited in that it only distinguishes between gradient magnitudes and not color relationships.

Others have incorporated color adjacency relationships as histograms (co-occurrence matrices) for use in image matching and retrieval and video scene segmentation [36, 29]. Unfortunately, such histograms can be enormous, requiring the number of colors to be constrained. In [36], the image is quantized and only dominant color transitions are modeled. In [29], the image is oversegmented, and the number of colors is limited to 16. Such color reduction limits the ability to accurately model color transitions.

Some approaches model the geometry of edges [13, 21] for segmentation or other applications. Region adjacencies have also been modeled [15, 70] by oversegmenting the image into regions of similar color and using a region adjacency graph. These approaches differ significantly from our goal of modeling color transitions.

An early version of our model was used as a component of a video segmentation system in [50]. This paper more fully describes the color adjacency model and extends it by allowing color information to be global or local and by using color decontamination. The model is applied to a new problem set, that of selecting objects in similar images. We also validate the effectiveness of the model on both similar image and video segmentation tasks.

3.4 Methods

3.4.1 General Color Adjacency Modeling

To model the co-occurrence of adjacent colors in images, we represent each color adjacency combination as an adjacency tuple t combining the color components of the adjacent colors. For example, for two neighboring pixels p and q, a tuple $t_{pq} = (r_p, g_p, b_p, r_q, g_q, b_q)$ may be created, where r_p, g_p, b_p are the red, green, and blue channels of the color at pixel p respectively. Other color representations such as LUV may be used.

To model the color adjacencies, we estimate likelihoods for data points t_{pq} using a kernel density estimation computed by the Fast Gauss Transform [83]. This method uses a sparse representation and does not require a large amount of storage or coarse quantization.

3.4.2 Global vs. Local Information

This formulation gives a model that is independent of the location of the pixels in the image, which may be desirable for applications such as transferring knowledge from one image to another when object placements are not aligned. However, locality may be desirable in certain applications such as video segmentation where the amount of motion is small between frames.

To include locality, spatial position can simply be added to the adjacency tuple. For example, given a two-dimensional image with an RGB color representation, $t_{pq} = (r_p, g_p, b_p, r_q, g_q, b_q, x, y)$, where x and y are the location of the adjacency relationship.

To adjust the locality of the model, the relative range of the location components to the color components of t_{pq} may be altered. We normalize the color values to the range $[0, 1]$, then scale the image width or height (whichever is greater) to the range $[0, \omega]$, with the other dimension scaled accordingly. Smaller values of the parameter ω cause the model to become less sensitive to spatial locality, while larger values of ω cause the significance of the locality to increase.

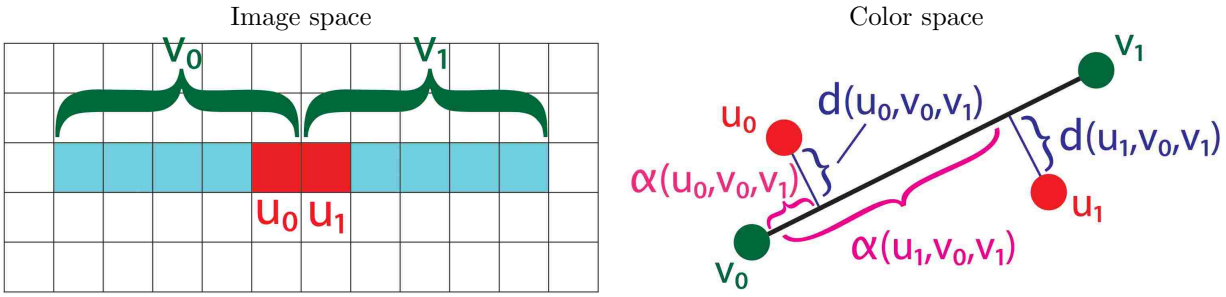
3.4.3 Color Decontamination

One difficulty in computing color adjacencies for adjacent pixels is that edges are rarely hard boundaries. Because of partial-pixel effects, focus or motion blur, or semi-transparent edges, object boundaries often extend over many pixels. While in some cases we may want to model these mixed pixels, in other cases we would like to use only the colors from the actual objects and not the mixed colors.

To account for this, we apply a color decontamination step by assuming that the colors at adjacent pixels are linear combinations of two nearby colors. This is fairly simple to do around the object boundary in the image on which we train our model since we know where the boundary is, and any standard color decontamination algorithm would do. We use a fairly simple method of moving in the direction $(p - q)$ of the color adjacency t_{pq} until intersecting a foreground pixel in one direction and a background pixel in the opposite direction.

The decontamination problem becomes much more difficult for the adjacencies that are entirely within the foreground or the background in the training image, and for all pixels in the test images. In these cases it is not known whether the adjacent pixels lie along an object edge or not, but not decontaminating these pixels can lead to problems in the adjacency model. For example, consider a simple case of an edge from white to black, which is blurred somewhat to create midrange grayscale values along the edge. If we decontaminate these pixels while training our model, we will model a black-to-white transition. If we do not decontaminate when evaluating new pixels, we will see midrange grayscale transitions that are not represented in the trained model. In order for the decontamination to work best, we must assume that color adjacencies in the test images are along the boundary and attempt to decontaminate them.

In order to decontaminate these unknown transitions, we choose candidate decontaminated colors by sampling pixels near the adjacent pixels in the same row or column as illustrated in Fig. 3.3(a). It is assumed that one of the decontaminated colors belongs to one



(a) Candidates (b) Model Evaluation

Figure 3.3: (a) Candidate decontaminated pixels with colors v_0 and v_1 are chosen from the bracketed pixels. (b) v_0 and v_1 should align well with the adjacent pixel colors u_0 and u_1 in color space.

of the pixels in one group (indicated by the green brackets), and the other color in the other group. The colors u_0 and u_1 of the adjacent pixels are thereby assumed to be a linear blend of the two (candidate) decontaminated colors v_0 and v_1. The blending coefficient α at the pixel with color u_0 is given by

$$\alpha(u_0, v_0, v_1) = \frac{(u_0 - v_0) \cdot (v_1 - v_0)}{||v_1 - v_0||^2} \tag{3.1}$$

The distance d from u_0 to the closest point on the line between v_0 and v_1 can then be computed by

$$d(u_0, v_0, v_1) = \frac{||(u_0 - v_0) \times (u_0 - v_1)||}{||v_1 - v_0||} \tag{3.2}$$

If the closest point is outside the line segment between v_0 and v_1, we set $d(u_0, v_0, v_1) = \infty$. The values of α and d are shown graphically in Fig. 3.3(b) as they relate to the colors u_0 and u_1 at adjacent pixels and the candidate decontaminated colors v_0 and v_1.

Two candidates v_0 and v_1 are chosen as the decontaminated colors for adjacent pixels with colors u_0 and u_1 so that they minimize the cost function

$$\Gamma(u_0, u_1, v_0, v_1) = \frac{d(u_0, v_0, v_1) + d(u_1, v_0, v_1) + 1}{||v_0 - v_1||^2 + 1} \tag{3.3}$$

on condition that the difference between u_0 and u_1 is sufficiently high (we use $||u_0 - u_1|| > 10$ for colors $u \in [0, 255]$) and that either $0.25 \leq \alpha(u_0, v_0, v_1) \leq 0.75$ or $0.25 \leq \alpha(u_1, v_0, v_1) \leq 0.75$. The cost function in Eq. 3.3 is designed to be small when the distance from the adjacent pixel colors to the line between the candidate decontaminated colors is low, ensuring that the adjacent pixel colors are well represented as linear blends of the candidate colors. It is also small when the distance between the candidate colors is large, which helps ensure that the decontaminated colors are not linear blends themselves.

3.4.4 Use in Object Selection

To use our color adjacency model to assist object selection, we modulate the gradient magnitudes by multipying them by

$$s(x) = \begin{cases} (1 + \beta[L_E(x) - L_I(x)])^2 & L_E(x) > L_I(x) \\ (1 + \beta[L_I(x) - L_E(x)])^{-2} & \text{otherwise} \end{cases} \tag{3.4}$$

where $\beta > 0$ adjusts the effect of the color adjacency model and $L_E(x)$ is the posterior probability (assuming equal priors) that the pixel transition x belongs to the object boundary (denoted by E) as given by

$$L_E(x) = \frac{P(x|E)}{P(x|E) + P(x|I)} \tag{3.5}$$

where I denotes the interior of the background or object, and P is the probability as computed by our color adjacency model. The effect of Eq. 3.4 is that when $L_E(x) = L_I(x)$, the edge strength is left unchanged. As $L_E(x)$ increases, the reweighting factor approaches $(1 + \beta)^2$, and the local edge weight increases. As $L_E(x)$ decreases, the reweighting factor approaches $(1 + \beta)^{-2}$ which decreases the edge weight.

Fig. 3.4 visualizes the effect that the probability difference $L_E(x) - L_I(x)$ has on an images. Red indicates that the probability of the transition being a desired edge is large, and white indicates that the probability of the transition being in the interior is large. More

precisely, the range $[0, 1]$ for the difference maps to shades of red, and the range $[0, -1]$ maps to shades of grey (white). Black indicates that both possibilities are equally likely. Notice how many of the edges in the image that do not correspond to the object boundary but would be attractive places to place a segmentation boundary based on the strength of the gradient have been suppressed. For example, the diagonal lines in the frog toy image have been largely eliminated. Conversely, many edges along the desired boundary have been strengthened, such as along the boundary of the cat.

3.5 Results

We show the effectiveness of our method by comparing segmentation results with and without the color adjacency model on sets of similar images and on video frames. We transfer the selection from a manually-segmented image to similar images or subsequent video frames within a common graph-cut framework [40]. We compare this selection to one created with [40] but with edge weights modified by modulating the gradient magnitudes according to Eq. 3.4 with and without decontamination. Note that we use the graph-cut framework for our segmentation results here since it is currently the most popular and arguably best interactive segmentation framework. While we use [40], Eq. 3.4 should apply to any graph-cut or similar method. The ground truth for our data was generated manually using the Quick Selection tool in Adobe Photoshop, and the number of mislabeled pixels is reported as the error. Any pixels within two pixels of the boundary are not included in the error measurement.

Table 3.1 shows the results for transferring a selection from one image to five related images, and from one video frame to the next five frames. For similar images, because the size, position, and orientation of the objects may differ greatly between images, we de-emphasize the local component in our model ($\omega = 0.05$). For each example, the error is reduced greatly when including the color adjacency model. For the video examples, we

Images	Standard	With Adjacency	Decontaminated Adjacency	% Reduced
Cat	7.5	2.4	0.5	93%
Orca	29.7	17.3	16.1	46%
Frog	10.4	2.8	2.4	77%
Video				
43	6.2	0.9	0.5	91%
54	3.4	0.4	0.4	88%
Cat	9.7	0.4	0.3	97%
Football	139.2	87.2	87.2	37%
Man	58.7	30.1	29.8	49%
Ballet	16.2	1.2	1.0	94%

Table 3.1: Error (in 1000's of mislabeled pixels) averaged over five similar images or videos frames. The selection is performed using [40] without and with the adjacency model and decontamination.

enforce locality more by using $\omega = 1.5$. Segmenting video using our method clearly improves performance.

Note that while for this experiment we use the same model for each frame in a video sequence, in long videos where the object or background change dramatically best performance is achieved by recomputing the color adjacency model at each frame. This is how the color adjacency model (computed in subsecond time) was applied in [50], which gives segmentation results using our model for sequences of more than 350 frames in length.

Results with and without color adjacency modeling are shown in Fig. 3.5. When using global color models alone, there are many disjoint pieces and holes in the result. Reweighting the edges based on color adjacency greatly improves the results of the segmentation. Fig. 3.6 shows the results for all five of the frog images.

3.6 Conclusion

We have presented a method of improving object selection in cases where the boundary information is known, such as in video or sets of similar images, by the application of a color adjacency relationship model. By increasing the strength of edges similar to the desired

object boundary and weakening other edges using a color adjacency model, segmentation algorithms can more easily isolate the correct object boundary.

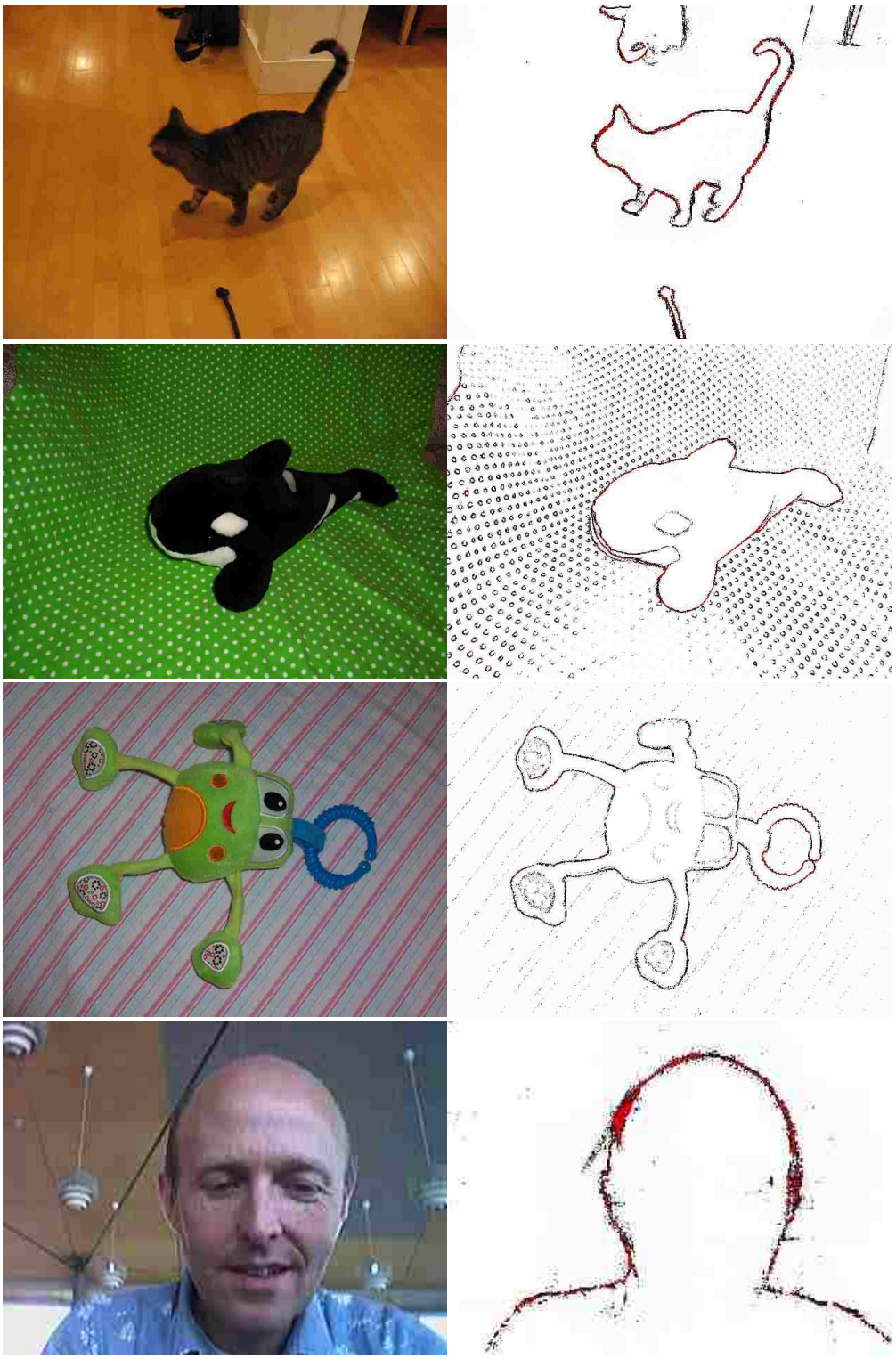


Image Adjacency

Figure 3.4: Edge reweighting. Red indicates likely desired edges, white likely interior edges, and black equally likely edges.

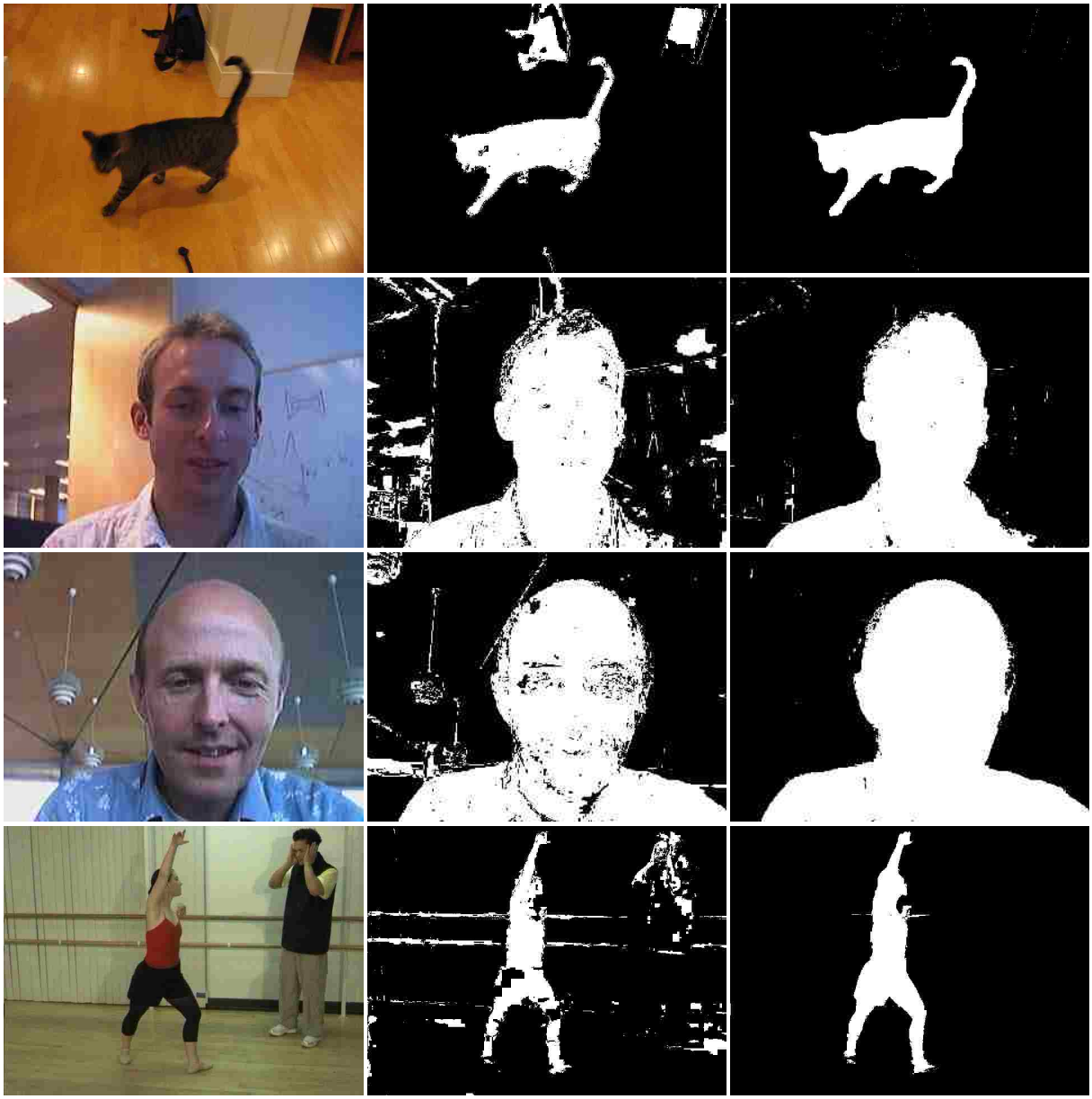


Image Standard With Adjacency

Figure 3.5: Example segmentation results.

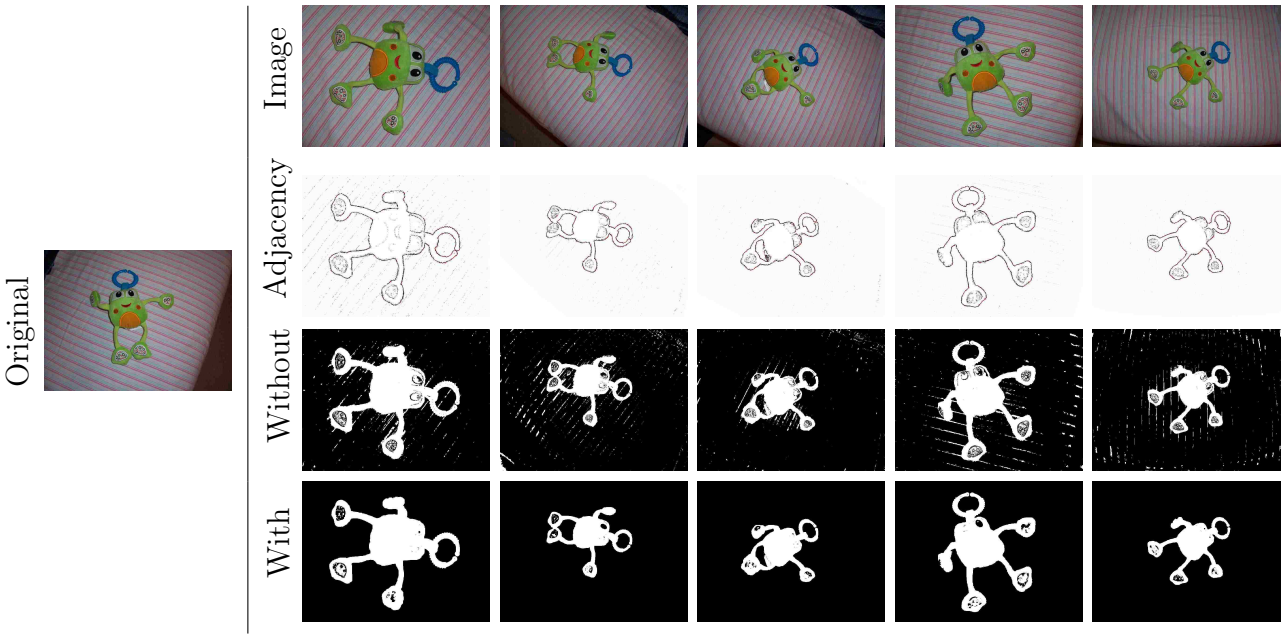


Figure 3.6: Results from the frog example. The original image (left) was segmented, and a color adjacency model is applied to the five similar images (row 1). The adjacency probability as shown in Fig. 3.4 (row 2) and selection without (row 3) and with (row 4) the adjacency model are shown.

Chapter 4

Geodesic Graph Cut

Chapter 2 presents a method of dynamically applying multiple cues to different regions of a video in order to improve segmentation. This chapter explores this basic idea applied to single-image segmentation. In this context, we do not have prior knowledge about the object of interest, like a previously-segmented frame from a video sequence, to use to weight cooperating cues. Instead, we use previous knowledge of the failure cases from two different algorithms to combine them for better results. This chapter was originally published in the *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR), June 2010 under the title *Geodesic Graph Cut for Interactive Image Segmentation* [52].

4.1 Abstract

Many types of interactive segmentation algorithms have been developed, each with their own strengths and drawbacks. Segmentation methods based on graph cut are among the most popular but suffer from an inherent bias toward small boundaries, causing them to "short-cut" across objects and cut off object appendages. Distance-transform based algorithms are also commonly used and excel at segmenting along narrow appendages without short-cutting. However, unlike graph cut, these methods do not explicitly encourage the segmentation boundaries to fall on image edges, and are limited by the accuracy of their distance function to cases with relatively simple color distributions. This paper introduces a novel method of incorporating geodesic distance transform information into the graph-cut segmentation model. This hybrid approach reduces the drawbacks of each algorithm by relying on the

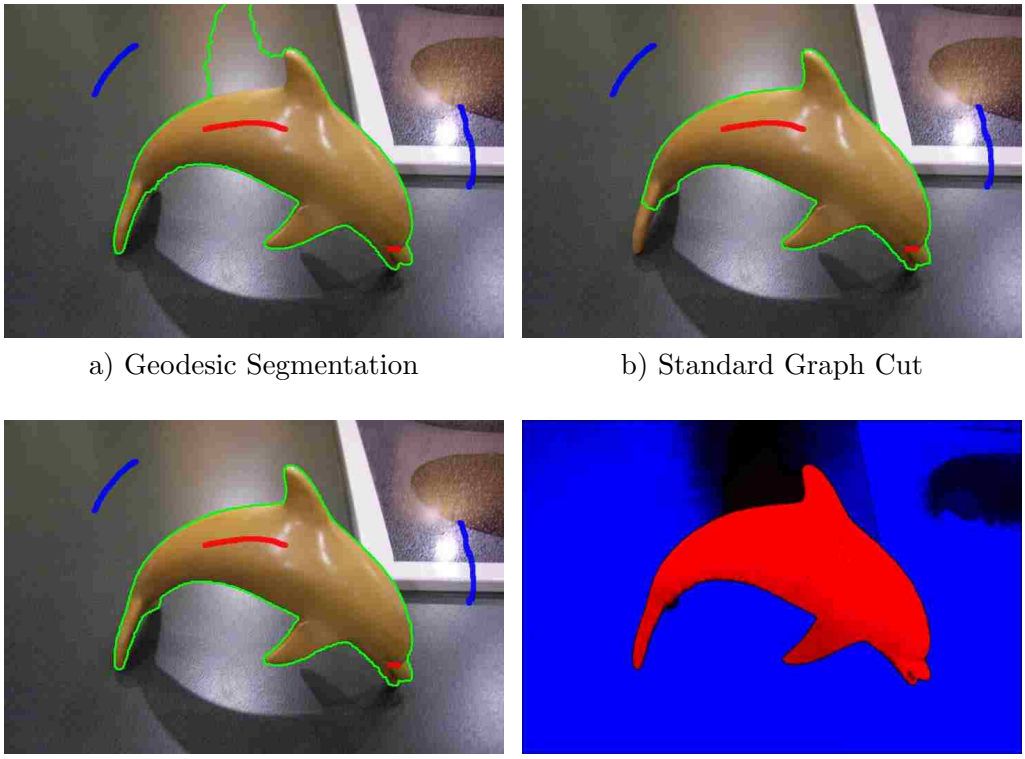
strengths of each. The relative weighting of traditional graph-cut information verses geodesic information is automatically determined using image data, allowing the algorithm to adapt on a per-image basis. Various examples show the improvement of our method over graph cut or geodesics alone.

4.2 Introduction

Segmentation is one of the most fundamental and well-studied problems in computer vision. Because of the inherent difficulty and ambiguity, many methods use interactive segmentation, which allows a user to supply information regarding the object of interest. Many forms of interaction have been used, ranging from loosely tracing the desired boundary (e.g., [24, 32, 47, 7, 79]) to loosely marking parts of the desired object and/or background (e.g., [4, 9, 10, 25, 54, 55, 66, 40]) to loosely placing a bounding box around the desired object (e.g., [60, 37]). In all forms, the goal is to allow the user to accurately select objects of interest with minimal effort.

We focus here on approaches where the user marks or "scribbles" on parts of the desired foreground and background regions to seed the segmentation (Figure 4.1). Such approaches are popular because they generally require less precise input from the user, allowing them to loosely mark broader interior regions instead of more finely tracing near object boundaries, though each approach can sometimes be advantageous. Allowing the user to draw a bounding box [60] is simpler in many cases, though may not provide sufficient control in all cases, in which scribble-based corrections are often employed to refine the results.

Many methods for seeded segmentation expand outward from the seeds to selectively fill the desired region, either explicitly [88, 55, 54, 4] or conceptually [25]. Because these approaches work from the interior of the selected object outwards and do not explicitly consider the object boundary, they are particularly useful for selecting objects with complex boundaries such as those with long, thin parts. However, because these expansions are



a) Geodesic Segmentation b) Standard Graph Cut

c) Geodesic Graph Cut d) Geodesic Confidence

Figure 4.1: Geodesic graph cut. Without edge information, geodesic segmentation alone can fail in areas where the foreground/background colors are not distinct (a). Graph cut segmentation does a better job of aligning with edges but is susceptible to short-cutting (b). Graph-cut optimization with an automatically tuned geodesic-distance region term leverages the strengths of the approaches and more accurately selects the object (c). In addition to global tuning, spatially adaptive weighting (d) is used to prevent boundary placement in clearly foreground (red) or background (blue) regions, shifting greater control to the edge-finding component when uncertain (black).

monotonic, these approaches suffer from a bias that favors shorter paths back to the seeds. As a result, they can be sensitive to seed placement, as illustrated for geodesic segmentation in Figure 4.2. Because they lack an explicit edge component, these methods may also fail to accurately localize object boundaries. The stronger the image edges are, the more likely these methods are to make these transitions here, but this is not guaranteed, as illustrated for geodesic segmentation in Figure 4.3.

The most popular approach to seeded segmentation is currently the graph-cut approach of [9], with numerous proposed variations (e.g., [40]). This method combines explicit

edge-finding and region-modeling components, formulated as a weighted combination and optimized by framing the problem as a minimum cut in a weighted graph that partitions foreground seeds from background seeds. (See Section 4.4 for a more detailed description.) In its original form (and most subsequent forms) the region term uses foreground/background color models inferred from the respective seed pixels. This region/edge combination can be an effective method in many cases, frequently improving on edge- or region-based segmentation methods alone.

However, because the boundary term in graph-cut methods consists of a summation over the boundary of the segmented regions, there is an inherent and well-known bias towards shorter paths, sometimes known as the *length* or *shrinking bias*. (This length bias predates graph-cut methods and is present in earlier least-cost path approaches [47].) This can be especially noticeable when these methods short-cut across the interior of an object to avoid segmenting an appendage as illustrated in Figure 4.4. This is offset somewhat by the region term, which tries to penalize shortcuts through areas where the labeling is clear from the coloring. However, this is not without tradeoffs—overweighting the region term to compensate for the boundary term's shrinking bias can result in discontiguous objects with coloring similar to the user's respective scribbles being incorrectly selected (as also illustrated in Figure 4.4). This leads to a delicate balance between the weighting of the two terms and the resulting strengths and limitations of each.

Most approaches for otherwise avoiding the shrinking bias in graph-cut and similar approaches involve variations on normalizing the cost of the cut by the size of the resulting object(s). This may be done for grey-level images for which flux may be defined along the boundary of the region [33], but as noted in [75], this does not readily extend to color images. Optimizing general normalized cost functions directly is NP-hard but may be approximated [64, among others]. Alternatively, a subspace of solutions may be explored by varying the relative weighting of the boundary and region terms [34]. (The authors of [34] note, however, that this method is not fast enough for interactive color image segmenta-

tion.) More recent work in [63] uses a curvature-minimizing rather than length-minimizing regularization term to smooth the resulting boundaries while avoiding shortcutting, but this does not use an edge component to localize edges, nor does it run in interactive time.

Because of the complementary strengths and weaknesses of seed-expansion and graph-cut approaches, some have suggested combining them. Work in [69] showed that graph-cuts and random-walkers [25] (a form of seed expansion), along with a new method similar in principle to geodesic segmentation [4], could be placed in a common framework in which the three methods differ by only the (integer) selection of a single parameter, an idea further expanded in [16] by varying a second parameter. They also showed empirically that of these three approaches the new method (analagous to geodesic segmentation) is most sensitive to seed placement while "because of the 'small cut' phenomenon, the Graph Cuts segmentation is the least robust to the quantity of seeds." They went on to suggest a way that the two "could be combined" but did not explore this idea further in that work. More recent work [66] has explored a way to blend the respective strengths of these methods using non-integer selections for the free parameter in [69], determining a suitable parameter selection empirically over a set of sample images with known ground truth.

This paper introduces a new method for interactive segmentation that makes the following three contributions. First, it combines geodesic-distance region information with explicit edge information in a graph-cut optimization framework. This combines the ability of seed-expansion approaches to fill contiguous, coherent regions without regard to boundary length with the ability of edge-based segmentation to accurately localize boundaries. Second, it uses pre-segmentation evaluation of the color models inferred from the user's seeds to assess the likely effectiveness of the geodesic distance component and weights the terms accordingly. This avoids the tendency for geodesic segmentation to degenerate to simple distance maps when the foreground/background color models are indistinct. Third, it introduces a spatially-varying weighting based on the local confidence of the geodesic component and uses this to further adjust the relative weighting of the terms. This makes cuts

even more expensive in object interiors (or exteriors) while transferring more control to the edge-localizing component when near the object's boundary. The result is a method that adapts both globally and locally to the relative strengths of each approach, providing better boundary placement than geodesic segmentation and stronger region connectivity and less short-cutting than typical graph-cut methods (Figure 4.1). This leads to less user interaction needed to produce a desired segmentation.

4.3 Geodesic Segmentation Revisited

Geodesic segmentation [4], like other seed-expansion approaches, can robustly segment long, thin structures without regard to boundary length. By incorporating mixture-based color models inferred from the user's seeds into an inter-pixel distance metric, it can select even multicolored or textured objects. Although we point out here key limitations of this approach in order to address and improve upon it, we refer the reader to the original paper for examples of its successful application.

Geodesic segmentation labels pixels by computing their geodesic distance $D_l(x)$ from the nearest foreground ($\mathcal{F}$) and background ($\mathcal{B}$) strokes using

$$D_l(x) = \min_{s \in \Omega_l} d_l(s, x) \tag{4.1}$$

where Ω_l is the set of seeds with label $l \in \{\mathcal{F}, \mathcal{B}\}$. The pixel is labeled according to the smaller of the two distances.

The geodesic distance from any point to any other according to the color model for the label l is given by

$$d_l(x_0, x_1) = \min_{L_{x_0,x_1}} \int_0^1 |W_l(L_{x_0,x_1}(p)) \cdot \dot{L}_{x_0,x_1}(p)| \; dp \tag{4.2}$$

where L_{x_0,x_1} is a path parameterized by $p = [0, 1]$ connecting x_0 to x_1 respectively, and $W_l(x)$ gives the geodesic weight according to the model l. [4] defines $W_l(x) = \nabla P_l(C(x))$, where

$$P_l(c) = \frac{Pr(c|l)}{Pr(c|\mathcal{F}) + Pr(c|\mathcal{B})}, \tag{4.3}$$

$C(x_i)$ is the color at x_i, and $Pr(c|l)$ is the probability of the color c given by the color model generated from pixels Ω_l.

Geodesic segmentation performs best when the geodesic distance between neighboring pixels inside of (or outside of) the desired object is small relative to the geodesic distance between neighboring pixels across the object boundary. This requires an accurate foreground/background color model that is consistent in assigning probabilities to the pixels. However, even small errors or variations in the probabilities can accumulate over the geodesic paths and lead to incorrect results in two keys ways:

1. If the color models are not distinct, the probabilities $P_l(c)$ may be highly unstable from one pixel to the next due to unavoidable image noise. This neutralizes the color-based distance metric and in the limit causes geodesic segmentation to degenerate to simple (and noisy) distance maps. This causes the segmentation to be highly sensitive to seed placement, as illustrated in Figure 4.2.

2. Even for more distinct color models, the lack of an explicit edge-finding component can cause geodesic segmentation to come close to but not precisely localize object boundaries (as in Figure 4.3 for a simple 1-D example). As the transition in the geodesic distances increases the method is more likely to place the boundary correctly, but with softer edges or with even modest noise it can sometimes fail to do so.

The methods in this paper address these two limitations respectively by 1) globally weighting geodesic-distance component by assessing the relative distinctiveness of the foreground/background color models, and 2) transferring relative control from the geodesic-segmentation component to more explicit edge-finding near object boundaries.

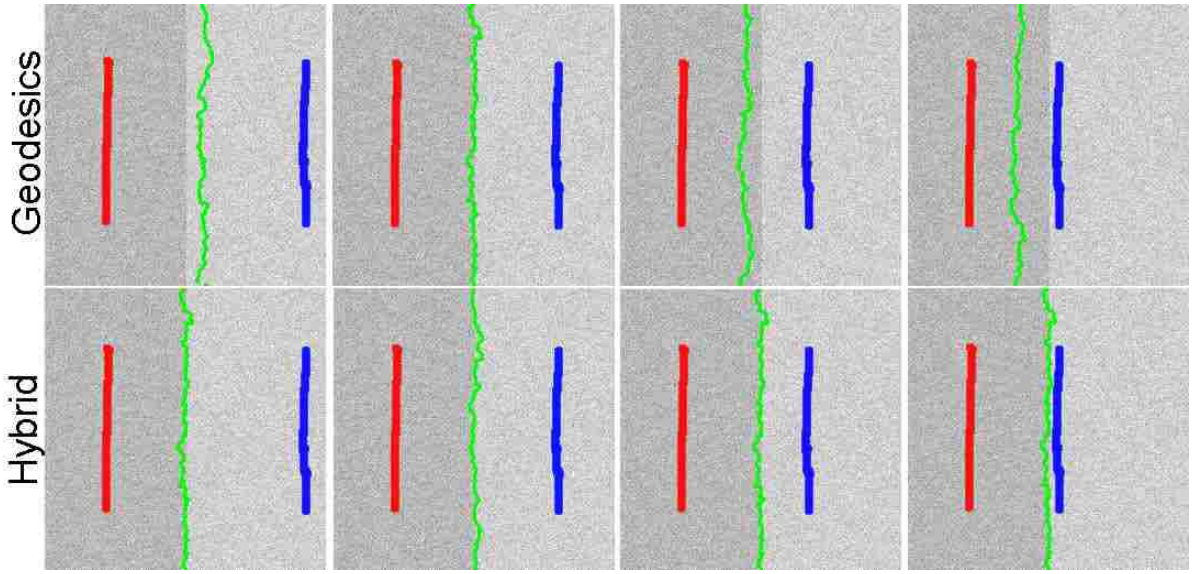


Figure 4.2: Sensitivity of geodesic segmentation to seed placement. As the background stroke (blue) is translated, the object boundary computed by geodesic segmentation shifts also (top row), while geodesic graph cut (and other graph cut approaches) give a more consistent result (bottom row).

We note here one related approach of interest [18], in which a number of geodesic distance transforms are generated by varying the parameters to generate multiple candidate solutions. The candidate minimizing a cost function that includes an edge-finding component (similar to that used in graph-cut approaches) is then selected as the final result. This method differs from that proposed here in that the region and edge information that graph cut uses are not used while generating geodesic candidates. Because of this, the space of candidate solutions may only approximate the optimal solution.

4.4 Graph Cut Segmentation Revisited

Graph cut segmentation [9] seeks to minimize a cost function of the form

$$E(\mathcal{L}) = \sum_{x_i \in P} R_{\mathcal{L}_i}(x_i) + \lambda \sum_{(x_i,x_j)\in N} B(x_i, x_j) \;\; |\mathcal{L}_i - \mathcal{L}_j| \tag{4.4}$$

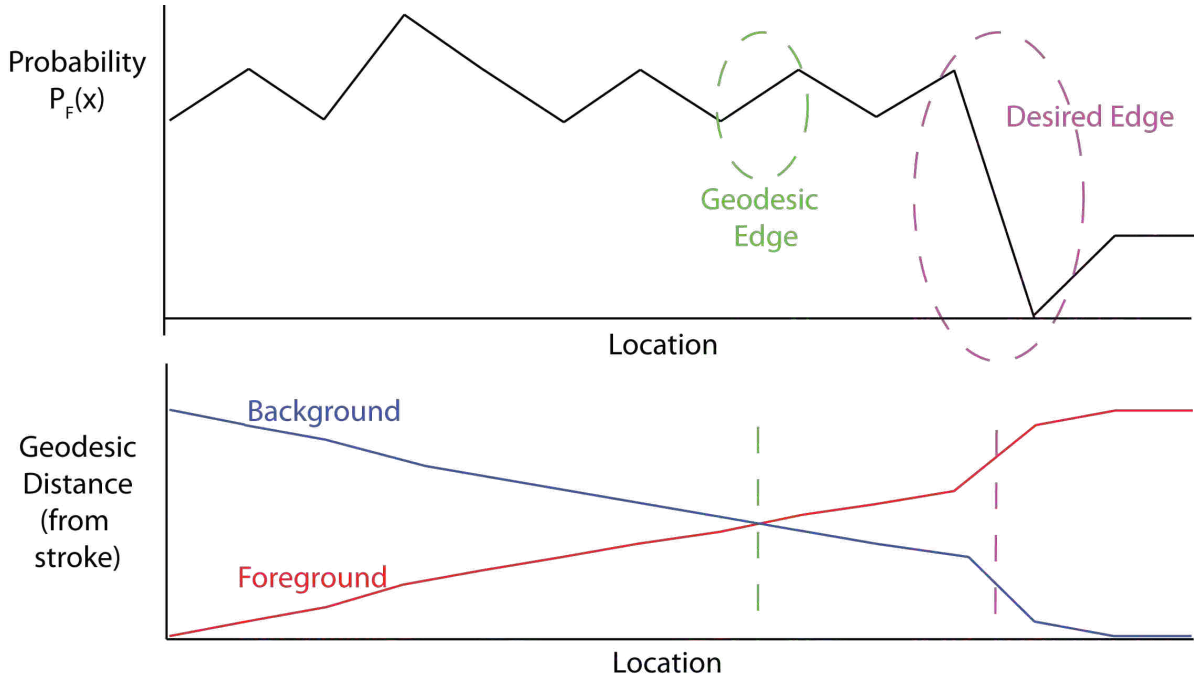


Figure 4.3: Why geodesic segmentation can miss edges. Because of noise in the probability along the geodesic paths (top), the geodesic boundary (green) between the user's foreground stroke and background stroke misses the true image edge (magenta).

where $\mathcal{L} = (\mathcal{L}_i)$ is a binary vector of labels and $\mathcal{L}_i$ is the label $\mathcal{F}$ or $\mathcal{B}$ for pixel x_i, $R_l(x_i)$ is a region cost term based on the label l, $B(x_i, x_j)$ is a boundary cost-term, λ is a relative weighting of R and B, P is the set of pixels in the image, and N is the set of pairs of neighboring pixels. The terms R and B have been defined in various ways by different researchers. Generally B corresponds to a measure of the similarity between the colors of adjacent pixels and R is based on color models of the foreground and background.

Graph cut methods minimize Eq. 4.4 by casting the problem as a graph-partitioning one and using the mincut/max-flow graph algorithm, where boundary costs are assigned to graph edges between pixels and region costs are assigned to edges that connect pixel nodes to the source and sink nodes [9].

Graph-cut methods perform well over a variety of images. Because both region and boundary information are explicitly captured in the algorithm, they are capable of both selecting objects consistent with region information and placing object boundaries on image

edges. Many variations have sought to improve on this approach, including using watershed regions as primitives in order to reduce the size of the graph and accelerate the computation [40], using tensor-based region terms to model texture [44], and iteratively alternating segmentation and model updating to converge to the solution [60]. Because it can easily operate on multi-dimensional images, the graph-cut approach has also been applied to videos [41, 80] and image volumes [3, 11, 42]. It is important to note that the term "graph cut segmentation" has grown to encompass a wide variety of approaches that minimize cost functions of the form in Eq. 4.4 by framing the problem as a graph-based min-cut/max-flow one. (See [11] for an excellent discussion of the variety of approaches for which graph-cut optimization has been used.)

As noted in Section 4.2, a key weakness of graph-cut approaches is that the boundary term in Eq. 4.4 causes an inherent shrinking bias toward shorter segmentation boundaries. This can be especially noticeable when the algorithm short-cuts across the interior of an object to avoid segmenting an appendage as illustrated in Figure 4.4. The ideal boundary of the object contains the segment B_2, but since the segment B_1 is so much shorter, the cost of cutting all the links along B_1 may be less than the cost of cutting the links along B_2, even if no individual link in B_1 is cheaper than in B_2. This leaves the region A_{12} out of the selection.

Using this intuition, short-cutting may be prevented by increasing the cost of B_1 relative to B_2. However, increasing the edge sensitivity may cause even weak gradients to become attractive options. A better strategy is to increase the incurred region cost of the shortcut by increasing the sensitivity of the color model or by increasing its weight relative to the boundary terms by decreasing λ. However, this can have an ill effect when using a global color-similarity model as is common with graph-cut methods. Other background objects with properties (region A_0 in Figure 4.4) similar to the foreground user stroke are more likely to be incorrectly segmented as foreground in such cases. Problems with short-cutting and selection of disconnected unseeded regions can be reduced by allowing users to

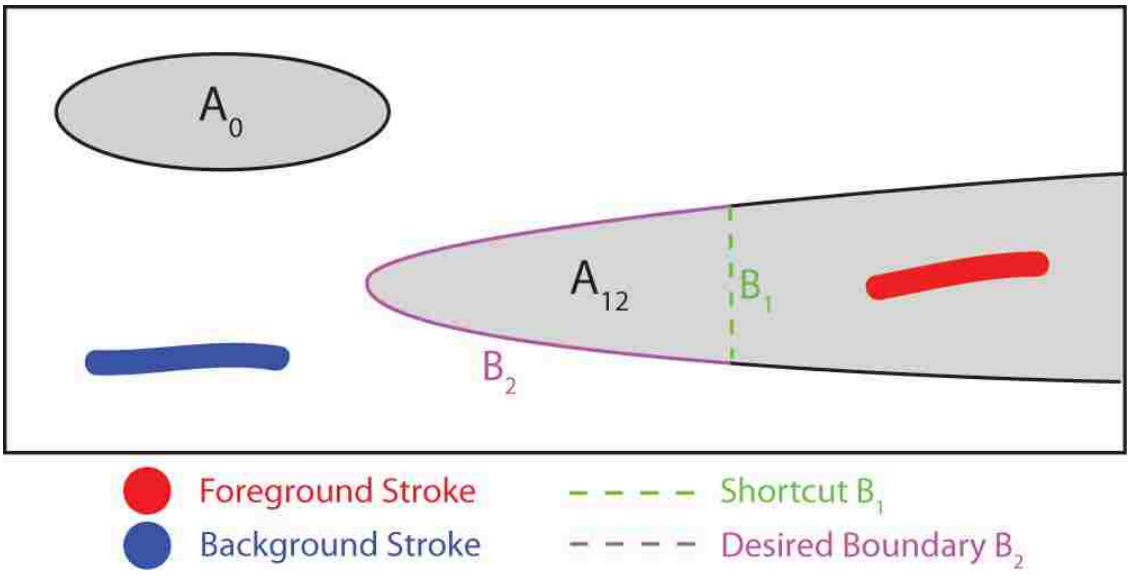


Figure 4.4: Shortcuts. Graph-cut methods may short-cut across the desired object along B_1 instead of following the true edge B_2 because less cost is accrued transversing the shorter path.

explicitly specify that certain regions should stay connected or disconnected [75], but this requires either prior knowledge or further user interaction.

4.5 Geodesic Graph Cut

As discussed in Section 4.3, in cases where the color models inferred from the user's strokes are indistinct, geodesic segmentation can be improved by the inclusion of explicit edge information to encourage placement of selection boundaries on edges in the image and allow the user more freedom in placing strokes. In cases where the color models are more distinct, though, the edge information (with its inherent shrinking bias) is not as necessary. The region term alone can often carry the segmentation in such cases, but as discussed in Section 4.4 global color models without spatial locality information can often select disjoint regions. The use of geodesic distance rather than simple color-similarity alone can avoid this. This section presents how geodesic distances and edge information can be combined in a graph-cut optimization framework, and then presents a way to use the predicted classifi-

cation accuracy from the inferred color models to automatically tune the tradeoff between the strengths and weaknesses of the two.

4.5.1 Using geodesic distance as a unary term

We formulate our algorithm in simplest form as a graph-cut problem using a normalized form of geodesic distance as one of the unary region terms. Using Eq. 4.4 and minimizing it using graph cut, we compute the unary region term as follows:

$$R_l(x_i) = s_l(x_i) + M_l(x_i) + G_l(x_i) \tag{4.5}$$

where $M_l(x_i)$ is based on a global color model as is often used for graph-cut segmentation, $G_l(x_i)$ is based on geodesic distance, and

$$s_l(x_i) = \begin{cases} \infty & \text{if } x_i \in \Omega_{\bar{l}} \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

indicates the presence of a user stroke where $\bar{l}$ is the label opposite l (i.e. if $l = \mathcal{F}$, then $\bar{l} = \mathcal{B}$).

We use the Fast Gauss Transform [83] to compute the foreground/background color models $P_l(c)$ (see Eq. 4.3) for both global similarity and geodesic distances. $M_l(x_i)$ is computed by

$$M_l(x_i) = P_{\bar{l}}(C(x_i)). \tag{4.7}$$

$G_l(x_i)$ is computed by normalizing the relative foreground/background geodesic distances (see Equation 4.1):

$$G_l(x_i) = \frac{D_l(x_i)}{D_{\mathcal{F}}(x_i) + D_{\mathcal{B}}(x_i)}. \tag{4.8}$$

and using the efficient method in [4] to compute the distances $D_l(x_i)$.

For the boundary term we use [40]:

$$B(x_i, x_j) = \frac{1}{1 + ||C(x_i) - C(x_j)||^2} \tag{4.9}$$

where $C(x) \in [0, 255]$.

4.5.2 Global weighting based on color model error

The simple form (Eq. 4.4) in Section 4.5.1 can give good results in many cases, generally performing better than either geodesic or graph-cut segmentation alone. However, we would like to allow it to provide greater weight to the geodesic-based unary term in cases where this method is known to perform well, specifically when the foreground/background color distributions are well-separated. This increased reliance on geodesic distance for the region term serves to reduce the potential for short-cutting due to the boundary term. But caution must be exercised with this, because over-reliance on the geodesic component can cause increased sensitivity to seed placement when the color models are not distinct.

To allow for global weighting of the relative importance of the region and boundary components, we modify Eq. 4.4 as follows:

$$E(\mathcal{L}) = \lambda_R \sum_{x_i \in P} R_{\mathcal{L}_i}(x_i) + \lambda_B \sum_{(x_i, x_j) \in N} B(x_i, x_j) \ \ |\mathcal{L}_i - \mathcal{L}_j| \tag{4.10}$$

Although we could fold the separate region (λ_R) and boundary (λ_B) weights into a single weight, we choose to keep them separate to make their respective purposes clearer. The boundary weight λ_B serves the role of the traditional fixed region/boundary weighting in graph cut methods, and we adjust it to individual images by considering only the size of the image (due to the disproportionate scaling of an object's area (unary term) and perimeter (boundary term)). The region weight λ_R is the relative weighting of the geodesic-distance and other region components. While the user could tune λ_R manually, this would require excessive tweaking and is undesireable; instead, we want to automatically tune this parameter

on a per-image basis by predicting the segmentation performance of the geodesic distance term.

To do this, we consider that Eq. 4.3 is the posterior probability of a pixel with color c belonging to foreground ($\mathcal{F}$) or background ($\mathcal{B}$) respectively, assuming equal priors. As such, it is essentially functioning as a simple Bayesian classifier, the error in which can be estimated by (using the notation of Eq. 4.7)

$$\varepsilon = \frac{1}{2}\left[\frac{\sum_{x\in\mathcal{F}} P_{\mathcal{B}}(C(x))}{|\Omega_{\mathcal{F}}|} + \frac{\sum_{x\in\mathcal{B}} P_{\mathcal{F}}(C(x))}{|\Omega_{\mathcal{B}}|}\right] \tag{4.11}$$

When there is no error ($\varepsilon = 0$), we would like to give the color-based terms (M and G) full weight, and when the color models become indistinct ($\varepsilon \geq 0.5$), we want to give them no weight:

$$\lambda_R = \begin{cases} 1 - 2\varepsilon & \text{if } \varepsilon < 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{4.12}$$

4.5.3 Local weighting based on geodesic confidence

Globally adjusting the relative weighting of the region and boundary terms on a per-image basis can help automatically tune the method to different image types, but it does not account for the properties of different local areas. We further weight the geodesic and boundary terms based on the local confidence $u(x)$ of the geodesic components:

$$u(x_i) = \left|\frac{D_{\mathcal{F}}(x_i) - D_{\mathcal{B}}(x_i)}{D_{\mathcal{F}}(x_i) + D_{\mathcal{B}}(x_i)}\right|^{\gamma} \tag{4.13}$$

where empirically we have found $\gamma = 2$ to 2.5 to work well. (For the results in Section 4.6, we use a value of $\gamma = 2.5$.)

We redefine the region terms to weight the geodesic component by $u(x_i)$:

$$R_l(x_i) = s_l(x_i) + M_l(x_i) + u(x_i)G_l(x_i) \tag{4.14}$$

This maintains the weight of the geodesic distance term when relatively certain that the pixel x_i is clearly in the object's interior or exterior ($u(x_i)$ close to 1) and decreases it near where geodesic segmentation would place boundaries ($u(x_i)$ close to 0).

We also correspondingly spatially adapt the weighting of the boundary costs based on $u(x)$ as follows:

$$B(x_i, x_j) = \frac{1 + (u(x_i) + u(x_j))/2}{1 + ||C(x_i) - C(x_j)||^2} \tag{4.15}$$

Note that when the average geodesic certainty of the two pixels is high, this suggests an object interior/exterior, and the cost of placing a cut here is further increased. When this geodesic confidence is low, this suggests that geodesic segmentation alone would consider this to be near a boundary, and we reduce the effect of the geodesic component, shifting control to the more accurate edge-finding term.

The net effect of this spatially adaptive weighting is to both *increase* the relative weighting of the unary geodesic distance term and *increase* the cost of a boundary cut in what are clearly interior/exterior regions, while both *decreasing* the relative weighting of the unary geodesic term and *decreasing* the cost of a boundary cut in areas where we want to more accurately localize the object boundary.

4.6 Results

Geodesic graph cut with automatic tuning and spatial adaptation works well both in cases suitable for geodesic segmentation and in cases suitable for standard graph cut methods, in many cases outperforming both. While accuracy is an essential element of interactive segmentation, so too is the minimization of user interaction required to achieve that level of accuracy. This section demonstrates the accuracy of geodesic graph cut, and the interaction required for these examples is shown in recorded videos[1]. The time for our algorithm on these images ranged from 0.2–2.6 seconds for image sizes from 256×256 to 720×480, with most computations requiring approximately 1 second or less. *In all cases the unary term*

[1] http://vision.cs.byu.edu/papers/ggc

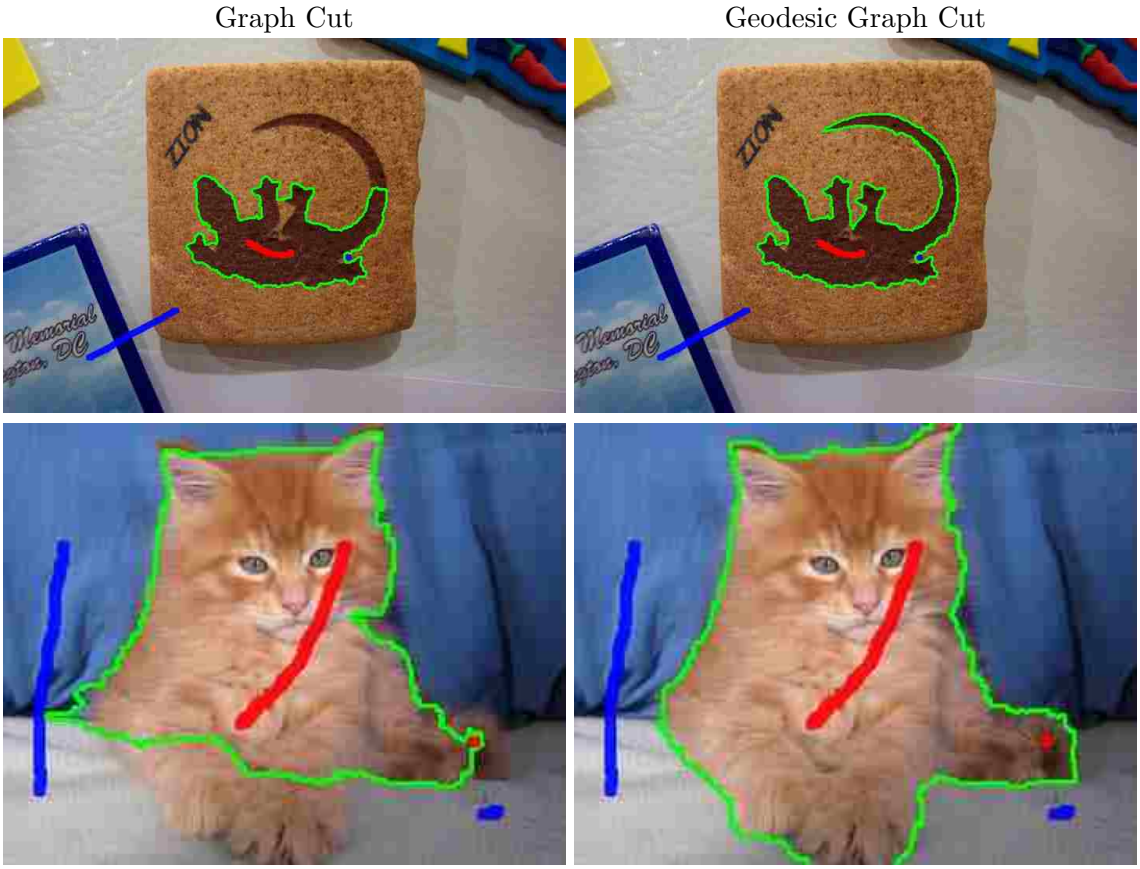


Figure 4.5: Examples with distinct color models. For these images geodesic graph cut segmentation automatically relies more on geodesic distances ($\lambda_R = 0.97, 0.99$ respectively), avoiding short-cutting common to non-adaptive graph-cut methods.

weighting (λ_R) and the spatially adaptive weights ($u(x)$) are set automatically by the method, with no per-image manual tuning.

In Figure 4.1, geodesic segmentation fails to segment along the dolphin's back due to the specular reflection on the table. Graph cut, because of its explicit edge term, can better segment along the back but shortcuts across the tail. Geodesic graph cut leverages the strengths of each approach and correctly segments both areas. While additional strokes could of course correct either the graph-cut or geodesic segmentation, this increases the required user interaction.

Figure 4.5 shows examples where geodesic graph cut segmentation automatically adjusts to the distinct color models, exploiting the geodesic distance term to avoid shortcutting.

For these images, geodesic segmentation performs comparably to the results shown here for our method.

Figure 4.6 shows similar examples for images whose color models are less distinct. In these cases, our method recognizes the error in the color models and automatically adjusts to rely less heavily on geodesic distances. In the pyramid (top) and candy (middle) examples, the foreground and background color models overlap considerably. Without distinct color models, geodesic segmentation alone fails noticeably. In particular, the candy (middle) result demonstrates the way geodesic distance can degenerate to noisy distance maps when the colors are not distinct. In the ram (bottom) example, the color models are more distinct but are still mixed enough to cause geodesic segmentation to mistakenly select the part of the background away from the user-placed background seeds. The results of graph cut segmentation for these examples are comparable to our method.

Figures 4.7 and 4.8 show examples where our method outperforms both geodesic and graph-cut segmentation. For the rolling-pin example (top), because of the similarity between the foreground and background colors, geodesic segmentation again mistakenly selects part of the background away from the seeds. Graph cut segmentation avoids this problem but again exhibits shortcutting. Our method places only a moderate geodesic distance weight for this image ($\lambda_R = 0.72$), avoiding the problems exhibited by geodesic segmentation alone while using sufficient weighting of this term to avoid region shortcutting. For the elephant example (bottom), geodesic segmentation fails to segment along the top of the elephant's back due to the similar background even though there is still an edge there, while graph cut segmentation shortcuts the trunk and part of the back. Geodesic graph cut corrects both of these problems for this image.

To quantitatively evaluate the accuracy of the segmentations produced by geodesic graph cut, we applied it to the Microsoft GrabCut dataset [7]. As noted in [20], this dataset is not well suited to evaluating interactive scribble-based segmentation because it assumes the user loosely traces the contour of the desired object. As such, it provides far more seeds

than are typically provided with interactive scribbles, and the seeds are more uniformly placed on either side of the boundary. We believe that interactive scribble-based methods cannot be evaluated with static seeds—once the user places the first scribble the resulting scribbles are dependent on the segmentation result from that and each successive one. But, as also noted in [20], this is the only evaluation database to provide seeds, and we compute our results on this dataset for comparison (Table 4.1).

Over all 50 images in the database, geodesic graph cut averaged 4.8% error, better than either geodesic segmentation (6.8%) or standard graph-cut segmentation (6.7%) individually. This error was also lower than that of any of the other compared-to methods that do not have a spatial position bias, either explicitly [37] or implicitly [27].[2] To our knowledge, this is the lowest error rate reported for this dataset by a scribble-based selection method.

For 48 of the 50 images, geodesic graph cut outperformed either graph cut (43 of 50) or geodesic segmentation (39 of 50) alone, typically performing at a level near the better of the two methods for each image. For 34 of the 50 images, it outperformed both.

We also used this dataset with the skeleton-based initialization suggested in [66] as provided by its authors. This provides fewer seeds overall but tends to place more seeds in object protrusions. Graph cut segmentation had an error rate of 6.3% with this form of initialization, while geodesic segmentation had an error rate of 10% and geodesic graph cut had an error rate of 3.6%.

From our observations, geodesic graph cut usually does as well or better than the better of the two individual methods. It struggles on inputs for which both individual methods have difficulties. This typically happens when the foreground and background color models are overlapping and either there are weak edges or highly textured regions around the object boundary. Another less severe problem we have observed is that some long thin

[2] As noted in [20], the adaptive thresholding method of [27] has a strong bias towards placing an object boundary in the middle of the uncertainty band of the trimap. This is because it uses an adaptive window that grows to "include at least one $\alpha = 0$ pixel and one $\alpha = 1$ pixel." Due to the nature of the GrabCut database, which uses an uncertainty band centered close to the actual boundary, this causes this method to have artificially low error. Although [20] use and report this method for their method and [25], they add this specific disclaimer on the reported error rates.

Segmentation Model	Error Rate
GMMRF [7]	7.9%
Geodesic Segmentation [4]	6.8%
Graph Cut (as reported by [37])	6.7%
Random Walker (s=2) [25]	5.4%
Segmentation by Transduction [20]	5.4%
Geodesic Graph Cut (proposed method)	**4.8%**
Guan and Qiu [27] with AT*	4.6%
Random Walker (s=2) [25] with AT [27]*	3.3%
Segmentation by Transduction [20] with AT [27]*	3.3%
GrabCut-GC (as reported by [37])	5.9%
Bounding Box Prior (LP-Pinpoint) [37]	5.0%
Bounding Box Prior (GrabCut-Pinpoint) [37]	3.7%

Table 4.1: Quantitative comparison using the Microsoft GrabCut database [7]. Error rates reported here are either computed by us (our method, [4]), reported by the method's authors ([7, 27]), or were previously reported by [20] or [37]. The first nine were initialized using the "Lasso" form of the trimap provided by the database. The adaptive thresholding (AT) method of [27] is biased towards the middle of the trimap's uncertainty band and is artificially favored by this form of initialization. The lower three used the corresponding bounding box initializations, with the last two using this box as a prior on the spatial extent of the object.

structures are sometimes still short-cutted near the tip, although our adaptive weighting usually decreases the size of the short-cutted area and thus requires less additional effort from the user to correct than typical graph cut. This problem is most likely to occur when there are strong edges across or texture within the long thin structure and when the color models overlap enough that our algorithm cannot rely on the geodesic information to entirely prevent a shortcut.

4.7 Conclusion

This paper has presented a way to incorporate both geodesic-distance region information and explicit edge information together in the popular graph-cut optimization framework in a way that leverages the strengths of each. Rather than a simple fixed combination, the method tries to best leverage the respective strengths of the two approaches by adaptively tuning their combination based on pre-segmentation assessment of the classification performance of the color models inferred from the user's input. When the image's foreground/background color

models as inferred from the user's marked seeds are distinct, greater weight is given to the geodesic-distance component in order to provide greater region coherence and avoid boundary short-cutting. As the color models become less distinct, the geodesic-distance approach becomes increasingly unreliable and is weighted less accordingly. In addition, a spatially adaptive weighting is introduced that makes boundary short-cutting more expensive in object interiors or exteriors while transferring greater control to the edge-finding component to better localize edges near object boundaries. Results demonstrate that geodesic graph cut is able to segment objects in a variety of images, generally performing as well as the better of these two methods for each image, and often outperforming both methods.

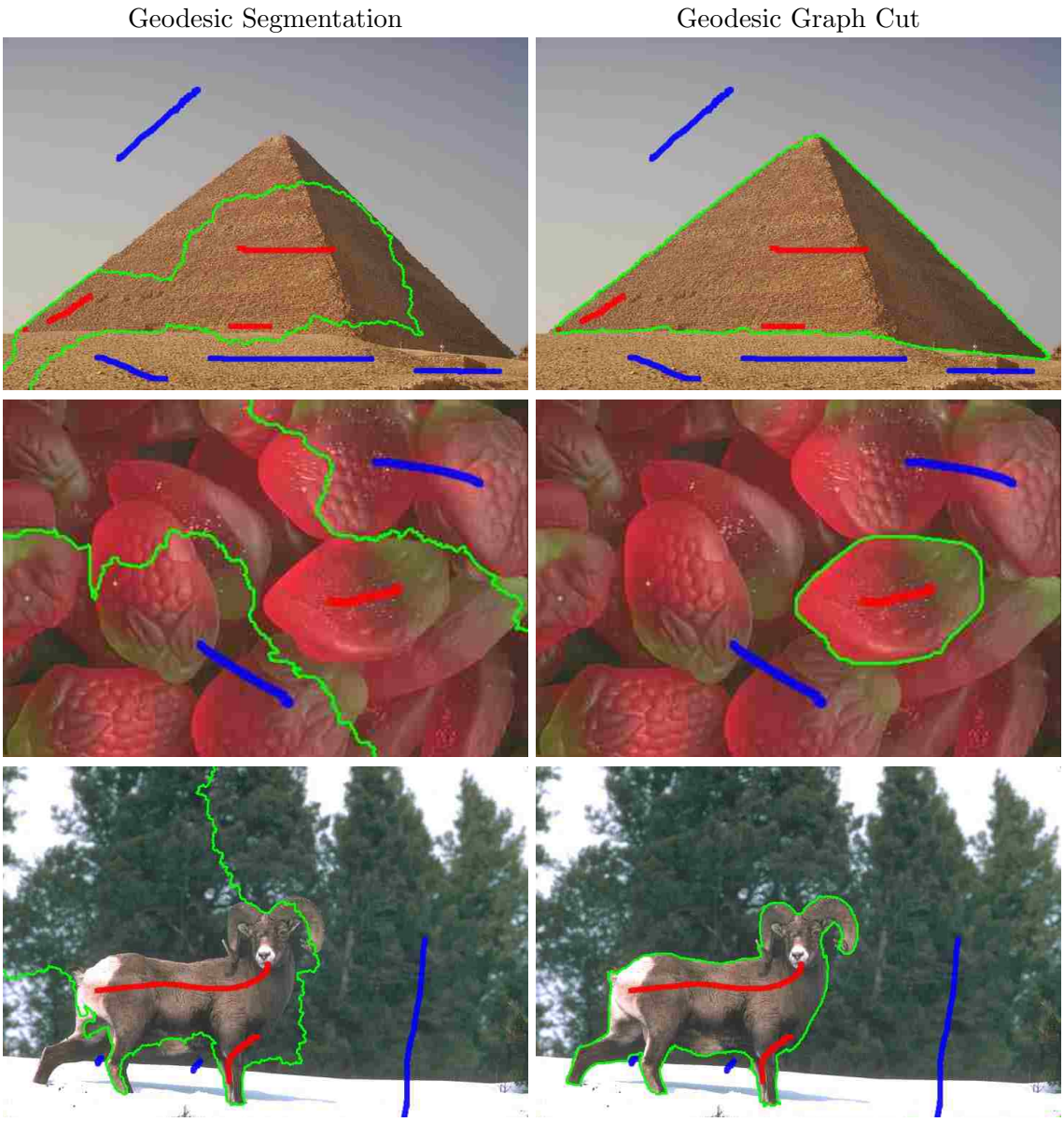


Figure 4.6: Examples with less-distinct color models. In these cases geodesic graph cut cannot rely on the geodesic distances and automatically adjusts to rely more on explicit edges or a combination of color models and edges ($\lambda_R = 0.40, 0.22, 0.65$ respectively).

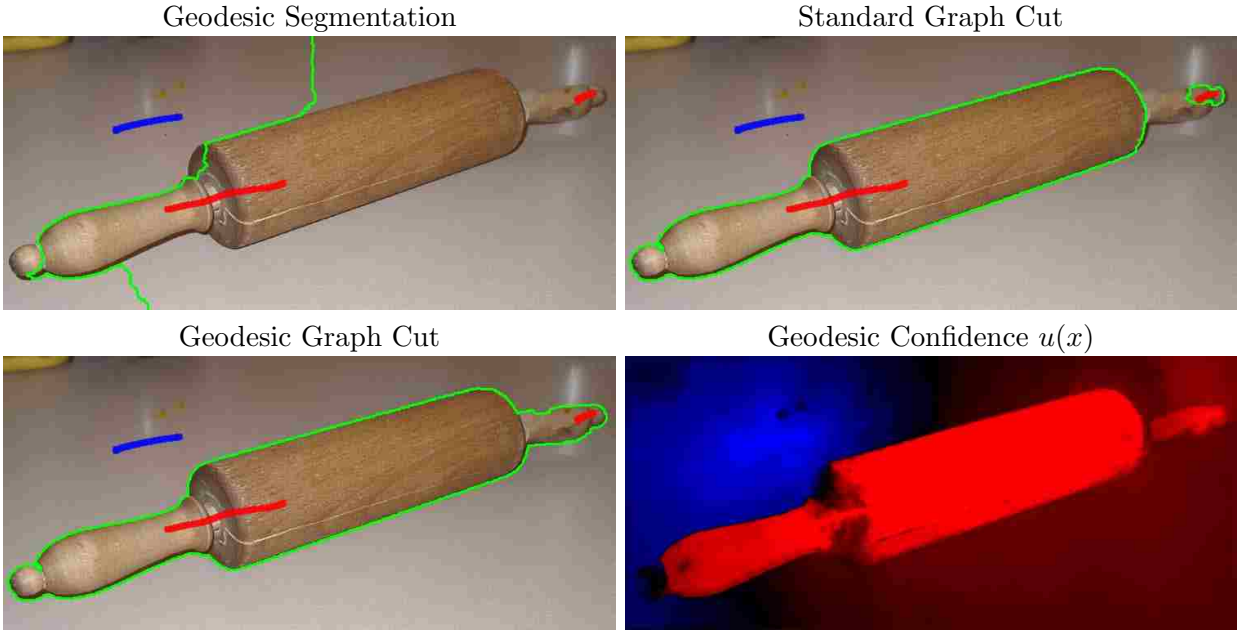


Figure 4.7: Example for which geodesic graph cut segmentation outperforms both geodesic segmentation and standard graph cut. Geodesic segmentation mislabels the area beneath the rolling pin because of its distance to the background user stroke. Graph cut segmentation shortcuts across the handle. Our method correctly segments the rolling pin.

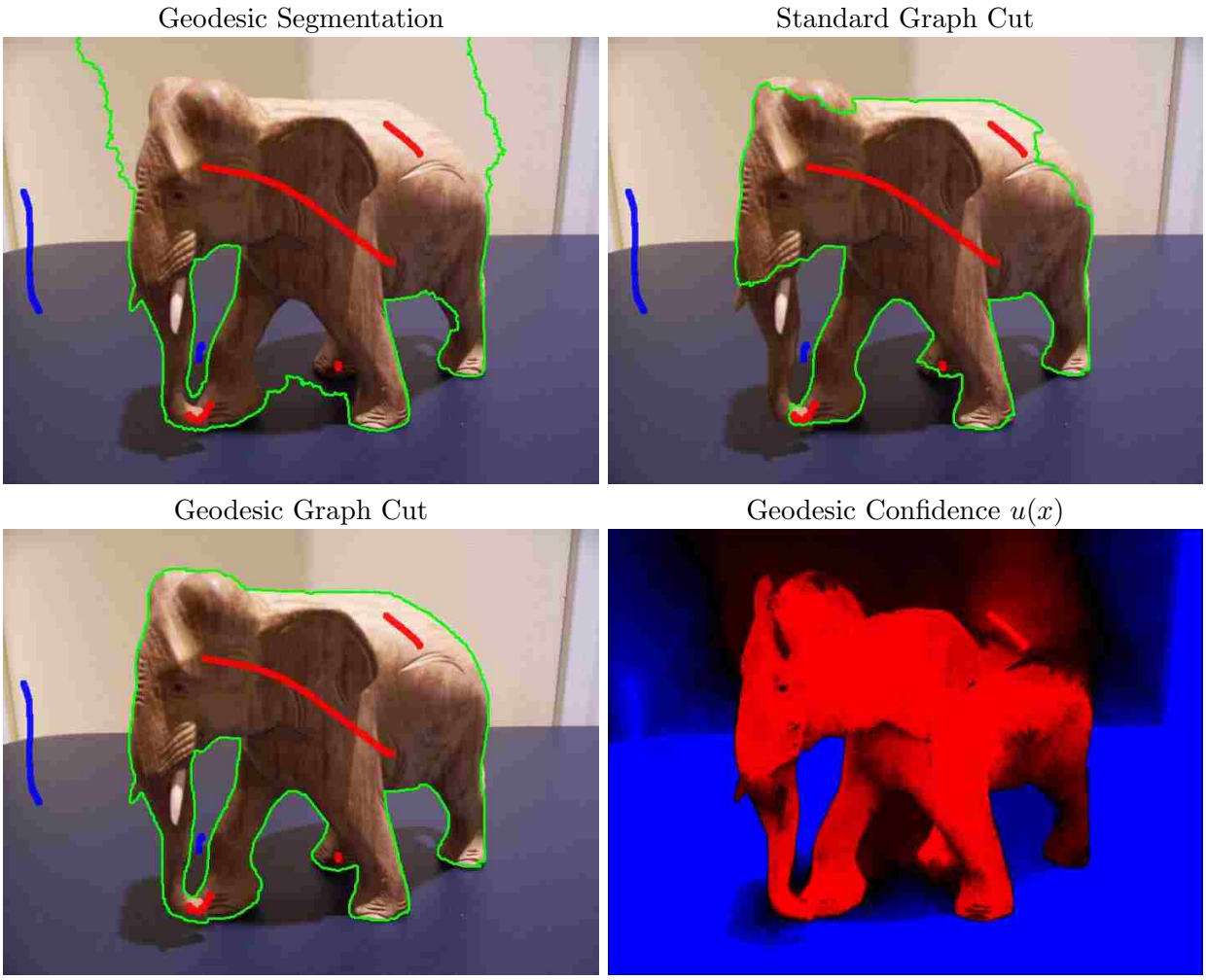


Figure 4.8: Example for which geodesic graph cut segmentation outperforms both geodesic segmentation and standard graph cut. Geodesic segmentation mislabels the wall above the elephant because of the similarity in color to the elephant and the distance to the background user strokes. Graph cut segmentation shortcuts across the elephant's trunk, forehead, and back. Our method correctly segments the elephant.

Chapter 5

Foreground, Background and Alpha Matting

Chapters 2 through 4 present methods for producing a "hard" segmentation, a segmentation where each pixel is assigned to either the foreground object or the background. For many pixels, however, the color present at the pixel is actually a blended combination of the color of the foreground and the color of the background. This is true at the edges of objects, for pixels covering semi-transparent objects, and for objects thinner than a pixel such as hair. To properly segment these objects a "soft" segmentation must be produced that assigns to a pixel a certain percentage of foreground and a percentage of background.

This chapter addresses the problem of matting for images. Most previous work solves for the percentage of foreground versus background, called the alpha value, while ignoring the foreground and background colors at the pixel. This chapter presents a method of solving for the foreground color, background color, and alpha value at a pixel simultaneously. This chapter was originally published in the *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (CVPR) in June 2010 under the title *Simultaneous Foreground, Background, and Alpha Estimation for Image Matting* [53].

5.1 Abstract

Image matting is the process of extracting a soft segmentation of an object in an image as defined by the matting equation. Most current techniques focus largely on computing the alpha values of unknown pixels and treat computation of the foreground and background colors as an afterthought, if at all. However, for many applications, such as compositing an

object into a new scene or deleting an object from the scene, the foreground and background colors are vital for an acceptable answer. We propose a method of solving for the foreground, background, and alpha of an unknown region in an image simultaneously. This allows for novel constraints to be placed directly on the foreground and background as well as on alpha. We show through both visual results and quantitative measurements on standard datasets that this approach produces more accurate foreground and background values at each pixel while maintaining competitive results on the alpha matte.

5.2 Introduction

The goal of image matting is to produce a soft segmentation of an image I by computing the relative contribution of foreground F and background B at each pixel according to the equation

$$I = \alpha F + (1 - \alpha)B \tag{5.1}$$

where α is the opacity of the foreground. If I is a three-channel color image, the matting formula yields seven unknowns with only three equations, making the solution greatly underconstrained. Most techniques constrain the system with a user-defined trimap (which identifies known foreground and background regions and unknown regions) as well as additional terms to produce an energy function to optimize or a system of equations to solve.

Matting is important for image and video editing when one wishes to select an object exactly for editing. While some applications may only require an alpha value for the selected object, such as perhaps applying a filter that tails off as alpha drops to zero, many others require both the alpha value and the extracted foreground and/or background color. A prime example of this is composition, where a selected object may have a noticeable "halo" around the object when placed on a different-colored background if the background colors are still maintained in the foreground. Another such task is object deletion, where the background colors are needed to restore the image.

Interestingly, most existing matting algorithms do not attempt to solve for the foreground and background colors, focusing on solving for alpha exclusively [26, 39, 56, 57, 67, 71, 81, 87]. Such solutions still result in an underconstrained problem, with three equations and six unknowns, which some methods attempt to solve as a postprocess [4, 38, 61, 79]. Of course, given a trimap, no information is initially known about alpha, but only about foreground and background. Some methods will constrain the foreground and background colors [38, 39, 56, 61, 67, 81] or begin with an initial estimate of them [56, 57, 71, 81], but few attempt to solve for them alongside alpha.

In this paper, we introduce a method for solving the foreground, background, and alpha simultaneously and make the following three contributions. First, by optimizing over the foreground and background as opposed to only addressing them as a postprocess or not at all, we produce more accurate foreground and background values. Second, we produce alpha matte results that are competitive with the state-of-the-art. Third, we contribute several new ideas regarding the additional terms used to solve the matting equation. These include a novel color term that affects the foreground and background directly, as well as an important observation and new terms involving the gradient of the matting equation and smoothness assumptions.

5.3 Related Work

Despite the matting problem consisting of seven unknowns per pixel, the majority of matting algorithms focus only on computing one unknown, the alpha channel. Bayesian Matting [14], an exception to this, computes foreground, background, and alpha to provide a complete solution to the matting equation. The colors of the known foreground and background are represented using Gaussians, and the unknown values are computed by an optimization over this information and the matting equation using *maximum a posteriori* estimation. While this approach computes the foreground and background, the imprecise representation of the known color values does not lead to robust results in the general case.

Easy Matting [28] also computes the foreground and background at each unknown pixel but limits the possible color to one of twenty samples, severely limiting the range of possible values. The alpha value is then computed using the matting equation and a smoothness prior on alpha. Similarly, Wang and Cohen [82] presuppose that the foreground and background belong to a small, unspecified sample set, and then compute an alpha using the matting equation and an alpha smoothness prior. Unfortunately, these methods often oversmooth the matte due to the alpha smoothness prior, and the limited freedom in foreground and background values limits their results.

While few algorithms explicitly optimize over the foreground and background, many do constrain those values to some extent. An early example of this is seen in the work of Ruzon and Tomasi [61]. The foreground and background are modeled as Gaussians, and the assumption is made that the color at the unknown pixels comes from a distribution between these Gaussians. An alpha matte is then computed by maximizing the probability of the image color on possible intermediary distributions.

The closed-form solution of Levin et al. [38] constrains the foreground and background colors in a local neighborhood to each be a linear combination of two colors respectively using the matting Laplacian. This allows them to eliminate the foreground and background colors from the matting equation and solve directly for alpha. While this works well in many cases, it fails when the color model may not be well represented by their linear model.

The elegant solution in [38] has inspired several improvements. Spectral matting [39] extends the framework to segment multiple layers without the use of a trimap or user interaction. The variant in [67] also extends this framework to computing mattes of multiple layers. Singaraju et al. [68] introduced an improvement to the matting Laplacian that handles cases where the colors in a neighborhood may be represented by a point rather than a line.

Another example of a paper that applies constraints to the foreground and background without solving for them is the work of Zheng and Kambhamettu [87]. They approach the matting problem in a semi-supervised machine learning paradigm. This allows them to model

the matting equation using both linear and non-linear equations, and is used in conjunction with a smoothness prior on alpha to compute an alpha matte.

Several other approaches include an initial estimate of the foreground and background in their computation but do not seek to improve those values. In Robust Matting [81] an estimate of the foreground and background is computed by sampling along the nearest edge in the known foreground or background regions. This information is used to form a sparsity bias that encourages most of the unknown pixels to take the value of 0 or 1. The sparsity bias is used in conjunction with the foreground and background smoothness constraint of [38] to produce an alpha matte. While this method works well in many cases, it can fail when its initial estimation of foreground and background is not accurate.

Rhemann et al. [56] formulate matte computation similar to [81] but with several improvements. The initial foreground and background estimation is improved by taking samples from known regions that are close to the pixel in question using a geodesic distance measure. A hard segmentation is also used to help formulate the sparsity prior. Robust Matting is also improved to handle high resolution images through interactive trimap generation in [57].

Poisson Matting [71] computes an estimate of the difference of the foreground and background colors in grayscale. This estimate is updated iteratively following a computation of the matting equation using Poisson equations under the assumption that the foreground and background are smooth. The simplified method of updating the foreground-background difference in grayscale and the smoothness assumption on the foreground colors prohibit good results without excessive user interaction.

Another approach to creating an alpha matte is to forgo the matting equation and compute alpha using other means. Grady et al. [26] equates alpha to the probability that a random walker will reach a given pixel from the known foreground and background regions. Bai and Sapiro [4] compute alpha as a ratio of geodesic distances from a pixel to the known foreground and background regions. While these approaches do well in some cases, it is

difficult to achieve good results over a wide range of problems without a more explicit representation of the matting equation.

Several methods compute the foreground and background as a postprocess after computing alpha. Ruzon and Tomasi [61] solve for the foreground and background by interpolating the means of their respective Gaussians. Levin et al. [38] minimizes an equation combining the matting equation and a smoothness prior on the foreground and background. Soft Scissors [79], an interactive extension of [81], computes the foreground using a random walk. Bai and Sapiro [4] solve for the foreground and background by randomly sampling the known areas looking for values that minimize the matting equation.

5.4 Methods

We approach the matting problem as an energy minimization problem over the seven values for each pixel in the unknown region of the trimap. By optimizing not just over alpha, but also over the foreground and background, we may introduce additional regularizations to the problem that are otherwise difficult or impossible. This leads to improved estimates of foreground and background colors compared to existing algorithms.

5.4.1 Problem formulation

We formulate our solution to the matting equation as an energy minimization problem of the form

$$E(X) = \sum_{p \in \Omega} \sum_{i=0}^{|U|-1} \lambda_i U_i(p) + \sum_{(p,q) \in N} \sum_{i=0}^{|V|-1} \gamma_i V_i(p,q) \tag{5.2}$$

where X is the set of 7-tuples $(F_r, F_g, F_b, B_r, B_g, B_b, \alpha)$ corresponding to the foreground, background, and alpha values at each pixel p in the unknown region Ω, U is the set of all unary terms being minimized, V is the set of all pairwise terms being minimized, N is the set of all pairs of tuples whose corresponding pixels are adjacent (4-connected), and λ_i and γ_i are weighting factors.

Since our formulation minimizes over foreground, background, and alpha, we may include terms that affect any of these values. The remainder of this section details the terms we use in Equation 5.2 and the resulting optimization.

5.4.2 Matting equation

Adherence to the matting equation (Equation 5.1) is maintained by

$$U_0(p) = ||I(p) - [\alpha(p)F(p) + (1 - \alpha(p))B(p)]||_2^2 \tag{5.3}$$

where $I(p)$ is the color of image I at pixel p (and similarly for F, B, and α).

5.4.3 Matting derivatives

We can constrain the solution not only with respect to the matting equation, but also with respect to the derivatives of the matting equation:

$$\nabla I = \alpha \nabla F + (1 - \alpha)\nabla B + (F - B)\nabla \alpha \tag{5.4}$$

We define the derivative between adjacent pixels p and q (in the x and y direction) as $\Delta_{pq} I_j = I_j(p) - I_j(q)$, where I_j is an image consisting of only the j^{th} channel of I (and similarly for F and B). We then derive the following term:

$$V_0(p, q) = \sum_{j \in (r,g,b)} [\Delta_{pq} I_j - \alpha(p)\Delta_{pq} F_j - (1 - \alpha(p))\Delta_{pq} B_j - (F_j(p) - B_j(p))\Delta_{pq}\alpha]^2 \tag{5.5}$$

5.4.4 Smooth matting gradients

While Equation 5.1 is known to be true, it has an infinite number of solutions, corresponding to any line segment in the color cube that passes through the color at the pixel. The matting derivative of Equation 5.4 is derived from Equation 5.1 and is insufficient to completely constrain the problem. Because of this, additional terms are needed. One common practice

in current methods is to add a smoothness assumption on the foreground, background, or alpha. In other words, one or more of the terms ∇F, ∇B, and $\nabla \alpha$ are assumed to be zero. The effect of this is to assume that the gradient of the image is not affected by gradients in the term that was set to zero.

Differing algorithms make different decisions about which gradient to set to zero. For example, several algorithms make the assumption that the foreground and background values are locally smooth. Poisson Matting [71] does so by directly modifying Equation 5.4 by setting ∇F and ∇B to zero, leaving

$$\nabla I = (F - B)\nabla \alpha \tag{5.6}$$

Smoothness assumptions on foreground and background are also made in Levin et al. [38] and other works incorporating its matting Laplacian [39, 56, 57, 68, 67, 81], although the smoothness assumption in [38] is admittedly more complex and elegant than that of Equation 5.6.

On the other hand, other papers, such as Easy Matting [28], Wang and Cohen [82], and Digital Matting [87] make the opposite assumption, that the alpha is smooth (setting $\nabla \alpha$ to 0), thereby minimizing

$$\nabla I = \alpha \nabla F + (1 - \alpha)\nabla B \tag{5.7}$$

It is interesting that different methods simplify Equation 5.4 to make opposite assumptions. Of course, in places where the change in alpha values is primarily driving the image gradient, $\nabla F = \nabla B = 0$ is a better assumption to make. In areas where α is 0 or 1, or where there is smooth transparency, $\nabla \alpha = 0$ is the better assumption. In order to apply each of these assumptions where they are effective, we could instead make the smoothness

assumption

$$\nabla I = \begin{cases} \alpha \nabla F & \text{if } \nabla F \mapsto \nabla I \\ (1-\alpha)\nabla B & \text{if } \nabla B \mapsto \nabla I \\ (F-B)\nabla \alpha & \text{if } \nabla \alpha \mapsto \nabla I \end{cases} \tag{5.8}$$

where $\nabla F \mapsto \nabla I$ indicates that the gradient at I is due primarily to the gradient in F. This leads to a term that encourages smoothness in foreground, background, and alpha where there should be smoothness, and encourages the gradient to be proportional to the image where it should be proportional:

$$V_1(p,q) = \begin{cases} \sum_{j \in (r,g,b)} g_j^F(p,q) & \text{if } \nabla F \mapsto \nabla I \\ \sum_{j \in (r,g,b)} g_j^B(p,q) & \text{if } \nabla B \mapsto \nabla I \\ \sum_{j \in (r,g,b)} g_j^\alpha(p,q) & \text{if } \nabla \alpha \mapsto \nabla I \end{cases} \tag{5.9}$$

where (using the notation from Equation 5.5)

$$g_j^F(p,q) = (\Delta_{pq} I_j - \alpha(p)\Delta_{pq} F_j)^2 + (\Delta_{pq} B_j)^2 + (\Delta_{pq}\alpha)^2 \tag{5.10}$$

$$g_j^B(p,q) = (\Delta_{pq} I_j - (1-\alpha(p))\Delta_{pq} B_j)^2 + (\Delta_{pq} F_j)^2 + (\Delta_{pq}\alpha)^2 \tag{5.11}$$

$$g_j^\alpha(p,q) = (\Delta_{pq} I_j - (F_j(p) - B_j(p))\Delta_{pq}\alpha)^2 + (\Delta_{pq} F_j)^2 + (\Delta_{pq} B_j)^2 \tag{5.12}$$

Equations 5.10, 5.11, and 5.12 are just discretizations of Equation 5.4 encouraging $\nabla B = \nabla \alpha = 0$, $\nabla F = \nabla \alpha = 0$, and $\nabla F = \nabla B = 0$, respectively.

To decide which of the gradients is causing the image gradient, we use the alpha values at pixels p and q. For some threshold T, if $\alpha > T$ for both p and q, then we assume $\nabla F \mapsto \nabla I$. If $\alpha < (1-T)$ for both p and q, then we assume $\nabla B \mapsto \nabla I$. Otherwise, we assume $\nabla \alpha \mapsto \nabla I$.

5.4.5 Color

The color term encourages the foreground and background colors to be similar to the colors in the known foreground and background areas of the image respectively. This is achieved by imposing a cost on the foreground (background) color based on the squared distance in color space to the nearest pixel in the foreground (background) in a combined color and position space,

$$U_1(p) = ||F(p) - I^{\mathcal{F}}_{near}(p)||^2 + ||B(p) - I^{\mathcal{B}}_{near}(p)||^2 \tag{5.13}$$

where $I^{\mathcal{F}}_{near}(p)$ is the color of the nearest pixel to p in the known foreground region of I and $I^{\mathcal{B}}_{near}(p)$ the nearest pixel in the known background. To compute the nearest pixel, we calculate a distance for the foreground and background pixels respectively to all other possible x,y,r,g,b locations using the Maurer distance transform [45].

5.4.6 Sparsity in alpha

In most images, the number of mixed pixels is far less than the number of pixels that belong completely to foreground or background (pixels whose $\alpha = 0$ or 1). Because of this, it is reasonable to bias pixels toward having alpha values of 0 or 1. Our term to bias alpha toward 0 or 1 is given by

$$U_2(p) = \begin{cases} (1-\alpha)h(I,p,B,F) & \text{if } h(I,p,F,B) \leq T_h \\ \alpha h(I,p,F,B) & \text{if } h(I,p,B,F) \leq T_h \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

where

$$h(I,p,F,B) = \frac{dist(I,p,F)}{dist(I,p,F) + dist(I,p,B)} \tag{5.15}$$

$dist(I,p,F) = ||I(p) - I^{\mathcal{F}}_{near}(p)||^2$ and T_h is a threshold. Here we use a value of $T_h = 0.01$. This term introduces the assumption that if the color models indicate that the color at a pixel is distinctly foreground or background only (if it has a low squared distance to foreground

and high to background, or vice versa), then that pixel likely has an alpha value near 1 or 0 respectively.

5.4.7 Optimization method

Many previous matting algorithms have cost functions that are specifically designed to be easy to minimize. In our formulation, we are more concerned with including the best terms possible to allow for the simultaneous computation of the foreground, background, and alpha. Unfortunately, this approach results in a difficult function to optimize.

We use gradient descent in order to minimize Equation 5.2. Because our objective function is not convex, we can easily fall into local minima while optimizing the energy function. To address this, we add a momentum term to help overcome shallow local minima. We also borrow an idea from graduated non-convexity [8]. These methods attempt to minimize over a non-convex objective function by smoothing the function such that it is (more) convex, finding a minimum, and then iteratively using that minimum to initialize a less smooth form of the function. We achieve a similar end by changing the matting equation weight λ_0 in order to allow for a smoother function initially before converging on the weighted terms we desire. Although the matting equation term (Equation 5.3) is extremely important to finding the correct solution, it may be minimized to 0 at infinitely many values of F, B, and α, and so produces many local minima. By reducing λ_0 near the beginning of the gradient descent, we can more easily avoid local minima created by this term. Specifically, we use $\lambda_0 = 0.25$ initially and increment it in steps until $\lambda_0 = 4$. We fix the other parameters at $\lambda_1 = 1$, $\lambda_2 = 0.4$, $\gamma_0 = 0.2$, $\gamma_1 = 0.8$. We also increase the threshold T used for computing the primary gradient in Equation 5.9. We use a $T = 0.6$ initially, and increment it until $T = 0.99$. The lower initial value helps better compute F and B, and the later higher value enforces more smoothness in α once the foreground and background are better estimated. Note that we constrain F, B, and α to the range $[0, 1]$ at each iteration.

To initialize the gradient descent, we compute an initial estimate of the foreground, background, and alpha values. We initialize F and B by sampling the known regions similar to [81]. We initialize α by computing the ratio of the distance to the nearest foreground and background color for a given pixel as computed by Equation 5.15 ($\alpha = 1 - h(I, p, F, B)$). This distance metric is similar in spirit to those in [4, 26].

5.5 Results

We evaluate our matting technique using the two datasets introduced in [58]. The first dataset, which we will refer to as the public dataset, consists of 27 input images and trimaps, and has the ground truth alpha mattes and ground truth foreground images publicly available. The second dataset, the private dataset, consist of 8 images which are overall more difficult than the public dataset, and their accompanying trimaps only. The foreground, background, and alpha ground truth images of the private dataset are not available, although one may submit their alpha mattes to www.alphamatting.com to obtain an error score over four different metrics.

5.5.1 Alpha accuracy

Table 5.1 shows the error in α on the public dataset from [58] for our algorithm, Closed-Form Matting [38] using their publicly-available code, and Robust Matting [81] using our own implementation. The mean absolute difference (MAD), mean squared error (MSE), and gradient error (which detects errors in the gradient and is correlated to what humans visually perceive as correct as explained in [58]) averaged over all images in the dataset are shown. Our algorithm outperformed Closed-Form Matting and Robust Matting on all three measures. We also achieved a lower gradient error than both Closed-Form Matting and Robust Matting on all examples individually.

Tables 5.2 and 5.3 show our results on the private dataset for the sum of absolute difference and the mean squared error respectively as computed by submitting our results

Algorithm	MAD	MSE	Gradient
Our Method	0.017	0.0066	0.016
Closed-Form Matting [38]	0.019	0.0081	0.035
Robust Matting [81]	0.037	0.0177	0.032

Table 5.1: Average error in α on public set from [58] for our method, Closed-Form Matting [38], and Robust Matting[81].

Algorithm	Rank	Troll		Doll		Donkey		Elephant		Plant		Pineapple		Bag		Net	
	(of 11)	small	large	small	large	small	large	small	large	small	large	small	large	small	large	small	large
Our Method	3.8	16.9	27.1	10.4	14.2	6.5	8.0	3.4	8.3	6.4	8.9	6.8	11.0	25.9	28.3	40.7	51.2
Rhemann [56]	1.9	14.9	24.5	6.7	9.5	4.6	6.1	2.6	5.4	7.5	9.9	6	10.1	26.1	26.7	23.8	25.6
Closed-Form [38]	2.8	12.7	21.9	5.9	8.5	4.6	6.1	2.2	4.6	9.3	12.1	8.3	14.9	34.2	32.4	26.5	25.7
Robust [81]	3.8	17.3	28.4	10.1	16.9	4.8	6.5	2.8	7.3	7.3	14	6.8	14.6	22.7	26.1	34.4	37
High-Res [57]	4.5	18.6	25.8	8.6	14.1	5.0	6.2	2.5	8.3	7.8	14	8.5	18.1	35.3	38.1	38.7	54.6
Random Walk [26]	6.5	17.9	20.3	11.3	15.6	5.8	7	3.4	6.7	13.1	22.1	12.3	18	44.1	43.5	75.1	81.8
Geodesic [4]	7.3	26.9	38.5	14.2	16.5	11.7	14	7.6	15.1	12.8	16.7	7.3	12.1	37.3	37.4	48.6	50

Table 5.2: Sum of absolute difference on private dataset from [58]. The table includes the top seven of eleven algorithms and data from two of the three trimaps. Iterative BP [82], Easy Matting [28], Bayesian Matting [14], and Poisson Matting [71] finish out the rankings.

to www.alphamatting.com. For size reasons, only a subset of the tables are shown. On this more difficult dataset, we perform comparable to the current best algorithms in generating an alpha matte, while only one of these algorithms, [38], computes foreground and background colors (and does so as a postprocess). Note the magnitudes of the values in Tables 5.2 and 5.3 differ from those in Table 5.1 because they are scaled at www.alphamatting.com for presentation purposes.

A comparison of our technique to several others is shown in Figure 5.1 on the "Plant" example from the private dataset. Our algorithm is able to cleanly matte the leaves while maintaining the "holes" through which the background shows. Additional examples of alpha mattes generated by our algorithm are shown in Figure 5.3.

Algorithm	Rank	Troll		Doll		Donkey		Elephant		Plant		Pineapple		Bag		Net	
	(of 11)	small	large	small	large	small	large	small	large	small	large	small	large	small	large	small	large
Our Method	4.1	0.9	2.6	0.8	1.2	0.5	0.7	0.2	0.7	0.5	0.7	0.6	1.0	1.9	2.1	3.3	4.5
Rhemann[56]	1.9	0.8	2.4	0.3	0.5	0.3	0.4	0.1	0.3	0.7	0.7	0.4	0.7	2.0	1.9	1.3	1.5
Closed-Form[38]	3.1	0.5	1.8	0.3	0.4	0.3	0.4	0.1	0.3	1.2	1.4	0.8	1.6	3.0	2.7	1.3	1.2
Robust[81]	3.5	1.1	2.8	0.7	1.5	0.3	0.4	0.1	0.5	0.5	1.2	0.5	1.5	1.5	1.8	2.4	2.3
High-Res[57]	4.2	1.2	2.2	0.5	1.1	0.3	0.4	0.1	0.7	0.6	1.2	0.8	2.0	3.2	3.4	2.6	4.3
Iterative BP [82]	6.4	1.7	2.6	1.5	2.6	0.5	0.7	0.2	0.8	1.1	2	1	2	2.8	3.3	3	3.8
Random Walk [26]	6.7	1	1.1	1	1.7	0.5	0.6	0.2	0.4	2	3.4	1.6	2.3	4.6	4.4	8.3	9.4

Table 5.3: Mean squared error on private dataset from [58]. The table includes the top seven of eleven algorithms and data from two of the three trimaps. Geodesic Matting [4], Bayesian Matting [14], Easy Matting [28], and Poisson Matting [71] finish out the rankings.

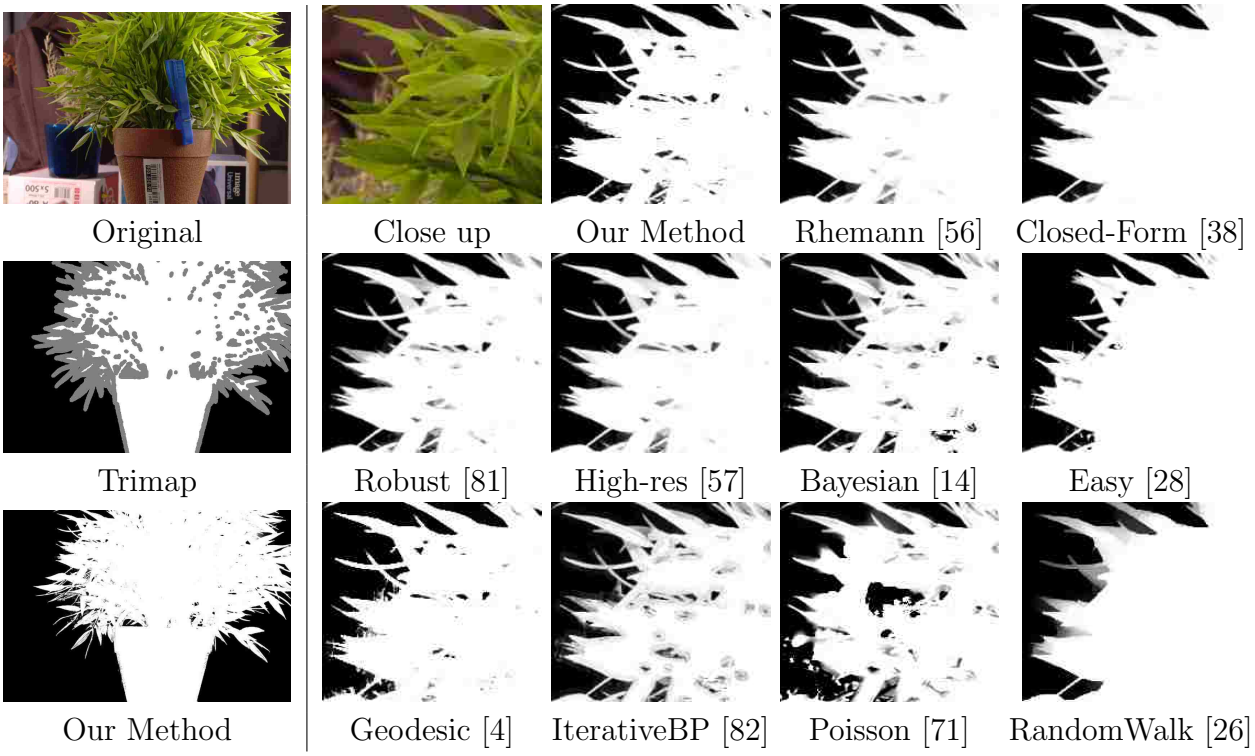


Figure 5.1: The left column shows our results on the "Plant" example from [58]. The remaining columns show a comparison of our technique to several others on a region of this image.

5.5.2 Foreground/background accuracy

A particular focus of our algorithm is achieving high accuracy for the foreground and background at each unknown pixel. To determine accuracy of our foreground estimation, we compare our method to several others using the true foreground colors of the public dataset in [58]. Because the foreground colors were only provided in a raw 16-bit format without gamma correction and white balance, we manually set the gamma and white balance of the foreground images and their corresponding raw 16-bit input images to produce input images similar in color appearance to the standard input images of the public dataset.

We compare our foreground colors to those generated simultaneously with alpha by Bayesian Matting [14], those generated as a postprocess by Closed-Form Matting [38], and those generated as an initialization to Robust Matting [81]. The error is computed by taking

Algorithm	MAD	MSE
Our Method	0.063	0.005
Closed-Form Matting [38]	0.082	0.016
Bayesian Matting [14]	0.11	0.02
Robust Matting [81]	0.16	0.031

Table 5.4: Foreground alpha product (αF) error over public dataset from [58]. The mean absolute difference (MAD) and mean squared error (MSE) are shown.

the difference of the foreground color multiplied by the ground truth α, since pixels with a high α are more important visually when compositing.

The results are shown in Table 5.4. The MAD and MSE averaged over all 27 images are shown for each algorithm. Our algorithm outperformed all other methods on 18 of the 27 images using the MAD as a metric, and 20 of the 27 using MSE.

Several examples of foregrounds and backgrounds generated by our algorithm are shown in Figure 5.2 in comparison to those generated by Closed-Form Matting [38]. The regions being shown in each close-up are indicated by red and blue boxes corresponding to foreground and background respectively. In each example, the foreground is multiplied by α and the background by $(1 - \alpha)$ using the α computed by the respective algorithm. The foreground and background colors produced by our method appear more plausible, confirming the quantitative results in Table 5.4. For example, this is easily seen in the top example of a plant, where the leaves of the plant appear blue and the spaces in between the leaves appears green in the Closed-Form Matting results, but appear more correct in our results. Additional examples are shown in Figure 5.3.

Unfortunately, only the true foreground colors from the public dataset of [58] are available, so a quantitative evaluation of this dataset cannot be performed for the background colors. Based on the visual appearance, we predict that the accuracy of the background would be similar to that of the foreground for each of these algorithms. For example, note how in Figure 5.2 our method appears to better reconstruct the occluded backgrounds.

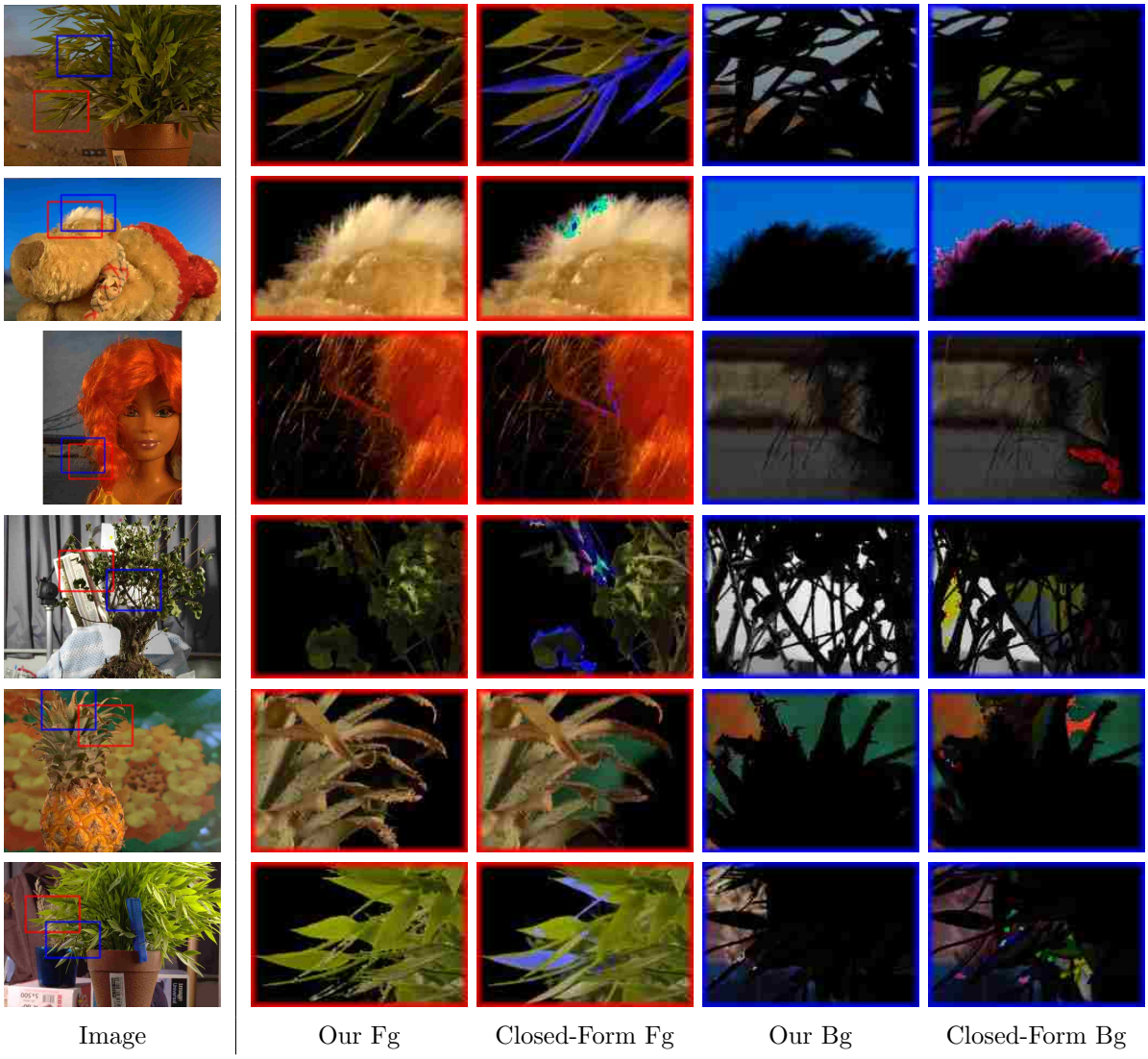


Figure 5.2: Comparison of the foreground and background generated by our method and Closed-Form Matting [38]. The foreground is multiplied by α and the background by $(1-\alpha)$ using the α computed by the respective method. The red and blue boxes in the original images indicate the close-up areas for the foreground and background respectively.

5.6 Conclusion

We have introduced a new method for computing a solution to the full matting equation for a given image by simultaneously estimating the foreground, background, and alpha at each unknown pixel. We do so by including additional terms to affect the foreground, background, and alpha values, formulating the terms into a single energy equation, and minimizing that equation using gradient descent with momentum and a partial form of

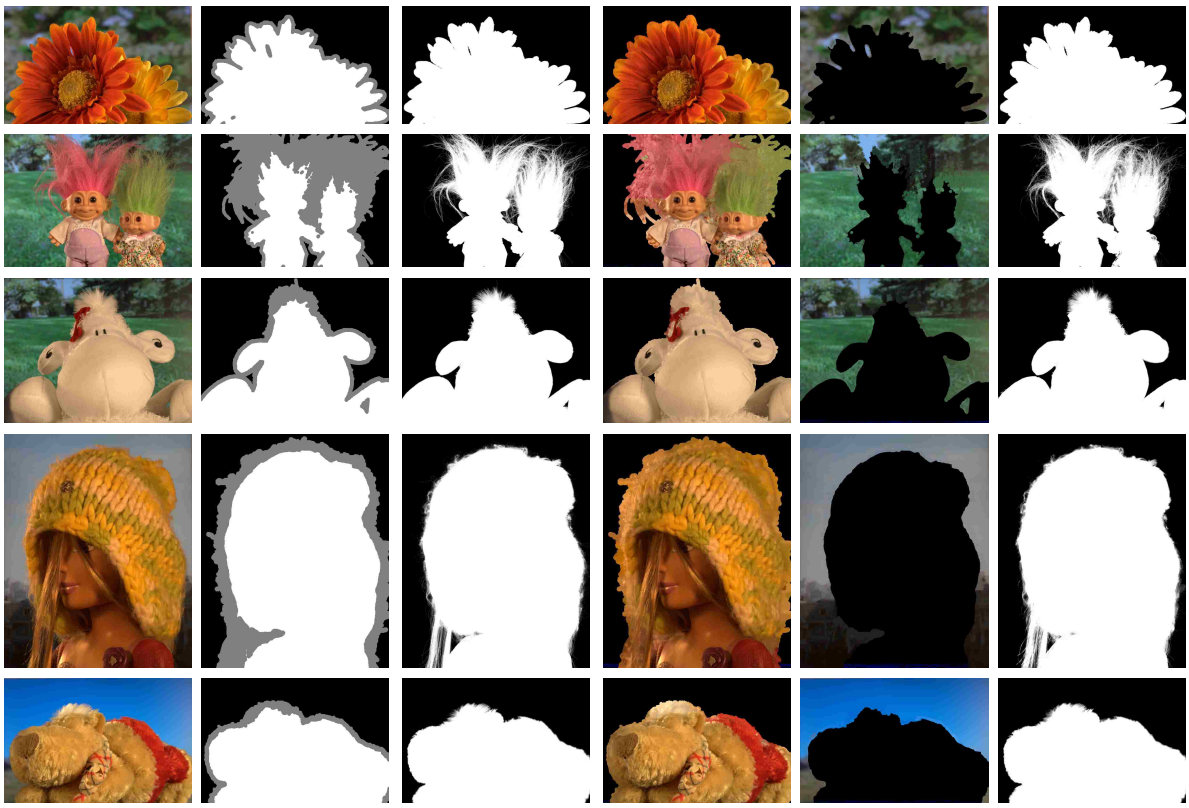


Figure 5.3: The foreground, background, and alpha generated by our algorithm.

graduated non-convexity. This focus on estimating not just alpha but also the foreground and background allows us to optimize over explicit constraints on the foreground and background, which most current algorithms that optimize only over alpha cannot do. Our results are comparable to the best current algorithms for computing the alpha matte and show superior performance in computing the foreground colors in all tests performed.

While this work improves foreground and background estimation for mattes and provides competitive results in computing alpha, our energy function is complex and difficult to optimize. Future work could improve this by incorporating more advanced optimization techniques or by simplifying the terms to allow for easier minimization. Additional or improved terms could also be added to potentially improve results.

Chapter 6

Video Segmentation

Chapter 2 presents an initial version of our video segmentation system. This chapter presents an extended and expanded version of that system and explores the system more fully. There are three major additions and contributions of this chapter over that of Chapter 2.

First, the explanation and motivation for the different cues is expanded. More details are given and several new figures provide increased understanding. Some minor changes to some of the cues improve their performance. A new cue is also introduced, a color model whose scope is global both spatially and temporally.

Second, this chapter better explores the process of learning from multiple cues using user interaction. Several new learning approaches are proposed and explained. The new learning process include the delta function, a prediction by expert advice learning algorithm, and two different probabilistic methods.

Finally, the validation of our system is extended. A test set comprised of many video sequences and ground-truth object selections from two sources is introduced. The effectiveness of the different learning algorithms is tested on this dataset. The impact of each cue on the final segmentation results is also examined using this dataset. Further analysis of the overall system and new results are given.

6.1 Abstract

Video sequences contain many cues that may be used to segment objects in them, such as color, gradient, color adjacency, shape, temporal coherence, camera and object motion, and

easily-identifiable points. This paper introduces a novel method for interactively selecting objects in video sequences by extracting and leveraging as much of this information as possible. Using a graph cut optimization framework, this method propagates the selection forward frame by frame, allowing the user to correct any mistakes along the way if needed. Enhanced methods of extracting many of the features are provided. In order to use the most accurate information from the various potentially-conflicting features, the effectiveness of each feature is automatically learned and updated based on user corrections. Several potential learning algorithms are explored and evaluated. The effectiveness of our system is shown through timing comparisons to other interactive methods, accuracy comparisons to unsupervised methods, and qualitatively through selections on various video sequences.

6.2 Introduction

Video segmentation is an essential process in many video applications. It is required for video editing and special effects whenever objects must be moved, deleted, individually edited, or layered. It is also used in object recognition, 3D reconstruction from video, and compression. Despite recent research in the area, industry still largely relies on chroma keying and manual rotoscoping, emphasizing the need for an effective, easy-to-use video segmentation tool.

This need remains due to the surprising difficulty of the problem. Video segmentation shares the difficulties of image and volume segmentation, such as overlapping color distributions, weak or blurred edges, complex textures, and compression artifacts. In addition to these challenges, video segmentation suffers from many other common problems. A video sequence may contain erratic camera and/or object movement and motion blur. Objects may move enough that there is no overlap between successive frames. Other moving objects may cause confusion. Lighting changes and shadows alter the color distributions, and movements in 3D space may greatly change an object's 2D projected boundary. Occlusions may temporarily hide portions of the object. A given video sequence can easily exhibit many of these challenges.

Many different kinds of information can be gleaned from successive video frames to aid object selection. Such features include color, gradient, adjacent color relationships, shape, spatiotemporal coherence, camera motion, object motion, and trackable points. The relative importance of the cues differs depending on the sequence, the frame, and even the location in the frame. For example, in Figure 6.1 a color model can easily distinguish the cat from the light brown floor but would struggle separating the tail from the similarly-colored bag. A shape feature, however, could separate the tail and bag. An algorithm that intelligently applies all of these cues based on specific circumstances will perform better than one relying only on a subset of these cues or on a static combination of them.

Despite the importance of each kind of information, most current algorithms do not use all these features. Algorithms that segment the video as a spatiotemporal volume [3, 4, 9, 80] can generally only extract information from the pixels under the user strokes to model the foreground and background. These methods have no information about some of these features such as shape or boundary information, and have limited knowledge of other features such as foreground and background color. By allowing the user to segment one frame and then propagating this information to other frames, these features can be used.

In this paper, we introduce LIVEcut, a frame-by-frame interactive video segmentation method designed to maximize the information propagated from one frame to the next. As shown in Figure 6.1, LIVEcut extracts various features and resolves them using graph-cut optimization. LIVEcut also learns automatically from user corrections how well each cue performed and weights their importance accordingly. Our local weighting allows LIVEcut to selectively apply the cues that will most effectively segment the object. Contributions are also made in the extraction of many of the individual cues. These include full foreground and background local color models, color adjacency models, separate foreground and background motion models, point tracking information, and a new shape prior.

The paper is organized as follows. Section 6.3 describes related work. Section 6.4 describes the basic interaction and framework of our system. An analysis of each informative

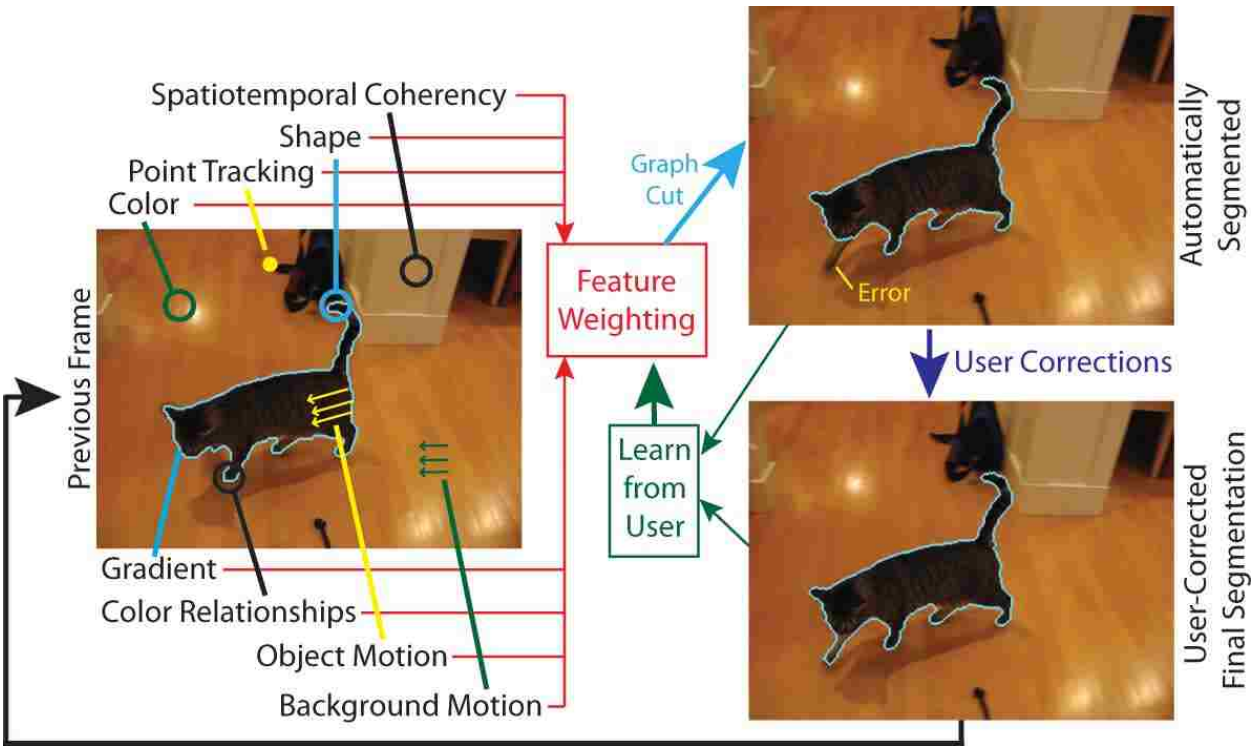


Figure 6.1: From an initial segmented frame, a variety of features are extracted and used to segment the next frame. If errors occur, the user may correct them, and the system learns which features are providing good information, and uses this information to weight the cues for proceeding frames. The corrected frame is used to continue propagating the segmentation.

cue used in presented in Section 6.5. This is followed by an explanation of how to automatically weight the different cues by learning from user corrections in Section 6.6. Section 6.7 gives our results, and our conclusion and future work is given in Section 6.8.

6.3 Related Work

Many approaches have been taken in interactive video segmentation. Some approaches focus on either boundary or region information only. Agarwala et al. [1] performs boundary tracking using splines that follow object boundaries between keyframes using both boundary color and shape-preserving terms. Snakes [32] also track the boundary by minimizing an energy term over the boundary curvature and image information. Bai and Sapiro [4] use region

color to compute a geodesic distance to each pixel to form a selection. These approaches perform well when a single type of cue is sufficient for selecting the desired object.

Many current techniques use graph cut to segment the video as a spatiotemporal volume. Graph cut, as formulated in [9], solves for a segmentation by minimizing an energy function over a combination of both region and boundary terms. It has been shown to be effective in the segmentation of images [40, 60] and volumes [3, 86].

A basic approach to segmenting video as a spatiotemporal volume was given in [9]. The graph connects pixels in a volume, which implicitly includes spatiotemporal coherence information. Graph cut is applied using a region term based on a color model of the pixels under the user strokes and a boundary term based on gradient.

Wang et al. [80] builds on this approach by allowing users to segment video by drawing strokes on arbitrary slices of the spatiotemporal volume. While this permits a user to mark several frames at once, it requires a steep learning curve to know how to carve the volume so that the right pixels are visible along the slice. The method uses a global color model based on the user strokes as well as a local color model for static backgrounds in addition to gradient values.

In Li et al. [41], users segment every tenth frame, and graph cut computes the selection between the frames using global color models from the key-frames, gradient, and coherence as its primary cues. The user may also manually indicate areas to which local color models are applied. While this method performs well, it requires the manual segmentation of many frames in addition to corrections.

In methods where the video is treated as a spatiotemporal volume [3, 4, 9, 80], the only information known for certain about the object and background are in the user-marked pixels. This provides very limited knowledge about the object interior and no knowledge about the boundary. While [41] is an exception to this, it requires the user to manually segment many frames. These methods contrast our own, where frame-by-frame propagation allows for the computation of complete features. Frame-by-frame propagation also provides

an interactive paradigm of moving through the video sequentially, which is arguably the most natural for video.

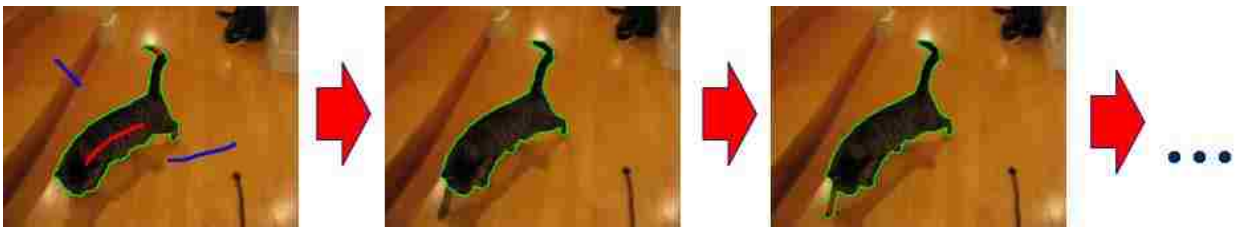
In parallel with our own work, Yin and Collins [85] proposed an automated video segmentation system that includes color, gradient, color adjacency, and shape information in a graph cut framework. They dynamically reweight these terms from frame to frame, but do so on a global basis without regard to user corrections.

Another recent work, Video Snapcut [5], segments a video using a frame-by-frame propagation method similar to our own. They select an object by dividing the boundary into overlapping portions and individually tracking each with a SIFT tracker. Region color and shape information are then used as part of a graph-cut optimization to produce a segmentation.

Some unsupervised video segmentation methods have also combined various cues [17, 73, 84]. While unsupervised techniques generally perform well at roughly separating motion layers, they do not produce the high-quality results required for many applications. The object of interest may also not correspond to a motion layer, leaving these methods incapable of generating the desired result.

6.4 Video Segmentation Framework

While the methods described in Section 6.3 provide good means of segmenting video, each relies only on a few cues to make decisions. LIVEcut extracts much more information about the sequence and uses this to improve the segmentation. The cues are combined in a graph-cut optimization to compute the object selection for the current frame. As the user proceeds through the sequence, the implicit verification of the previous frame allows LIVEcut to use the entire previous frame once again to segment the current frame.



Previous frame Next frame with error Corrected frame

Figure 6.2: Frame-by-frame interaction. The user selects the object in the first frame (left) by placing foreground strokes (red) and background strokes (blue). The boundary of the object is highlighted in green. The selection is then propagated to the following frame. If there are any mistakes, such as the front leg of the cat (middle), the user may correct it (right) before proceeding to the next frame of the sequence.

6.4.1 Interaction

We utilize a frame-by-frame interactive framework for segmenting video sequences as illustrated in Figure 6.2. The user begins the selection by marking the object in the first frame of the sequence (Figure 6.2 left) by placing foreground and background strokes as in most graph-cut segmentation methods [9, 40, 41, 80, 5, 3, 86]. LIVEcut then propagates various cues computed using the full frame to the next frame. If any mistakes occur such as the missing front leg in Figure 6.2 middle, the user may correct the error (Figure 6.2 right) before advancing the selection to the next frame.

The use of a frame-by-frame propagation method provides several advantages over other techniques. In methods where the sequence is segmented as a spatiotemporal volume [80], the only information known to the algorithm for certain about the foreground and background are the pixels located directly below the user stroke. This provides very limited knowledge about the object interior and no knowledge about the boundary characteristics. The interactive paradigm of moving through the video sequentially not only is the most natural for video, but allows the complete characteristics of each frame to be used in segmenting the next. While this is also true of keyframe-based systems such as [41], these require the user to completely segment many frames instead of allowing the algorithm to quickly compute this information for the user.

6.4.2 Graph cut framework

Before explaining the specific features we propagate from frame to frame, we present the overall framework in which the features are resolved. For this, we use minimum graph-cut optimization. Graph cut computes a segmentation over a set of pixels P by minimizing the equation

$$E(\mathcal{L}) = \sum_{x_i \in P} R(x_i, \mathcal{L}_i) + \lambda \sum_{(x_i, x_j) \in N} B(x_i, x_j) \ |\mathcal{L}_i - \mathcal{L}_j| \tag{6.1}$$

where $\mathcal{L} = (\mathcal{L}_i)$ is a binary vector of labels and $\mathcal{L}_i$ is the label (0 for background, 1 for foreground) for pixel x_i, $R(x_i, l)$ is a region cost term based on the label l, $B(x_i, x_j)$ is a boundary cost term, λ is a relative weighting of R and B, and N is the set of pairs of neighboring pixels.

Our region term $R(x_i, l)$ is the sum of all cues that apply to an individual pixel. Given a set of unary cues U,

$$R(x_i, l) = s(x_i, l) + \sum_{u \in U} w_u(x_i) \ R_u(x_i, l) \tag{6.2}$$

where $R_u(x_i, l)$ is the cost of labeling pixel x_i with label l according to cue u, $w_u(x_i)$ is a scalar weight giving the certainty of cue u at pixel x_i as set by our learning algorithm (Section 6.6), and $s(x_i, l) = 0$ if the pixel was labeled l by a user stroke or not labeled by the user and ∞ if labeled $\bar{l}$ (the other label) by the user.

Our boundary term $B(x_i, x_j)$ is given by

$$B(x_i, x_j) = B_a(x_i, x_j) \ B_g(x_i, x_j) \tag{6.3}$$

and encourages selection boundaries in the current frame to occur at image edges with color profiles similar to the selection boundaries in the previous frame. The unary terms (local color R_c, global color R_{gc}, spatiotemporal coherency R_h, shape R_s, and point tracking R_p) and binary terms (gradient B_g and color adjacency B_a) are defined in Sections 6.5.2–6.5.8.

6.4.3 Oversegmentation Regions

Graph-cut optimization can be somewhat slow for interactive applications when operating on pixels. In order to increase the speed of the algorithm, we apply our algorithm to an oversegmentation of the image. Similar techniques have been used to speed up graph-cut-based algorithms before [40, 41, 80]. While these methods use a single layer of oversegmentation regions to accelerate the segmentation, [3] instead uses a multi-layered hierarchy of regions computed using the tobogganing algorithm of [22] to produce an initial fine layer of oversegmenation regions that are then accumulated into larger regions to form a new coarser layer of the hierarchy using the technique from [55]. These regions can themselves be accumulated into even larger regions which builds a new layer of the region hierarchy, and this process can continue until only one region exists.

The advantage of coarser levels of the hierarchy is that fewer regions exist and algorithms such as graph-cut optimization can process them much quicker. However, at some point coarse regions no longer represent homogeneous regions of the image as desired. In our method, we use the coarse level of regions created by aggregating the initial fine regions produced by [22]. This allows for fewer regions than the finest level with the individual regions still maintaining a reasonable degree of homogeneity. We then refine the segmentation on the pixel level, using only those pixels that belong to oversegmentation regions bordering the coarse-level boundary.

For ease of explanation, the following terms used in our graph-cut optimization are all defined according to pixels. However, they can all be directly extended to oversegmented regions.

6.5 Cues

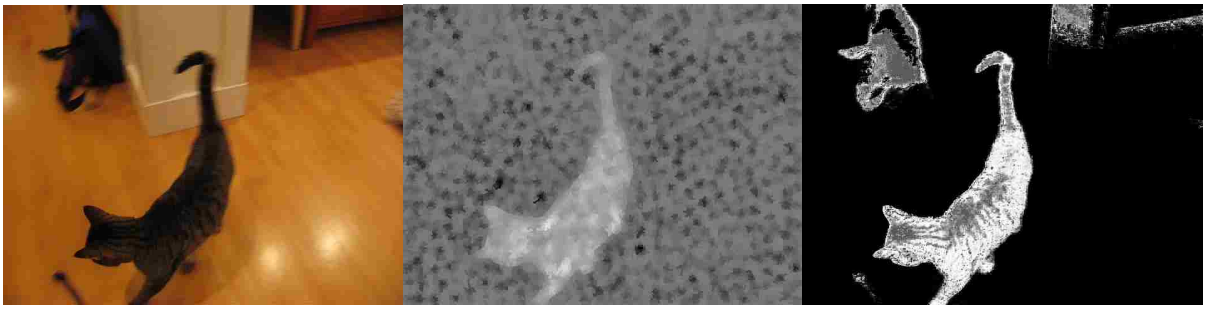
Many different pieces of information exist in video sequences that can be used to aid in selecting objects. In this section, we describe the various cues our system utilizes. These

cues will be applied on each frame using the graph-cut framework of 6.4.2 and weighted adaptively using the learning framework of 6.6.

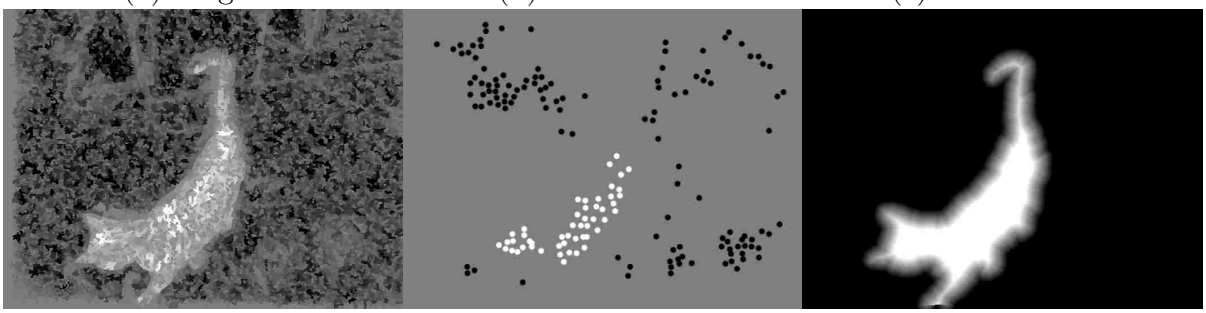
In choosing cues, it is worthwhile to look at the different types of cues possible. The graph-cut framework necessarily divides possible cues into region-based and boundary-based terms. This necessitates that some cues are applied directly to individual pixels to indicate whether they are foreground or background, while others operate on pairs of pixels to define their similarity. Additionally, the possibility exists of using cues to influence the performance of other cues rather than directly incorporating them as a term in the graph-cut optimization.

Another way to differentiate types of cues is by their scope over the video sequence, or the temporal scope over which the cues gather information. In a global scope, the cue would look at all preceding frames in order to determine the selection in the current frame. Since the user implicitly validates the segmentation of one frame before moving onto the next, we have the entire segmentation from all previous frames to use in computing the current selection. However, video objects and backgrounds can change significantly over the course of a video sequence, causing the global information to quickly become outdated and possibly causing it to hurt performance. In a local scope, just the previous frame (or small number of previous frames) would be used to create the cue. This assumes that the current frame to be segmented is entirely a product of the previous frame and treats the video sequence like a Markov chain. While the previous frame should contain most of the information needed about the object, in many cases such as where occlusions occur objects may disappear for a time before reappearing, which may require user interaction to select the part of the object that reappeared if more global information is not available. We use cues that are both local and global in scope over the video sequence.

The specific cues that we use in our system will be presented as follows. We first present object and background motion, which are used to adjust the locality information of the following cues rather than being directly applied as a term in the graph-cut optimization. We then present the region-based cues (color, global color, shape, spatiotemporal coherency,



(a) Original (b) Local Color (c) Global Color



(d) Temporal coherence (e) Point tracking (f) Shape

(g) Gradient (h) Color Adjacency

Figure 6.3: Visualization of the graph-cut terms for a frame from the "cat" sequence. For the region cues (top two rows), white indicates foreground likelihood (meaning the cost of labeling the pixel foreground according to the given cue is low and the cost of labeling background is high), black background likelihood, and mid-gray neutral. For the boundary cues (bottom row) white indicates an object boundary and black indicates no boundary.

and point tracking). The boundary-based cues (gradient and color adjacency) are then presented. Each of these cues is local in scope temporally except for the "global color" cue, which is global in scope temporally as suggested by its name. Figure 6.3 visualizes the effect of many of the cues.

6.5.1 Object and Background Motion

Motion is an important cue in video segmentation. By considering the motion of the object, more precise local information may be used to segment it. By removing camera motion, better local information can be used for the background where it is static.

Many methods account for the camera motion by aligning the frames in a preprocessing step [41, 80]. However, since the foreground object will often exhibit different motion patterns than the background, aligning the background will not correctly align the foreground and the object motion can potentially interfere with the background alignment.

Since we know the segmentation of the previous frame, we can align the foreground and background separately. To align the background, we assume that the majority of the motion is caused by camera movement. The background is aligned by locating good points to track [65], then computing and applying a homography.

While the foreground can be tracked in the same manner, problems can occur if the foreground does not have enough trackable points to generate a good homography due to large movements or little texture. To account for these cases, we use a novel method to roughly align the foreground.

We use an iterative-closest-point-style algorithm [6] to match pixels x_i in the selection M on the current frame I to pixels y_j in the next frame I_{next} with one affine transformation A. The iteration alternates between two steps. First, we use the rigid transformation A to map the points in M forward the next frame and then for each point search that area for a better corresponding pixel. Second, we recompute A to align it better to the new correspondences.

More precisely, in the first step we find the best matches $\{(x_i, y_{m(i)})\}$ for a given A. We match points in (xy position $\times$ RGB color) space so that points in M are matched to points in I_{next} that are similar in color and position after applying A. For each $x_i \in M$, we

solve a nearest neighbor problem ([49])

$$y_{m(i)} = \arg\min_{y_j} ||(Ax_i, \gamma I(x_i)) - (y_j, \gamma I_{\text{next}}(y_j))||_2^2 \tag{6.4}$$

where RGB values are in $[0, 1]$ and γ is the sum of the frame width and height. This results in a new corresponding pixel $y_{m(i)}$ for each x_i that matches more closely in terms of color and location than the pixel corresponding to x_i after simply applying A.

In the second step, our method finds a new transformation A to best align matches $\{(x_i, y_{m(i)})\}$. To accomplish this, we solve

$$A = \arg\min_{A} \sum_{i=1}^{n} ||Ax_i - y_{m(i)}||_2^2 \tag{6.5}$$

as in [62]. This step ideally improves the alignment A of the points in M to the next image I_{next}.

We begin the iteration with the identity transformation, although perhaps a better starting value could be obtained by a prediction based on motion in previous frames. We run a maximum of 25 ICP iterations and declare that the algorithm has converged if the positions of all the transformed points do not move by more than 0.1 pixels. Our algorithm is robust enough that it can use a small subset of uniformly sampled points from M and still find a good affine transformation. We set the number of selection points to track to the maximum of 200 and 2% of the total points in M. An optional input of the largest allowable interframe motion allows us to reduce the set of potential matches in I_{next} to something less than the entire frame.

If the number of points in M that we track is m and the number of potential matching locations in I_{next} is n, then building the search kd-tree requires $O(n \log n)$ time and computing all the nearest neighbors in the first step requires $O(m \log n)$ time for each ICP iteration. For SD-size videos, our novel foreground alignment procedure typically runs in tenths of a second with the specified size subset of M and a maximum interframe motion of 50 pixels.

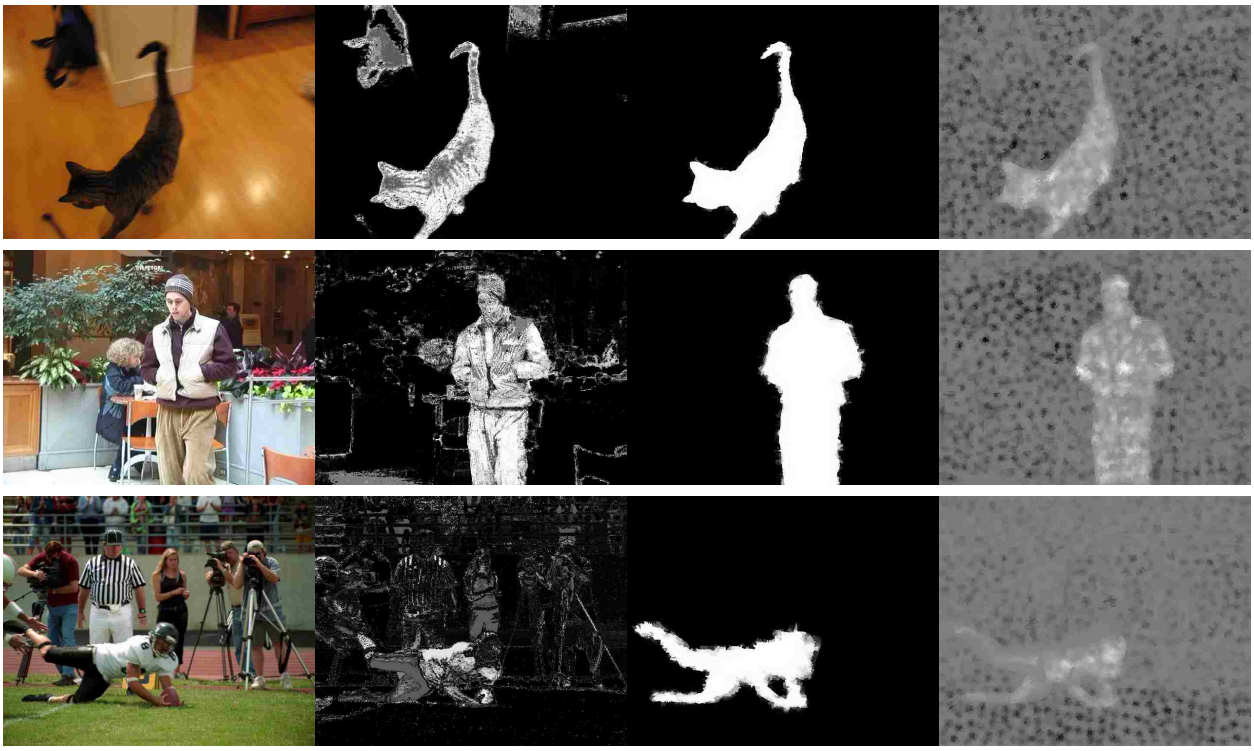
The object and background motions are not included as a term in graph cut. Rather, they are used to spatially transform the locality information of the other cues. While this transformation does not completely capture non-rigid motion, it improves the locality of the foreground information and works well in practice.

6.5.2 Color

A color-model region term encourages pixels to be labeled according to the color distribution of the model. Because most graph-cut algorithms [9, 40, 80] do not have access to a full segmentation of a frame, only the pixels under the user strokes are used to create the model. This limited sample does not always accurately represent the color properties of the image. These algorithms must also by necessity use a global color model, which does not differentiate colors located in different regions of the image. While [41] can use a local color model, it only does so over a small window if manually indicated by the user.

A contribution of LIVEcut is that it uses a *local* color model generated from the *entire* previous frame, which can distinguish between colors in different regions of the image. The effect of using a local color model is shown in Figure 6.4. The global (Figure 6.4(b)) and local (Figure 6.4(c)) color models applied to three images are illustrated, where white indicates a high likelihood of foreground, black a high likelihood of background, and the mid-gray uncertainty. Ideally the objects in these images would be white, and the background black, with very little mid-gray. However, because of the overlapping color models, the global color model shows less certainty for many of the pixels in the image and even suggests the wrong label for some pixels. The inclusion of position information allows the local color model to perform much better. For example, the bag and rope in the cat example are not indicated as foreground even though they are the same color as the cat. Our color model is also shown in Figure 6.3b in comparison to the other cues.

To model the color distributions, we estimate the likelihoods for each data point using a kernel density estimation computed by the Fast Gauss Transform [83]. The Fast Gauss



(a) Original (b) Global (c) Local (d) Local (scaled)

Figure 6.4: Global color model vs. local color model. For (a) a given image, the (b) global and (c) local color models for selecting an object are shown. White indicates foreground likelihood, black background, and mid-gray neutral. (c) shows the local color model computed using Equation 6.8 and (d) shows the local color model scaled by ϕ (Equation 6.10) as computed by Equation 6.11.

Transform computes the weighted sum of a set of Gaussians

$$G(p,l) = \sum_{j=1}^{N_l} q_j e^{-||p-y_j||^2/2h^2} \tag{6.6}$$

efficiently by expanding the sum into Hermite functions as explained in [83]. In Equation 6.6, p is a point for which we want a likelihood, y_i is a source data point from the previous frame with label l, N_l is the number of source points from the previous frame with label l, h is the bandwidth of the Gaussians, and q_i is a weighting coefficient. The weighted sum of Gaussians G represents the probability of an input p given a label l:

$$P(p|l) = G(p,l) \tag{6.7}$$

The local color model is generated by creating a (l, u, v, x, y) vector p_i for each pixel x_i in the previous frame (where (l, u, v) is the color and (x, y) is the motion-adjusted location). The probability of the pixel being foreground is then computed by a 5D Fast Gauss Transform [83] by

$$P(l|p_i) = \frac{P(p_i|l)}{P(p_i|l) + P(p_i|\bar{l})} \tag{6.8}$$

The probability is assigned to the cost term by

$$R_c(x_i, l) = \phi(x_i, l)P(\bar{l}|p_i)^2 \tag{6.9}$$

where ϕ is a function that scales the probability according to the confidence in the color models. The purpose of ϕ is to account for cases where the probabilities for both labels at a pixel x_i are very low but one probability is proportionally large compared to the other. For example, if $P(p_i|l) = 0.01$ and $P(p_i|\bar{l}) = 0.001$, the probability of x_i belonging to either label is very low, likely because a new color was introduced into the sequence. Despite $P(p_i|l)$ being very low, it is still 10 times the size of $P(p_i|\bar{l})$. Because of this, when the probabilities are normalized by the sum of both probabilities, the color model will indicate that the x_i almost certainly belongs to label l. To counteract this, ϕ scales the probabilities by a scaled sum of the probabilities

$$\phi(x_i, l) = \frac{(P(p_i|l) + P(p_i|\bar{l}))^2}{\max_{0<j<N, k\in\{l,\bar{l}\}} P(p_j|k)^2} \tag{6.10}$$

where N is the number of pixels. Combining Equations 6.8-6.10 and canceling terms gives

$$R_c(x_i, l) = \frac{P(p_i|\bar{l})^2}{\max_{0<j<N, k\in\{l,\bar{l}\}} P(p_j|k)^2} \tag{6.11}$$

The effect of scaling the color model by ϕ is shown in Figure 6.4(d), where much more gray is seen due to the increase in uncertainty caused by ϕ.

The degree of locality enforced by the color model can be altered based on relative scaling of the color information to the position information. We scale the chromaticity components (u and v) of the color to [0,1] and scale the intensity accordingly. We then scale the the image width or height (whichever is greater) to the range [0, r], with the other dimension scaled accordingly. By altering the value of r, we can change the locality of the color information. Smaller values of r cause the model to become less sensitive to spatial locality, while larger values of r cause the significance of the locality to increase. We use a fixed value of $r = 3$ in our formulation.

6.5.3 Global Color

In Section 6.5.2, we introduce a local color model that is generated from the previous frame. While this will account for the majority of the colors in most sequences, other colors may appear that have been occluded for several frames. In this case, a color model that is temporally global over the length of the sequence is needed to identify the new colors. Also, if the location of the object differs greatly from the previous frame, it may fall outside the locality of the same color from the previous frame. In this case, a color model that is global over the area of the frame is needed.

In our system, we include a color model that is global both temporally over the sequence and spatially over a frame to account for these cases. We record all previously-seen colors that are labeled l in a histogram H_l, and include this information in our graph-cut formulation using

$$R_{gc}(x_i, l) = \frac{H_l(C(x_i))}{H_l(C(x_i)) + H_{\bar{l}}(C(x_i))} \tag{6.12}$$

where $C(x_i)$ is the color at pixel x_i

6.5.4 Shape

When an object passes over a similarly colored background, no edge exists to indicate the boundary location. In these cases, the shape of the object is vital. Including a shape term in the features can handle such cases.

Recently there has been interest in including shape priors into graph cut [23, 35, 72, 77, 30]. The common approach is to align the shape to the image by user interaction and/or automated means, and then include a term in the cost function based on distance to the shape or a mismatch score.

In LIVEcut, because we have tracked the object motion forward, we already have an estimate of the motion-adjusted object shape Φ (where $\Phi(x_i) = 1$ if x_i is in the object mask and 0 otherwise) and its boundary Ω (where $\Omega = \partial\Phi$). We compute the distance from each pixel to the boundary after adjusting for object motion using

$$d_\Omega(x_i) = \min_{p \in \Omega}(||p - x_i||). \tag{6.13}$$

This distance is visualized in Figure 6.5. Our shape term is an extension to [77] but takes distance into account:

$$R_s(x_i, l) = |l - \Phi(x_i)| \min(d_\Omega(x_i)/M, 1) \tag{6.14}$$

where M is the maximum allowable distance (we use $M = 10$). If the estimated shape mask does not match the labeling of a pixel, this term penalizes the labeling based on the pixel's distance to the predicted shape boundary Ω. For example, in Figure 6.5, the pixel x_i is outside of the object shape Φ, and so this pixel will only occur a cost if labeled as foreground. Conversely, x_j will only occur a cost if labeled background since it is in Φ. Pixels are penalized based on the distance up to a maximum distance M. Using a small M, if the boundary is only off by a few pixels, it will have a minimal cost added. This cost

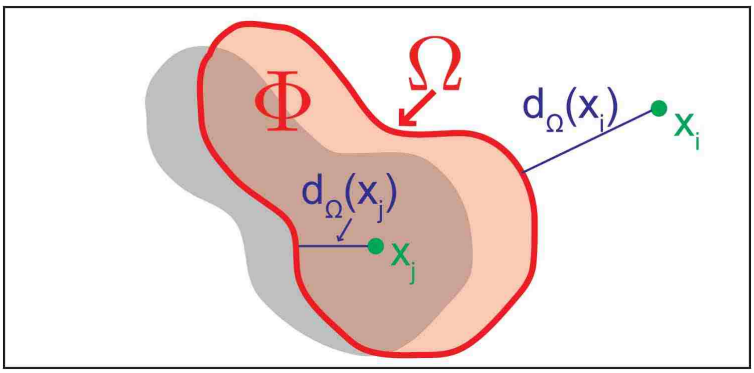


Figure 6.5: The distance $d_\Omega(x_i)$ (blue) is computed as the shortest distance from the point x_i (green) to Ω, the boundary of the object in the previous frame projected forward (red). Pixels within the object shape Φ (pink) such as x_j will occur a cost only if the pixel is labeled as background. Pixels outside of Φ such as x_i will occur a cost only if the pixel is labeled as foreground.

function combined with the estimation of the object motion comprise a novel shape prior for graph cut. The resulting costs produced by the shape cue are shown in Figure 6.3f.

6.5.5 Spatiotemporal Coherency

Videos usually exhibit a high amount of coherency between frames. Spatiotemporal-volume approaches [41, 80] implicitly capture this coherency through edges across frames. With our frame-by-frame approach, coherency between frames can be included without explicitly representing the labeled pixels from the previous frame. Rather, we assign a high region cost to label x_i as l if there is a nearby pixel (after motion adjustment) in the previous frame labeled $\bar{l}$ that has a similar color:

$$R_h(x_i, l) = \sum_{y_j \in N_{\bar{l}}(x_i)} \frac{1}{||C(x_i) - C(y_j)||^2 + 1} \tag{6.15}$$

where $N_{\bar{l}}(x_i)$ is the set of all neighbors of x_i from the previous frame that are labeled $\bar{l}$. Note that this formulation is similar to that of the gradient term described later in Section 6.5.7. Since this term can be represented as edges from one frame to similar pixels in the next, such a formulation is expected. Figure 6.3d shows the cost map for the spatiotemporal coherency

where the cat is likely foreground since it overlaps with the previous frame. The blockiness is due to the oversegmentation regions.

6.5.6 Point tracking

For most pixels in a typical video sequence, it is difficult to precisely determine the corresponding point in the next frame. However, easily-trackable points give nearly certain information about their labeling (see Figure 6.3e). While many algorithms make use of such points, video segmentation methods based on graph cut currently do not. We use [43, 65] to track these points and assign a penalty to labeling x_i as l if x_i is within a distance D (we use $D = 5$) of a tracked point that was labeled $\bar{l}$ in the previous frame:

$$R_p(x_i, l) = \begin{cases} 1 & \text{if } d_{\Theta_{\bar{l}}}(x_i) \leq D \\ 0 & \text{otherwise} \end{cases} \tag{6.16}$$

where $\Theta_{\bar{l}}$ is the set of tracked points labeled $\bar{l}$ and d is defined in Equation 6.13. Using Equation 6.16, labeling x_i as l incurs a penalty if x_i is within the distance D of a tracked point that was labeled $\bar{l}$ in the previous frame. Any points that were not reliably tracked are removed from Θ_l. We also filter out any points too close to the object boundary (within 10 pixels), because points near the boundary may potentially spill over onto the other side.

6.5.7 Gradient

Image gradients are important for encouraging selection boundaries to fall on image edges. As in [40], we use color difference as a boundary term:

$$B_g(x_i, x_j) = \frac{1}{||C(x_i) - C(x_j)||^2 + 1}. \tag{6.17}$$

where $C(x_i) \in [0, 255]^3$ is the color at x_i. Gradient boundary terms are standard practice in graph-cut segmentation.

6.5.8 Color Adjacency

Not only are the colors indicative of the objects, but the relationship of adjacent colors is as well. Certain color pairs may only exist within the object (background), while others only cross the object boundary. For example, the ballerina in Figure 6.15 contains a strong red-to-black edge in her clothing that only exists within her interior and never across her boundary. Ideally, a method should distinguish which transitions exist along the boundary and which do not by modeling the co-occurrence of colors along the object boundary and within the object or background.

While some methods have modeled the color profile of the object edge [47] or the geometry of an edge [48], they do not handle strong gradients within objects where a cut could occur. Recently, Cui et al. [19] modified gradient strength based on two colors on either side of the edge, but did not directly model the adjacency relationships. Additionally, their method required the color to be heavily quantized and did not specify exactly how the locality of edges is implemented. Other computer vision tasks have incorporated color adjacency relationships as histograms (essentially concurrence matrices) for use in image matching and retrieval and video scene segmentation [36, 29]. Unfortunately, the size of such histograms can be enormous, again requiring heavy quantization.

We introduce a new color-adjacency model to weight the importance of image gradients. The model is computed using a Fast Gauss Transform [83], similar to the color model. Adjacent pixels are represented by an 8D vector $e_{ij} = (l_i, u_i, v_i, l_j, u_j, v_j, x, y)$ where (l_i, u_i, v_i) is the color of pixel x_i, (l_j, u_j, v_j) the color of the x_j, and (x, y) their motion-adjusted location. A model is generated for all edges that are in the interior of either the foreground or background, and another model is generated for all edges along the boundary. Let the label I correspond to edges in the interior and the label E to edges along the desired boundary. These probabilities are combined into a boundary reweighting factor by

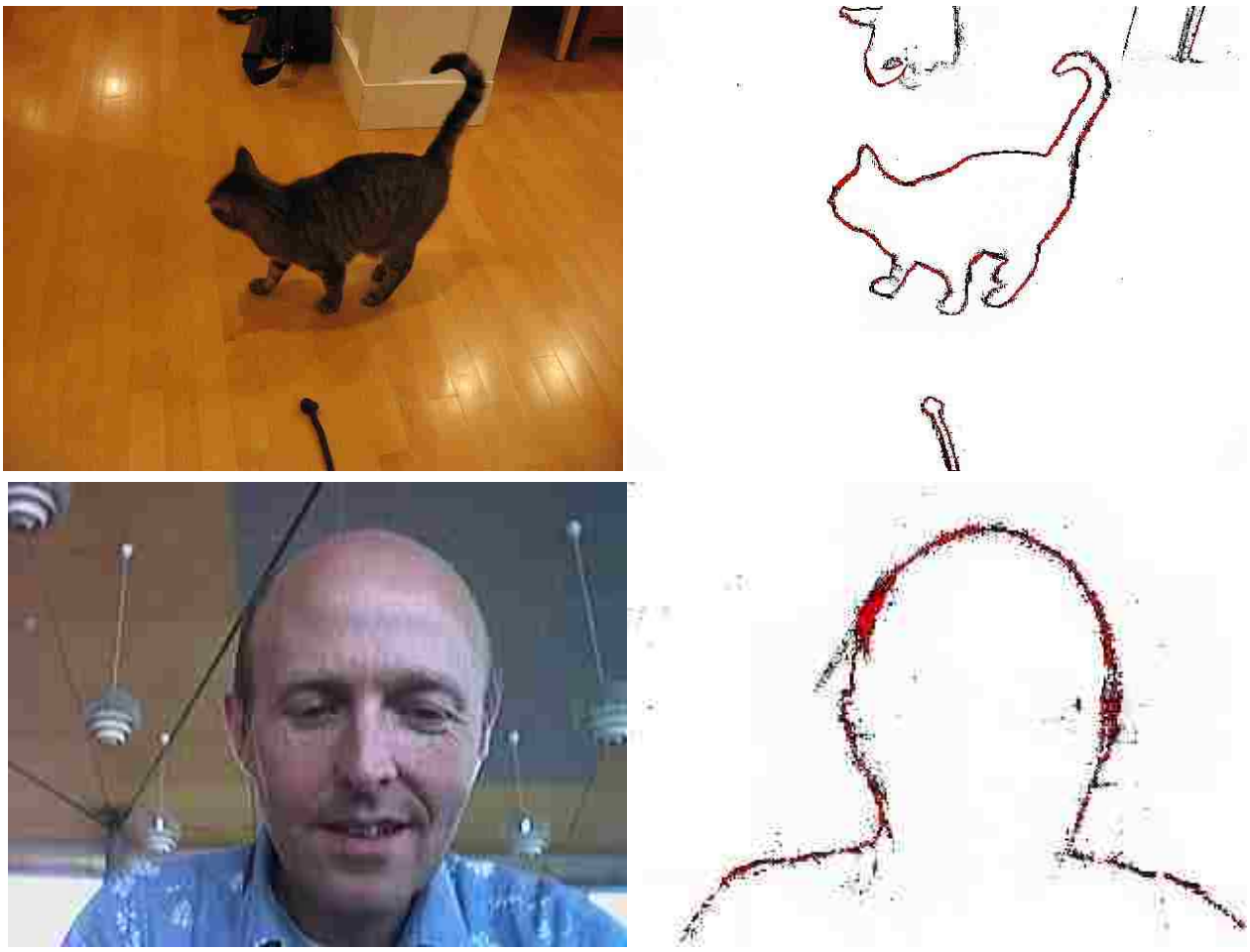
$$B_a(x_i, x_j) = \begin{cases} (1+\beta[P(E|e_{ij}) - P(I|e_{ij})])^{-2} & P(E|e_{ij}) > P(I|e_{ij}) \\ (1+\beta[P(I|e_{ij}) - P(E|e_{ij})])^{2} & \text{otherwise} \end{cases} \tag{6.18}$$

where $\beta > 0$ is a scalar that adjusts the effect of the color adjacency model and $P(E|e_{ij})$ is the posterior probability (assuming equal priors) that the pixel transition e_{ij} belongs to the object boundary (denoted by E) as given by

$$P(E|e_{ij}) = \frac{P(e_{ij}|E)}{P(e_{ij}|E) + P(e_{ij}|I)} \tag{6.19}$$

$P(e_{ij}|l)$ is defined as in Equation 6.7, except with an 8D vector as the initial parameter instead of a 5D vector as used in Equation 6.11. Equation 6.18 creates a scalar ranging from $(1+\beta)^{-2}$ if the models indicate a pure boundary ($P(I|e_{ij})=0, P(E|e_{ij})=1$) to $(1+\beta)^2$ for a pure interior edge ($P(I|e_{ij})=1, P(E|e_{ij})=0$), with a factor $B_a=1$ for equal interior and boundary probabilities ($P(I|e_{ij})=P(E|e_{ij})$). We use $\beta = 1$ in our formulation.

Figure 6.6 visualizes $P(E|e_{ij}) - P(I|e_{ij})$. The color red indicates that the probability of the transition being a desired edge is large, and white indicates that the probability of the transition being in the interior is large. More precisely, the range $[0, 1]$ for the difference is mapped to $[0, 255]$ in the red, green, and blue channels and the range $[0, -1]$ is mapped to $[0, 255]$ in the blue channel. Black indicates that both possibilities are equally likely. Notice how many of the gradients in the image which do not correspond to the object boundary but would be attractive places to place a segmentation boundary based on the strength of the edge have been suppressed. For example, the lines in the ceiling behind the man have been largely eliminated. Conversely, many edges along the desired boundary have been strengthened, such as along the boundary of the cat. Figure 6.3h shows the final weighting $B_a(x_i, x_j)$ of the color adjacency model in reference to the other cues. The cat's outline is clearly highlighted as the desired boundary, while other edges are suppressed.



(a) Image (b) Adjacency

Figure 6.6: Visualization of color adjacency difference $P(E|e_{ij}) - P(I|e_{ij})$ from Equation 6.18. Red indicates a high probability of the desired edge, white a high probability of an interior transition, and black indicates an equal probability of both.

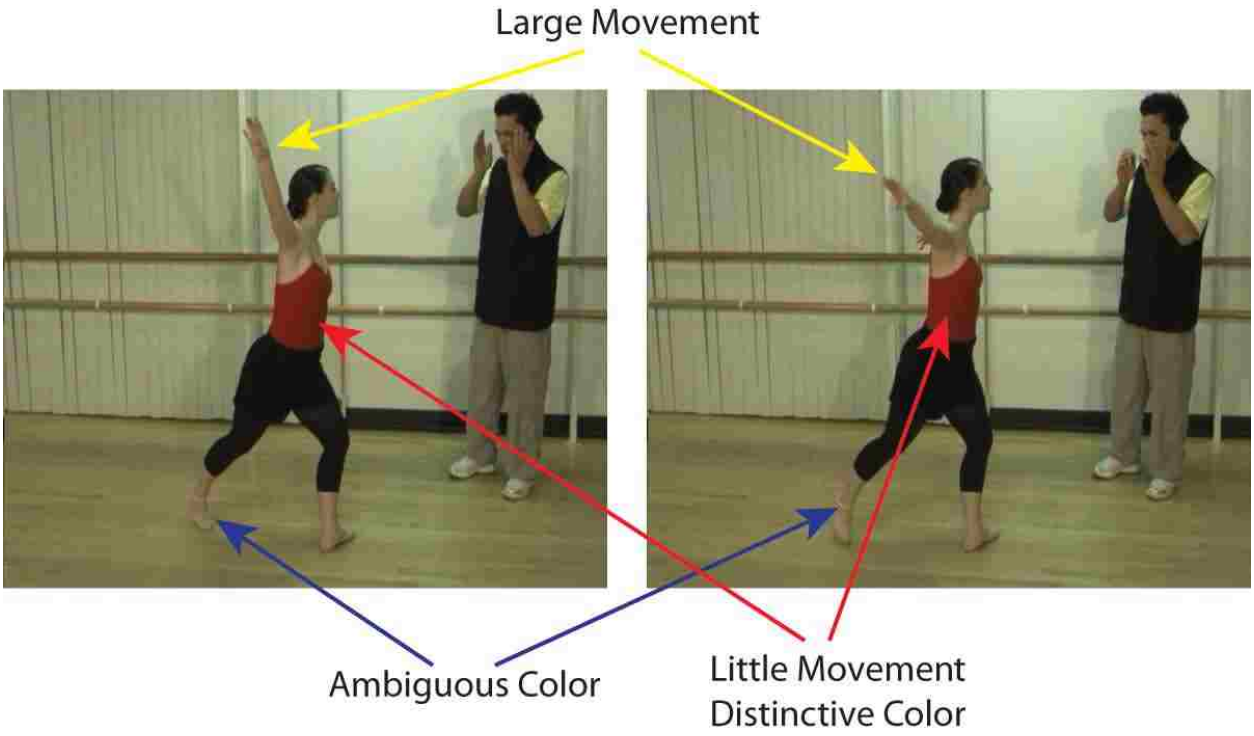


Figure 6.7: Spatially-differing difficulties in video. Two successive video frames are shown. Different regions of the video frames exhibit different difficulties.

6.6 Learning

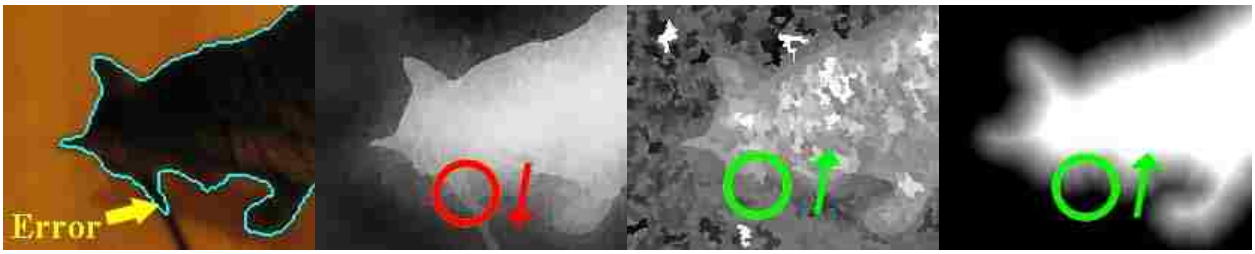
Section 6.5 introduces many different that cues can be combined in computing the selection of a video sequence. When using multiple informative cues, it is important that the different components be combined in a manner that will allow the best final result. This can be difficult because the ideal combination of cues will differ significantly from one video sequence to another, from one frame in a sequence to another, and from one area of one frame to another. One reason for this is because different cues will handle some of the difficulties inherent in video segmentation better than other cues.

For example, in Figure 6.7, the ballerina's feet have a similar color to the floor, so color cues will give little information about the selection, while the shape and location of the feet are more reliable. The ballerina's arm, on the other hand, moves such that much of it is not overlapping between frames, rendering spatiotemporal coherence information less effective. The red shirt changes very little between frames, so many cues would effectively

segment it. Because of the different properties of the object of interest and the background, the relative importance of each cue will change from one area of the frame to the next, and will not be consistent over time, so effective means of combining the cues will be essential.

In order to best leverage the various cues, we learn from the past performance of our cues and use that information to locally weight their application. We automatically weight the region terms in graph cut on a local basis, as shown in Equation 6.2 by the w_u factors. In this manner, the most effective cues will have a stronger effect. The basic idea behind this is as follows. An initial selection of a frame is computed. Since each region term gives a value in favor of the foreground $\mathcal{F}$ and the background $\mathcal{B}$, each suggests a label for each pixel. More precisely, if $R_u(x_i, \mathcal{F}) - R_u(x_i, \mathcal{B}) > 0$, then the term R_u would label x_i as background on its own because it costs more to label x_i as foreground than background. Conversely, if $R_u(x_i, \mathcal{F}) - R_u(x_i, \mathcal{B}) < 0$, R_u would label x_i as foreground on its own. This label can be viewed as the predicted label from a given cue. The user then corrects any mistakes by marking them with strokes before proceeding to the next frame. By comparing the initial propagated selection to the selection after corrections, we can determine which features were correct at each pixel and use that to weight their future performance. For example, consider the error and cost maps for three cues in Figure 6.8. Since the color cost for foreground $R_c(x_i, \mathcal{F})$ was less than the background cost $R_c(x_i, \mathcal{B})$, the future weight of the color term is weakened. The coherency and shape terms suggested the correct labeling and are strengthened.

We present several different algorithms for computing w_u: a statistical weighting, a simple punishment-reward method, one based on the delta rule [46], and the prediction with expert advise method [12]. We then present a methods of limiting the temporal scope of the learning algorithms.



(a) Error (b) Color (c) Coherency (d) Shape

Figure 6.8: (a) An error occurred in the propagated segmentation. Since the (b) color cue was incorrect, its weight is decreased. The (c) coherency and (d) shape cues were correct, so they are increased. The grayscale value in (b-d) visualizes the label the cue suggests, with white indicating foreground and black background.

6.6.1 Statistical

A straight-forward way to weight the differing cues is to use the previously-segmented frames to compute a probability that each cue is correct. This requires keeping a count at each pixel of how often each cue is correctly segmented and dividing that by the total number of frames already segmented.

Let us define the suggested labeling for a pixel x_i as suggested *solely* by cue u as

$$S_u(x_i) = \begin{cases} 1 & \text{if } R_u(x_i, \mathcal{B}) > R_u(x_i, \mathcal{F}) \\ 0 & \text{otherwise} \end{cases} \tag{6.20}$$

Weights for the next frame are then calculated by considering how often each of these suggestions agreed with the user's final segmentation:

$$w_u(x_i) \leftarrow 1 - \frac{1}{n} \sum_{j=1}^{n} \left| \hat{S}^j(x_i) - S_u^j(x_i) \right| \tag{6.21}$$

where S_u^j and $\hat{S}^j$ are the initial (Eq. 6.20) and final segmentations for frame j respectively, and n is the number of frames already segmented, including the current one.

6.6.2 Simple Reward-Punishment

Another way to have the segmentation system learn is to dynamically increment or decrement the weights after the user verifies the segmentation for each frame before proceeding to the next frame. This can be done using a straightforward reward/punishment approach as follows:

$$\Delta w_u(x_i) = \begin{cases} +\delta_0 & \text{if } S_u(x_i) = \hat{S}(x_i) \\ -\delta_1 & \text{if } S_u(x_i) \neq \hat{S}(x_i) \end{cases} \tag{6.22}$$

where $S_u(x_i)$ is again the initial segmentation suggestion made by considering cue u alone (Eq. 6.20), $\hat{S}(x_i)$ is the final segmentation for that pixel, and δ_0 and δ_1 are constant reward/punishment increments. We use $\delta_0 = 0.04$ and $\delta_1 = 0.08$, and w_u is initialized to 1.0.

We then update the weights accordingly:

$$w_u(x_i) \leftarrow w_u(x_i) + \Delta w_u(x_i) \tag{6.23}$$

The adjusted value of w_u is constrained to the range $[0, 1]$.

6.6.3 Delta-Rule Learning

Since we are training the weights for the unary (region-based) terms only, let us revisit Eq. 6.1 and focus solely on the unary component. In the absence of boundary terms, minimization of Eq. 6.1 devolves to a simple process of labeling each pixel based on choosing the least costly of $R(x_i, \mathcal{F})$ or $R(x_i, \mathcal{B})$, using Eq 6.2 to compute each.

Let $R_u^*(x_i)$ denote the difference between the foreground/background labeling costs for pixel x_i using cue u as follows:

$$R_u^*(x_i) = R_u(x_i, \mathcal{B}) - R_u(x_i, \mathcal{F}) \tag{6.24}$$

We can now write a variant of Eq. 6.2 in terms of R_u^*,

$$R^*(x_i) = s(x_i) + \sum_{u \in U} w_u(x_i)\ R_u^*(x_i) \tag{6.25}$$

and label suggestion becomes a trivial process:

$$S(x_i) = \begin{cases} 1 & \text{if } R^*(x_i) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.26}$$

Notice that except for the inclusion of user-placed seeds (which have infinite cost for mislabeling), *Eqs. 6.25 and 6.26 taken together have the form of a classic perceptron classifier* [59], where the per-cue $R_u^*(x_i)$ values serve as their feature inputs and $w_u(x_i)$ serve as the respective weights. We can thus use the well-known *delta rule* [46] to adjust the individual cue weights before proceeding to the next frame.

Training Using a Step Activation Function

Letting $S(x_i)$ continue to denote the initial segmentation of the current frame using only the region terms and $\hat{S}(x_i)$ again denote the final segmentation after user corrections, we update our individual cue weights as follows:

$$\Delta w_u(x_i) = \alpha \left[\hat{S}(x_i) - S(x_i)\right] R_u^*(x_i) \tag{6.27}$$

where α is a scalar controlling the learning rate.

For example, if cue u suggests that pixel x_i is foreground but the final label is background, then $\hat{S}(x_i) - S_u(x_i) = 0 - 1 = -1$. Since cue u suggested foreground, the cost to label x_i background was greater than the cost to label it foreground, so $R_u^*(x_i) > 0$. The overall product is negative, so $w_u(x_i)$ will be decremented according to the difference

$R_u^*(x_i)$ multiplied by a step size α. This is reversed for pixels initially incorrectly labeled as background.

Note that unlike offline training methods, we do not iterate this training to convergence over some training set. Instead, our online training approach adjusts the weights once each time we proceed from frame to frame.

A variation of his method is to use the initial segmentation generated by the graph-cut algorithm instead of the initial segmentation $S(x_i)$ computed using only the region terms:

$$\Delta w_u(x_i) = \alpha \left[\hat{S}(x_i) - \mathcal{L}_i \right] R_u^*(x_i) \tag{6.28}$$

In this case, the boundary term will effect the accuracy of the initial segmentation as well.

Training Using a Continuous Activation Function

The selection equation given Eq. 6.26 serves as a binary-valued *output activation function* when considering Eq. 6.25 as a feedforward neural network. As a result, the update rule in Eq. 6.27 adjusts individual cue weights only when the initial segmentation is incorrect, regardless of how close $R^*(x_i)$ is to the selection threshold (0). We can instead adjust the weights by using a continuous-valued activation function $f(x) \in [0, 1]$ when performing the training:

$$f(x) = \frac{1}{1 + \exp^{-x}} \tag{6.29}$$

and

$$\Delta w_u(x_i) = \alpha \left[\hat{S}(x_i) - f\left(R^*(x_i)\right) \right] R_u^*(x_i) \tag{6.30}$$

This form of update rule has the effect of continuing to adjust weights even when the initial frame segmentation is judged by the user to be correct. This is desirable to improve the future classification of margin cases where $R^*(x_i)$ is close to 0.

6.6.4 Advice from experts

In our segmentation system, many cues are used to determine the region cost for the graph cut algorithm. The combination of many cues is similar to the decision-making model of prediction with expert advice [12]. In this model, a forecaster makes label predictions based on the "advice" of several "experts", or the predictions of several other algorithms. Details of how the experts arrive at their advice need not be known. The model is learned online, with one input being introduced at each time step. Each expert gives its advice, and the forecaster then makes a prediction before learning the true label of the input. The forecaster may then re-evaluate how it regards the advice of each expert before moving onto the next input.

The goal of forecaster is to minimize the cumulative regret R_e for each expert (or cue) e in the finite set of experts E:

$$R_e = \sum_{t=1}^{n} (l(\hat{p}_t, y_t) - l(f_{e,t}, y_t)) \tag{6.31}$$

where n is the number of inputs already seen, $\hat{p}_t$ is the predicted label at time t, y_t is the input's true label, $f_{e,t}$ is the prediction of expert e, and l is a non-negative loss function. The predicted label $\hat{p}_t$ can be computed using a weighted average of the experts:

$$\hat{p}_t = \frac{\sum_{i=1}^{|E|} \omega_{i,t} f_{i,t}}{\sum_{j=1}^{|E|} \omega_{j,t}} \tag{6.32}$$

To minimize the regret, the weights must be set appropriately and updated with each new input. One method of updating the weights is to use an exponentially-weighted average forecaster:

$$\omega_{i,t+1} = \frac{\omega_{i,t} e^{-\eta l(f_{i,t}, y_t)}}{\sum_{j=1}^{|E|} \omega_{j,t} e^{-\eta l(f_{j,t}, y_t)}} \tag{6.33}$$

The formulation of the prediction with expert advice method is quite similar to that of our frame-by-frame segmentation algorithm. For a given pixel, a new input is given to

a collection of expert cues that give their prediction of the label. Their predictions are combined using graph cut to produce a final decision, and then the correct answer is given after the user has a chance to correct any errors. The cycle then repeats with a new input from the next frame.

Because of the similarity, we use a weight-update formula derived from Equation 6.33:

$$w_u(x_i) \leftarrow \frac{w_u(x_i)e^{-\eta l(R_u^*(x_i),\hat{S}(x_i))}}{\sum_{j=1}^{|U|} w_j(x_i)e^{-\eta l(R_j^*(x_i),\hat{S}(x_i))}} \tag{6.34}$$

with the loss function

$$l(R_u^*(x_i), \hat{S}(x_i)) = ||R_u^*(x_i) - \hat{S}(x_i)||^2 \tag{6.35}$$

and $\eta = \sqrt{2 \ln |U|/n}$ where n is the number of frames already segmented in the sequence including the current frame. Note that this update scheme works to minimize the regret according to the prediction in Equation 6.32, which corresponds well to how the region cues are combined to produce the region term R in our graph-cut formulation, adding only a normalization calculation.

6.6.5 Limiting the temporal scope of the learning

Video sequences often have large changes that occur during the sequence that change the characteristics of the object or background. This may occur due to objects entering or leaving the scene, to camera motion introducing a new background, or to changes in position or appearance of the object. When large changes occur, any learned information may no longer be valid. At this point, it often would be best to re-initialize our learning algorithm and begin learning again.

Determining when such changes occur can be a difficult problem in itself. We instead take a more simplistic approach of only accumulating the error over a small number of frames preceding the current frame (we use five frames). With this method, large changes will be quickly forgotten, and only the information most likely to be current will be used.

6.7 Results

In our method, we combine information cues adaptively using a learning algorithm. We evaluate these components individually to show their effectiveness in our system. First, we compare the proposed learning algorithms in Section 6.6 to determine which learning method is most successful. We then compare the performance of the individual cues.

For an interactive segmentation system, the real measure of success is the amount of user time required to perform a selection. Accordingly, we report the time required to select objects in several sequences. The segmentations can be judged qualitatively by the examples shown in Figure 6.15. In order to better evaluate the accuracy of LIVEcut, we also compare it to automatic segmentation techniques, despite the disadvantage this gives to an algorithm designed for interactive use.

6.7.1 Ground-truth Experiments

Our experiments with ground-truth data proceed as follows. The selection is initialized using the ground truth for the first frame. If the ground truth contains unlabeled pixels due to mixed pixels (along edges of objects, hair, or blurred areas), then these pixels are not seeded and are solved for using graph cut. The selection is then propagated to the next frame. This selection is compared to the ground-truth for that frame, and any incorrect pixel is counted toward the error. The ground truth is then used to correct the mistakes, and the algorithm proceeds to the next frame.

Our dataset of video sequences with ground-truth data is a collection of sequences and ground-truth provided by [74]. As these sequences were designed for automated motion layer segmentation algorithms, there are parts of sequences where no foreground objects are in view or where multiple objects exist. Accordingly, we chose frames that were suitable for an interactive system. The sequences and frames used are shown in Table 6.1. An example frame from each of the sequences is shown in Figure 6.9.

Sequence	Frames
dancing_chachacha_v1	185-1485
dancing_chachacha_v4	185-1485
exhausted_runner	60-125
hot_day	50-290
ingravity	1-90
playing_alone	1-306
teddy_bear	25-300

Table 6.1: Ground-truth video sequences from [74].

Figure 6.9: Example frames from each sequence in our ground-truth dataset. Starting on the top row, from left to right, the frames are from dancing_chachacha_v1, dancing_chachacha_v4, exhausted_runner, hot_day, ingravity, playing_alone, and teddy_bear. Sequences are from [74].

6.7.2 Learning Evaluation

We compare these learning algorithms on a set of video sequences with ground-truth data. For this experiment, the ground truth is used to initialize the segmentation in the first frame, similar to how a user would correctly segment the first frame. The proceeding frame is then segmented, and the number of mislabeled pixels according to the ground truth is computed as error. The ground truth is then used to correct the segmentation (as a user would), and the selection proceeds to the next frame.

All the cues presented in Section 6.5 are used to perform the segmentation and are weighted according to the learning algorithms. Table 6.2 gives the average error over all the

Sequence	*No Learning*	*Delta (graph cut)*	*Delta (step)*	*Delta (sigmoid)*	*Punish-Reward*	*Statistical*	*Experts*	*Experts (window)*
dancing_chachacha_v1	204	-8.5 %	-8.9 %	-9.1 %	-7.6 %	-8.7 %	-4 %	-5.6 %
dancing_chachacha_v2	148	-16.6 %	-15.3 %	-15.7 %	-18.6 %	-16.4 %	78.3 %	-8.5 %
exhausted_runner	54	15.6 %	16.2 %	16.2 %	16.4 %	19.1 %	-14.5 %	-13.1 %
hot_day	120	-7.4 %	-4.1 %	-5.9 %	-8 %	-4.9 %	-15.2 %	-14.2 %
ingravity_1	100	-16.9 %	-16.9 %	-16.9 %	-16.9 %	-16.9 %	-11.9 %	-9.9 %
playing_alone	140	-5 %	-3.3 %	-3.9 %	-4.6 %	-3.3 %	-16.4 %	-5.3 %
teddy_bear	45	-6.4 %	-6.1 %	-6.3 %	-8.5 %	-4.2 %	-3.3 %	-4.4 %
Sequence Average	116	-8.5 %	-7.6 %	-8.1 %	-8.8 %	-7.5 %	5.6 %	-8.3 %
Frame Average	155	-10.9 %	-10.3 %	-10.7 %	-11.2 %	-10.6 %	22.8 %	-7.1 %

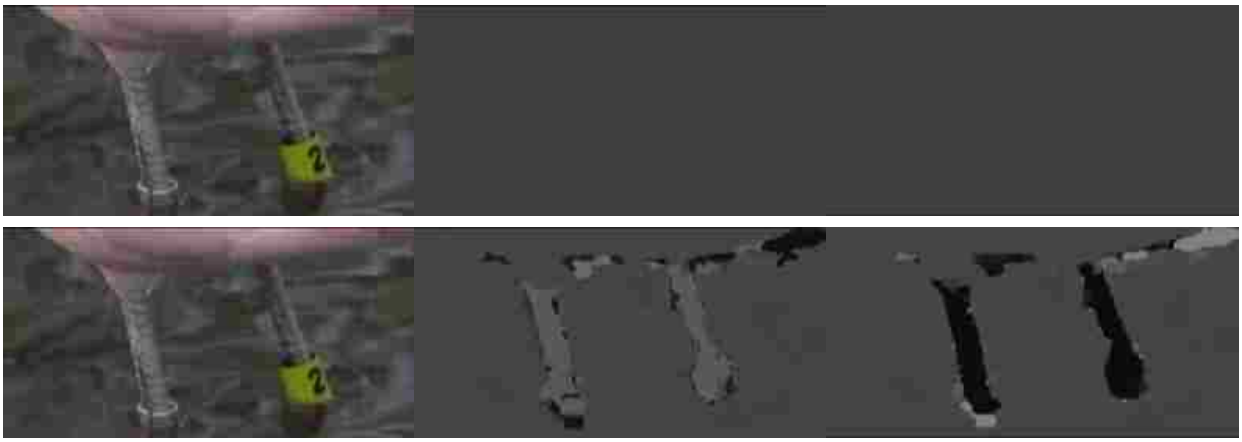
Table 6.2: The average error (number of mislabeled pixels) over the ground-truth sequences is calculated for each learning algorithm. All the cues presented in Section 6.5 are in use for this experiment. The three variations of the delta function (graph cut, step, and sigmoid) use Equations 6.28, 6.27, and 6.30 respectively. The "windows" variation of experts limits the temporal scope of the experts method to a window. Green text shows a desired decrease in error when using a learning method, and red text indicates and increase in error.

video sequences for each of the learning algorithms. We also tested the learning methods when restricting the learning to only a small temporal window around the frame in question. This had a small or negative effect on all learning methods except for the learning from expert advice method, which is the only result shown in the table.

Two average scores are given in Table 6.2. The sequence average is the average of the average error over each sequence (the average of the numbers in the column above it) and does not take into consideration that the sequences are of different lengths. The frame average respects the length of the sequences by averaging over all frames from all sequences (equivalent to scaling each sequence error in Table 6.2 by the number of ground-truth frames for each sequence before averaging them).

Overall, the best performance was achieved by the reward-punishment method accumulating error over the entire sequence. The delta function method performed nearly as well overall, and many methods outperformed the reward-punishment method on individual sequences.

A visualization of changing learning weights is shown in Figure 6.10. The grayscale value in the weight images represents the value of the weights, with white indicating a high



Original Shape Color

Figure 6.10: Learning weights for the "flamingo" example. A portion of the frame as well as the weights for the color and shape cues are shown for an initial frame (top row) and the following frame (bottom row). The grayscale value in the weight images correlates to the weight values. The weights on the following frame have changed to favor the shape over the color near the legs of the flamingo since the color is ambiguous in those areas.

weight and black a low weight. In this example, the legs of the flamingo are of a similar color to the background water so the color cue performs poorly in those areas. The shape of the legs, however, is consistent. As the user proceeds to the second frame, the weights adjust to favor shape over color around the legs.

6.7.3 Cue Evaluation

To evaluate the performance of the cues, we segment the video sequences described in Section 6.7.1. In each case, the sequences are segmented with one of the cues disabled. The number of mislabeled pixels is then computed using the ground truth. By disabling one cue at a time, we may see the impact that individual cue is having on the overall error. In all cases, we use the reward-punishment learning algorithm described in Section 6.6.2 to weight the applications of the cues.

Table 6.3 gives the results of this experiment. The largest increase in error occurs with the removal of the gradient cue which apparently contributes the most toward computing the correct solution. The global color model, motion, and coherence cues also have a large

effect on error. The other cues have a smaller effect. The only effect that causes a decrease of error on average when removed is the color cue.

These results also show that there remains room for improvement in our learning method. If there is improvement in the error rate for a given sequence if a cue is dropped entirely, then the learning algorithm was unable to accurately weight the cues to provide the best results.

To show the effect of the region cues, each cue was used to segment the object on its own. The results are shown in Table 6.4. The spatiotemporal coherence cue proved to be most correct overall. The shape and color cues are fairly comparable in their average error over the sequences to the spatiotemporal cue but have much higher error in the frame average. The global color cue performs far worse than the other cues when in isolation. The point-tracking cue was not evaluated because it only provides a labeling for a small subset of the total number of pixels in the image.

One interesting note is that the global color model performs worse on its own when compared to the other region cues as shown in Table 6.4 but is more helpful in reducing error when combined with the other cues as shown in Table 6.3. This is likely because it provides a different type of information, global in scope temporally and spatially, allowing it to provide information that the other cues cannot.

6.7.4 Timing and Qualitative Results

Table 6.5 gives timing results over several challenging video sequences. The "footballer" sequence exhibits large motions, a drastically changing object shape, and a partial occlusion from another moving object. "Bass guitar" and "lemurs" both contain overlapping color models, boundaries where there is no gradient information, and motion blur. While much of the body of the "flamingo" is easy to segment, the legs are narrow, exhibit large movements, are often heavily blurred, and have a similar color to the background. Using LIVEcut, a user is able to segment the objects without excessive interaction. The selection computed

Sequence	All	No Motion	No Color	No Global Color	No Shape	No Coherence	No Point Tracking	No Gradient	No Adjacency
dancing_chachacha_v1	189	49.2 %	-4.2 %	15.7 %	17.4 %	117.7 %	-1.1 %	1143.1 %	10.1 %
dancing_chachacha_v4	120	86.3 %	-3.3 %	45.2 %	12.4 %	107.8 %	4.3 %	2013.1 %	6.9 %
exhausted_runner	63	57 %	-5.3 %	37.6 %	-22.5 %	15.2 %	-8.6 %	3084 %	32.2 %
hot_day	111	51 %	-3.6 %	35.4 %	26.9 %	82.1 %	1.6 %	1518 %	3.8 %
ingravity_1	83	70 %	-0.1 %	8.6 %	14.9 %	278.7 %	-0.1 %	436.7 %	-27.1 %
playing_alone	134	40 %	-4.3 %	50.6 %	10.1 %	48.4 %	0.7 %	1226.6 %	17.8 %
teddy_bear	42	102.8 %	0.1 %	68.9 %	106.8 %	70.6 %	2 %	2487.3 %	3.4 %
Sequence Average	106	59.8 %	-3.4 %	33.8 %	18 %	105.1 %	0.1 %	1515.3 %	7.4 %
Frame Average	138	61.9 %	-3.8 %	30.4 %	17.4 %	107.4 %	0.9 %	1483.3 %	8.9 %

Table 6.3: The effect of the cues on ground-truth video sequences. The average error (number of mislabeled pixels) over the ground-truth sequences is shown in the column labeled "All". Each of the cues is then turned off one at a time, and the change in the error is shown. The green text shows an increase of error, indicating that the disabled cue must have contributed information that helped lower the cue. Red text shows a decrease in error, indicating that the disabled cue actually hurt the average performance and is best not used in that case.

Sequence	Color	Global Color	Shape	Coherence
dancing_chachacha_v1	2208	3275	1498	782
dancing_chachacha_v4	1570	3620	1493	443
exhausted_runner	1670	1142	1587	520
hot_day	930	1822	1023	271
ingravity_1	169	128	178	96
playing_alone	1403	752	1402	562
teddy_bear	531	1584	449	143
Sequence Average	1212	1760	1090	402
Frame Average	1632	2841	1345	535

Table 6.4: Error in segmentation when performed using only a single region cue. The sequence average is computed by averaging the average error over all the sequences as shown in the table. The frame average is computed by averaging the error over all frames in all sequences.

Video	Size	Graph Cut Time	User Time
Bass Guitar	960×540×72	0.047 / 3.14 sec	20 min
Cat	640×480×56	0.024 / 1.50 sec	4 min
Flamingo	960×540×76	0.046 / 2.33 sec	16 min
Footballer	720×576×19	0.030 / 1.79 sec	4 min
Lemurs	960×540×86	0.037 / 2.71 sec	17 min

Table 6.5: Timing results from several sequences. The graph-cut time first gives the time to process an interaction on one frame, and then the time to propagate information to the next frame. "Footballer" is courtesy of Artbeats (www.artbeats.com).

		[80]				LIVEcut	
Video	Size	Pre-process	Graph Cut Time	User Time	Post-process	Graph Cut Time	User Time
amira	640×480×35(80)*	12 min	5 sec	15 min	35 min	0.030 / 1.41 sec	5 min
ballerina	640×480×150	25 min	11.5 sec	140 min	30 min	0.025 / 1.51 sec	48 min
elephant	720×480×100	20 min	9.1 sec	40 min	30 min	0.031 / 2.11 sec	27 min
manincap	640×480×150	30 min	16.5 sec	20 min	35 min	0.022 / 1.57 sec	21 min
stairs	640×480×63(100)*	20 min	8.5 sec	20 min	30 min	0.019 / 1.36 sec	13 min

Table 6.6: Comparison of LIVEcut to Interactive Video Cutout [80]. The graph-cut time for LIVEcut lists first the time to process an interaction on one frame, and then the time to propagate the selection to the next frame. For the "amira" video sequence, [80] could not effectively segment the object, and [1] was also used. The '*' indicates that the video we obtained differed in length to that reported in [80] (shown in parentheses). The postprocess time for [80] consists of pixel-level refinement, but also includes matting, which is not reported for our method. LIVEcut does not need any pre-processing time.

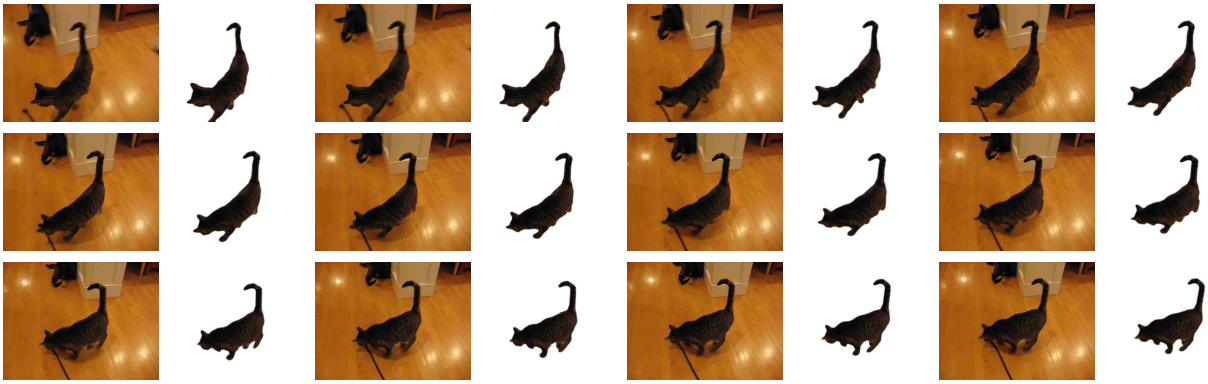


Figure 6.11: Frames and object selections from the "cat" sequence.

over a range of consecutive frames is shown in Figures 6.11-6.14 for several sequences. The selection from selected frames from many of the sequences is also shown in Figure 6.15. We apply the robust matter [81] to our output to account for mixed pixels on boundaries.

We compare LIVEcut to [80] using videos from this paper in Table 6.6. The user time to acquire binary segmentation results similar in quality to these techniques is comparable or less in these examples. The time the user must wait between each interaction for the selection to update is also less, providing a better interactive experience. Our algorithm also does not need the large preprocessing time that [80] requires. We were able to segment "amira" with LIVEcut, while [80] required the help of [1] to do so. We also were able to segment the "ballerina" as one object, while [80] required one pass for the feet and another for the body. Finally, our user interaction is simpler, requiring only drawing strokes on individual frames and allowing sequential processing of the video, while [80] also requires rotating and slicing through a spatiotemporal volume.

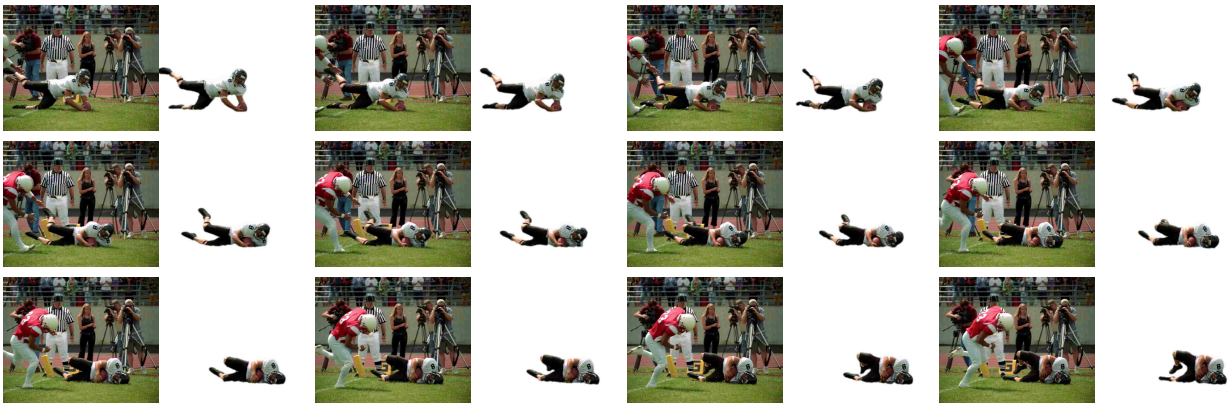


Figure 6.12: Frames and object selections from the “footballer” sequence.

Figure 6.13: Frames and object selections from the “manincap” sequence.

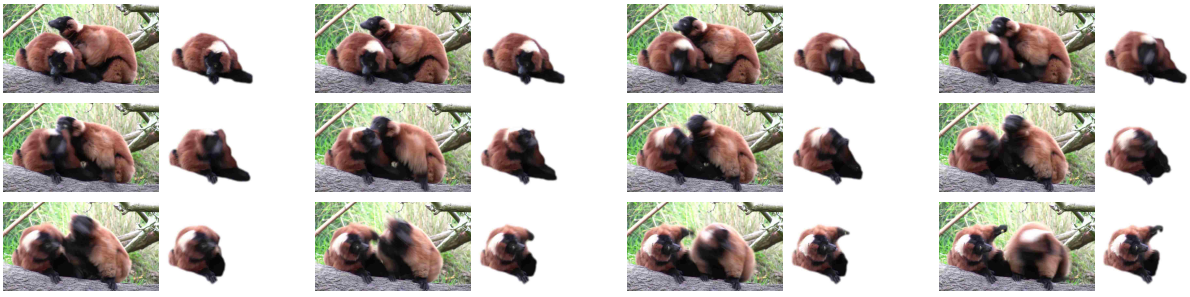


Figure 6.14: Frames and object selections from the “lemurs” sequence.

Figure 6.15: Several examples of object selections using LIVEcut.

Sequence	41	43	50	51	54
LIVEcut Error %	2.71	6.71	1.45	0.38	0.90
Yin 2007 Error %	0.80	0.02	1.31	1.06	0.33
Sequence	56	58	60	IU	JM
LIVEcut Error %	2.27	0.07	14.85	3.52	28.16
Yin 2007 Error %	0.93	0.79	6.33	2.56	0.27

Table 6.7: Comparison of [84] to LIVEcut using automatic segmentation (i.e. *without* allowing user corrections).

6.7.5 Accuracy and Stability

For interactive segmentation systems, accuracy is difficult to measure since a user can always achieve perfect accuracy given enough time. To demonstrate accuracy, we perform automatic segmentations and compare to the unsupervised method from [84] on their database. In doing so, LIVEcut faces a large disadvantage. LIVEcut was designed to assume that the previous frame was correctly segmented by the user, and proceeds under that assumption. Furthermore, LIVEcut receives no user training, while [84] is trained on similar data. While this test neutralizes many of the strengths of LIVEcut, it allows us to show the algorithm's accuracy and stability.

For this test, we segmented the first frame of ten sequences, each of size 320×240 with an average length of over 350 frames. We then computed the segmentation *without* additional user interaction and compare to the results from [84] in Table 6.7. For several of the videos (50, 51, 54, 58, IU), we have comparable or better results. For the others, the accuracy over time is shown in Figure 6.16. Our segmentation error in each case was very low until an abrupt increase due to a change in the scene. For three of the cases, the error is low until the subject moves his hand in front of his body. In these cases, LIVEcut assumes that the hand is an occluding object that it should not segment and does not recover the entire object once the hand leaves. In the other cases, a rapid motion confuses our algorithm. Note that in each case, the error is quite stable after the initial mistake because LIVEcut accurately tracks what it assumes is the new state of the object.

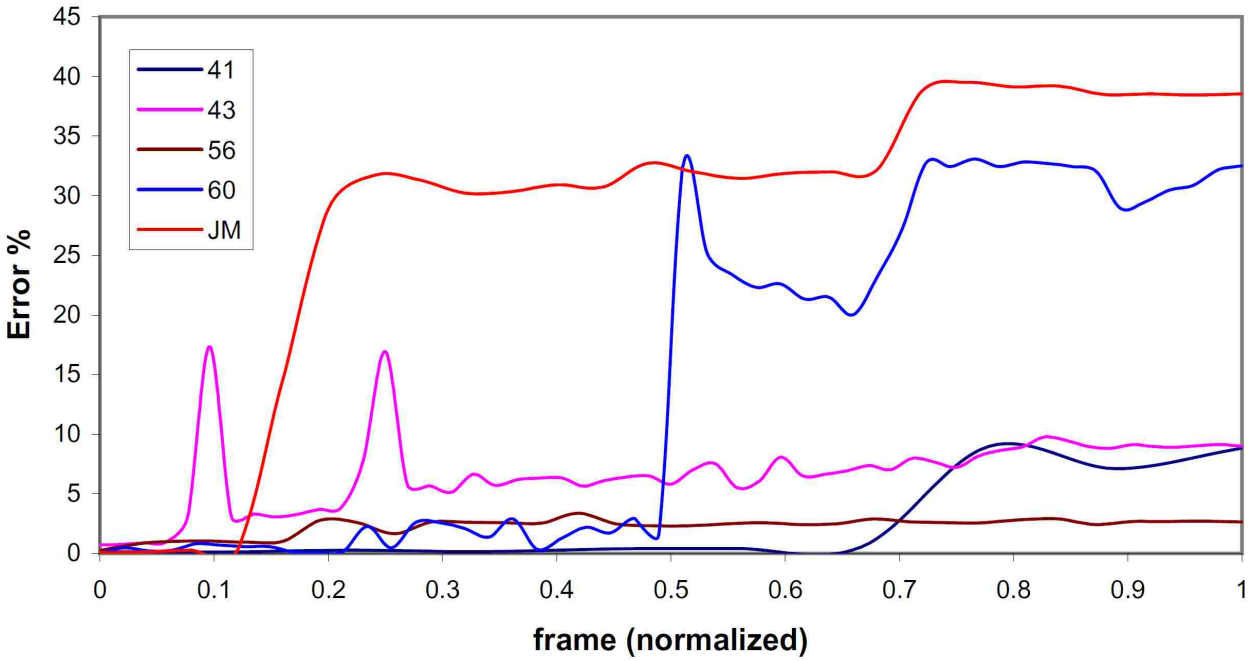


Figure 6.16: Accuracy of several sequences from Table 6.7.

Sequence	41	43	56	60	JM
LIVEcut Error %	1.02	1.46	1.56	2.34	1.11
# frames corrected	1	4	1	1	6

Table 6.8: Accuracy of LIVEcut on sequences from Table 6.7 after corrections on the number of frames shown.

To better show the accuracy and stability on these sequences, we resegmented the video allowing corrections only on or near the frames where large errors occur. Table 6.8 shows that the accuracy is now similar to or less than [84] while allowing very few corrections. While LIVEcut can achieve similar results to unsupervised methods with little or no corrections, these methods could not produce the high quality results from LIVEcut shown in Figure 6.15.

6.8 Conclusion

We have presented a new method for interactively segmenting video sequences by propagating multiple cues from one frame to another. These cues are automatically weighted based on learning from user corrections. Many of the cues also include novel improvements in the context of video segmentation using graph cut.

While propagating multiple weighted cues is effective in segmenting video, further improvements can be made. LIVEcut only uses cues from the previous frame together with the accumulated learning. However, more global information about the entire video sequence may assist the segmentation. Improved learning techniques may better weight the graph-cut terms.

Chapter 7

Conclusion

This dissertation has investigated computing object selections in video sequences by combining multiple informative cues. The cues are dynamically weighted based on user corrections in order to improve overall performance. Many novel cues or new improvements to cues have also been introduced. This includes a color adjacency model that allows the algorithm to focus only on edges of interest. Results, both qualitative and quantitative, have shown that this system provides improved performance on many video sequences.

The concept of combining multiple pieces of information to improve selection results was also applied to image segmentation. This work has shown that geodesic segmentation and graph-cut segmentation have complementary strengths and weaknesses. Because the failure cases of geodesic segmentation are relatively easy to identify, the two algorithms may be combined such that each is applied in local areas where it is most effective. The approach allows objects to be selected using fewer user strokes. This approach also achieved the best performance on the GrabCut database of any reported method not using a specific input constraint or implicit bias.

In order to provide complete object selection in video, video matting is required. In order to help the state-of-the-art progress to the point where video matting is effective, this work has introduced an improved image matting algorithm. This algorithm produces mattes comparable to other current techniques while improving on the foreground and background color estimations.

While much progress has been made in improving video segmentation, many difficult challenges remain. Several different problem cases have not been fully solved, such as occlusions and fast-moving or fast-changing objects. Higher-level models and object detection techniques may assist with these cases. Improvements in shape modeling would also help improve results.

Another problem that can still occur is to correctly localize the object boundary in cases with difficult textures, overlapping color models, or motion blur. While the method presented here does quite well at identifying the bulk of the object, in these difficult cases the user may be required to correct small mistakes on the boundary. It may be helpful to further research methods of modeling and identifying small regions of the object boundary. This is a strength of the Video Snapcut paper [5], which tracks and models small sections of the object boundary to try to localize it correctly. More work should be done in this area.

Image matting still remains largely unsolved. Despite the progress made in this dissertation as well as other current research, matting is only effective in simple cases. In areas with complex textures or overlapping color models, matting still largely fails. Unfortunately, these complications are very common in natural images. Improved constrains, especially those handling textures, may lead to improved results. Higher-order image understanding may be necessary in order to handle many of the difficult cases.

In order to correctly segment video, video matting is required. One of the hurdles to be faced by those looking to improve video matting is that image matting still falls short. Besides this significant problem, the other major problem in video matting is maintaining good temporal coherency. Perhaps the easiest way to enforce temporal coherency is to view the lack of coherency as high-frequency noise and simply blur it away. Unfortunately, blurring in the spatial domain in many cases would completely destroy the results, so the blur must only occur in the temporal domain. This requires highly accurate frame-to-frame pixel correlation. Current correlation algorithms cannot provide the accuracy needed given natural videos that may contain huge movements, lighting changes, occlusions, and blur.

Either great improvements in correlation must be made, or a method of enforcing temporal coherence without correlation must be developed.

One area that I feel is greatly lacking in research is that of interactively editing mattes. While fully automated image and video segmentation algorithms exist, accurate object selection is largely viewed as an interactive process, and tools are provided to users to easily make corrections to acquire the desired selection. While producing correct mattes is even more difficult than producing a binary segmentation, easy-to-use accurate matting editing tools do not exist. Currently, the only interactive methods consist of refining the trimap [26, 57, 79] or specifying areas over which to reapply the matting algorithm locally [71]. While these are legitimate approaches, they do not provide easy and direct methods of altering the foreground, background, or alpha estimation. More direct methods are needed to allow users to generate a desirable matte.

References

[1] Aseem Agarwala, Aaron Hertzmann, David H. Salesin, and Steven M. Seitz. Keyframe-based tracking for rotoscoping and animation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 23(3):584–591, 2004. ISSN 0730-0301.

[2] Christopher Armstrong, William Barrett, and Brian Price. Live surface. *Proceedings of Volume Graphics 2006*, 2006.

[3] Christopher J. Armstrong, Brian L. Price, and William A. Barrett. Interactive segmentation of image volumes with live surface. *Computers and Graphics*, 31(2):212–229, April 2007.

[4] Xue Bai and Guillermo Sapiro. A geodesic framework for fast interactive image and video segmentation and matting. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, 2007.

[5] Xue Bai, Jue Wang, David Simons, and Guillermo Sapiro. Video snapcut: robust video object cutout using localized classifiers. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 1–11, 2009.

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

[7] Andrew Blake, Carsten Rother, M. Brown, Patrick Perez, and Philip Torr. Interactive image segmentation using an adaptive GMMRF model. In *Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, pages 428–441, May 2004.

[8] Andrew Blake and Andrew Zisserman. *Visual Reconstruction*. MIT Press, Cambridge, MA, 1987.

[9] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 105–112, 2001.

[10] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, volume 26, pages 1124–1137, September 2004.

[11] Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient N-D image segmentation. *International Journal of Computer Vision (IJCV)*, 70(2):109–131, 2006.

[12] Nicolo Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006. ISBN 0521841089.

[13] Hanna Chidiac and Djemel Ziou. Classification of image edges. In *Vision Interface*, pages 17–24, 1999.

[14] Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. A bayesian approach to digital matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 264–271. IEEE Computer Society, December 2001.

[15] P. Colantoni and B. Laget. Color image segmentation using region adjacency graphs. In *Proceedings of the International Conference on Image Processing and Its Applications*, volume 2, pages 698–702, Jul 1997.

[16] Camille Couprie, Leo Grady, Laurent Najman, and Hugues Talbot. Power watersheds: A new image segmentation framework extending graph cuts, random walker and optimal spanning forest. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2009.

[17] A. Criminisi, G. Cross, A. Blake, and V. Kolmogorov. Bilayer segmentation of live video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 53–60, June 2006.

[18] A. Criminisi, T. Sharp, and A. Blake. GeoS: Geodesic image segmentation. In *Proceedings of the IEEE European Conference on Computer Vision (ECCV)*, 2008.

[19] Jingyu Cui, Qiong Yang, Fang Wen, Qiying Wu, Changshui Zhang, L. Van Gool, and Xiaoou Tang. Transductive object cutout. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008.

[20] O. Duchenne, J.-Y. Audibert, R. Keriven, J. Ponce, and F. Segonne. Segmentation by transduction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2008.

[21] James H. Elder and Steven W. Zucker. Scale space localization, blur, and contour-based image coding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 27–34. IEEE Computer Society, 1996.

[22] J. Fairfield. Toboggan contrast enhancement for contrast segmentation. *Proceedings of the International Conference on Pattern Recognition (ICPR)*, 1:712–716, 1990.

[23] Daniel Freedman and Tao Zhang. Interactive graph cut based segmentation with shape priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 755–762, 2005.

[24] Michael Gleicher. Image snapping. *International Conference on Computer Graphics and Interactive Techniques*, pages 183–190, 1995.

[25] Leo Grady. Random walks for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 28(11):1768–1783, November 2006. ISSN 0162-8828.

[26] Leo Grady, Thomas Schiwietz, Shmuel Aharon, and Rüdiger Westermann. Random walks for interactive alpha-matting. In *Proceedings of the Conference on Visualization, Imaging and Image Processing (VIIP)*, pages 423–429, 2005.

[27] J. Guan and G. Qiu. Interactive image segmentation using optimization with statistical priors. In *ECCV International Workshop on The Representation and Use of Prior Knowledge in Vision*, 2006.

[28] Yu Guan, Wei Chen, Xiao Liang, Zi'ang Ding, and Qunsheng Peng. Easy matting: A stroke based approach for continuous image matting. In *Proceedings of the Conference of the European Association for Computer Graphics (Eurographics)*, pages 567–576, 2006.

[29] Allan Hanbury and Beatriz Marcotegui. Colour adjacency histograms for image matching. In *Proceedings of the International Conference on Computer Analysis of Images and Patterns (CAIP)*, pages 424–431, 2007.

[30] Chun hao Wang and Ling Guan. Graph cut video object segmentation using histogram of oriented gradients. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2590–2593, May 2008.

[31] Seth Holladay. Intelligent rotoscoping: A semi-automated interactive boundary tracking approach to video segmentation. Master's thesis, Brigham Young University, Provo, UT, 2007.

[32] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 259–268, 1987.

[33] V. Kolmogorov and Y. Boykov. What metrics can be approximated by geo-cuts, or global optimization of length/area and flux. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 1, pages 564–571, Oct. 2005.

[34] V. Kolmogorov, Y. Boykov, and C. Rother. Applications of parametric maxflow in computer vision. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2007. ISSN 1550-5499.

[35] M.P. Kumar, P.H.S. Ton, and A. Zisserman. Obj cut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 18–25, June 2005.

[36] Ho Young Lee, Ho Keun Lee, and Yeong Ho Ha. Spatial color descriptor for image retrieval and video segmentation. *IEEE Transactions on Multimedia*, 5(3):358–367, 2003.

[37] Victor Lempitsky, Pushmeet Kohli, Carsten Rother, and Toby Sharp. Image segmentation with a bounding box prior. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2009.

[38] Anat Levin, Dani Lischinski, and Yair Weiss. A closed form solution to natural image matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 61–68, Washington, DC, USA, 2006. IEEE Computer Society.

[39] Anat Levin, Alex Rav-Acha, and Dani Lischinski. Spectral matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(10):1699–1712, 2008.

[40] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 303–308, 2004.

[41] Yin Li, Jian Sun, and Heung-Yeung Shum. Video object cut and paste. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 24(3):595–600, 2005. ISSN 0730-0301.

[42] Herve Lombaert, Yiyong Sun, Leo Grady, and Chenyang Xu. A multilevel banded graph cuts method for fast image segmentation. In *Proceedings of the IEEE International*

Conference on Computer Vision (ICCV), pages 259–265, Washington, DC, USA, 2005. IEEE Computer Society.

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

[44] J. Malcolm, Y. Rathi, and A. Tannenbaum. A graph cut approach to image segmentation in tensor space. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2007.

[45] Calvin R. Maurer, Jr., Rensheng Qi, and Vijay Raghavan. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 25(2):265–270, 2003. ISSN 0162-8828.

[46] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[47] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 191–198, 1995.

[48] E. N. Mortensen and W. A. Barrett. Toboggan-based intelligent scissors with a four parameter edge model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 452–458, 1999.

[49] David M. Mount and Sunil Arya. ANN: A library for approximate nearest neighbor searching. http://www.cs.umd.edu/~mount/ANN/, 2010.

[50] Brian Price, Bryan Morse, and Scott Cohen. LIVEcut: Learning-based interactive video segmentation by evaluation of multiple propagated cues. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2009.

[51] Brian Price, Bryan Morse, and Scott Cohen. Color adjacency modeling for improved image and video segmentation. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, 2010.

[52] Brian Price, Bryan Morse, and Scott Cohen. Geodesic graph cut for interactive image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[53] Brian Price, Bryan Morse, and Scott Cohen. Simultaneous foreground, background, and alpha estimation for image matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[54] A. Protiere and G. Sapiro. Interactive image segmentation via adaptive weighted distances. *IEEE Transactions on Image Processing*, 16(4):1046–1057, 2007.

[55] L. Jack Reese and William A. Barrett. Image editing with intelligent paint. In *Proceedings of the Conference of the European Association for Computer Graphics (Eurographics)*, volume 21, pages 714–724, 2002.

[56] C. Rhemann, C. Rother, and M. Gelautz. Improving color modeling for alpha matting. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 1155–1164, 2008.

[57] C. Rhemann, C. Rother, A. Rav-Acha, M. Gelautz, and T. Sharp. High resolution matting via interactive trimap segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[58] Christoph Rhemann, Carsten Rother, Jue Wang, Margrit Gelautz, Pushmeet Kohli, and Pamela Rott. A perceptually motivated online benchmark for image matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2009.

[59] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. In *Psychological Review*, volume 65, pages 386–408, 1958.

[60] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 309–314, 2004.

[61] M.A. Ruzon and C. Tomasi. Alpha estimation in natural images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 18–25, 2000.

[62] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 25(3):533–540, 2006.

[63] Thomas Schoenemann, Fredrik Kahl, and Daniel Cremers. Curvature regularity for region-based image segmentation and inpainting: A linear programming relaxation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2009.

[64] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 731–737, June 1997.

[65] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[66] D. Singaraju, L. Grady, and R. Vidal. P-Brush: Continuous valued MRFs with normed pairwise distributions for image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1303–1310, 2009.

[67] D. Singaraju and R. Vidal. Interactive image matting for multiple layers. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–7, June 2008. ISSN 1063-6919.

[68] Dheeraj Singaraju, Carsten Rother, and Christoph Rhemann. New appearance models for natural image matting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[69] Ali Kemal Sinop and Leo Grady. A seeded image segmentation framework unifying graph cuts and random walker which yields a new algorithm. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2007.

[70] Minsoo Suk and Tai-Hoon Cho. An object-detection algorithm based on the region-adjacency graph. *Proceedings of the IEEE*, 72(7):985–986, 1984.

[71] Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. Poisson matting. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 315–321, New York, NY, USA, 2004. ACM.

[72] P. Tang and L. Gao. Video object segmentation based on graph cut with dynamic shape prior constraint. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 1–4, 2008.

[73] A. Thayananthan, M. Iwasaki, and R. Cipolla. Principled fusion of high-level model and low-level cues for motion segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008.

[74] Fabrizio Tiburzi, Marcos Escudero, Jess Bescs, and Jos M. Martnez. A ground-truth for motion-based video-object segmentation. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, pages 17–20, 2008.

[75] Sara Vicente, Vladimir Kolmogorov, and Carsten Rother. Graph cut based image segmentation with connectivity priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[76] Luc Vincent and Pierre Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 13(6):583–598, 1991. ISSN 0162-8828.

[77] Nhat Vu and B.S. Manjunath. Shape prior segmentation of multiple objects with graph cuts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2008.

[78] Jinjun Wang, Wei Xu, Shenghuo Zhu, and Yihong Gong. Efficient video object segmentation by graph-cut. *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, pages 496–499, July 2007.

[79] Jue Wang, Maneesh Agrawala, and Michael Cohen. Soft scissors : An interactive tool for realtime high quality matting. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, pages 9:1–9:6, August 2007.

[80] Jue Wang, Pravin Bhat, R. Alex Colburn, Maneesh Agrawala, and Michael F. Cohen. Interactive video cutout. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 24(3):585–594, 2005. ISSN 0730-0301.

[81] Jue Wang and M.F. Cohen. Optimized color sampling for robust matting. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2007.

[82] Jue Wang and Michael F. Cohen. An iterative optimization approach for unified image segmentation and matting. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 936–943, Washington, DC, USA, 2005. IEEE Computer Society.

[83] Changjiang Yang, Ramani Duraiswami, Nail A. Gumerov, and Larry Davis. Improved fast Gauss transform and efficient kernel density estimation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 464–471, 2003.

[84] Pei Yin, Antonio Criminisi, John Winn, and Irfan Essa. Tree-based classifiers for bilayer video segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[85] Zhaozheng Yin and Robert Collins. Shape constrained figure-ground segmentation and tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[86] Xiaoru Yuan, Nan Zhang, Minh X. Nguyen, and Baoquan Chen. Volume cutout. *The Visual Computer (Special Issue of Pacific Graphics 2005)*, 21(8–10):745–754, 2005. ISSN 0178-2789.

[87] Yuanjie Zheng and Chandra Kambhamettu. Learning based digital matting. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2009.

[88] S.W. Zucker. Region growing: Childhood and adolescence. *Computer graphics and image processing*, 5(3):382–399, 1976.